

EDB Postgres AI for CloudNativePG Cluster

リリース *1.30.0*

EnterpriseDB Corporation

2026 年 08 月 21 日

目 次

第 1 章　はじめに **3**
1.1 Kubernetes の EDB Postgres に固有の機能 . . . 3
1.2 ライセンス . . . 4
1.3 サポートされているリリースと Kubernetes ディストリビューション . . . 4
1.3.1 マルチプルのアーキテクチャ . . . 4
1.4 このガイドについて . . . 4

第 2 章　EDB CloudNativePG Cluster release notes **7**
2.1 EDB CloudNativePG Cluster 1.30.0 release notes . . . 10
2.1.1 ハイライト . . . 11
2.1.2 機能 . . . 11
2.1.3 機能強化 . . . 12
2.1.4 セキュリティ修正 . . . 13
2.1.5 の変更 . . . 14
2.1.6 バグ修正 . . . 14
2.2 EDB CloudNativePG Cluster 1.29.2 release notes . . . 18
2.2.1 ハイライト . . . 18
2.2.2 機能強化 . . . 19
2.2.3 セキュリティ修正 . . . 19
2.2.4 の変更 . . . 20
2.2.5 バグ修正 . . . 20
2.3 EDB CloudNativePG Cluster 1.29.1 release notes . . . 24
2.3.1 ハイライト . . . 24
2.3.2 セキュリティ修正 . . . 25
2.3.3 の変更 . . . 25
2.3.4 バグ修正 . . . 25
2.4 EDB CloudNativePG Cluster 1.29.0 release notes . . . 26
2.4.1 ハイライト . . . 26
2.4.2 機能 . . . 27
2.4.3 機能強化 . . . 27
2.4.4 セキュリティ修正 . . . 29
2.4.5 の変更 . . . 29
2.4.6 バグ修正 . . . 30
2.5 EDB CloudNativePG Cluster 1.28.2 release notes . . . 31
2.5.1 ハイライト . . . 32
2.5.2 機能強化 . . . 32
2.5.3 セキュリティ修正 . . . 32
2.5.4 の変更 . . . 33
2.5.5 バグ修正 . . . 33
2.6 EDB CloudNativePG Cluster 1.28.1 release notes . . . 35

2.6.1 機能強化 . . . 35
2.6.2 バグ修正 . . . 36
2.7 EDB CloudNativePG Cluster 1.28.0 release notes . . . 38
2.7.1 ハイライト . . . 38
2.7.2 機能 . . . 38
2.7.3 機能強化 . . . 39
2.7.4 セキュリティ修正 . . . 40
2.7.5 の変更 . . . 40
2.7.6 バグ修正 . . . 41
2.8 EDB CloudNativePG Cluster 1.27.3 release notes . . . 42
2.8.1 機能強化 . . . 42
2.8.2 バグ修正 . . . 42
2.9 EDB CloudNativePG Cluster 1.27.2 release notes . . . 44
2.9.1 ハイライト . . . 44
2.9.2 の変更 . . . 44
2.10 EDB CloudNativePG Cluster 1.27.1 release notes . . . 45
2.10.1 機能強化 . . . 45
2.10.2 の変更 . . . 46
2.10.3 バグ修正 . . . 46
2.11 EDB CloudNativePG Cluster 1.27.0 release notes . . . 47
2.11.1 重要な変更 . . . 47
2.12 EDB CloudNativePG Cluster 1.26.3 release notes . . . 49
2.12.1 ハイライト . . . 49
2.12.2 の変更 . . . 49
2.13 EDB CloudNativePG Cluster 1.26.2 release notes . . . 49
2.13.1 の変更 . . . 49
2.14 EDB CloudNativePG Cluster 1.26.1 release notes . . . 49
2.14.1 の変更 . . . 50
2.15 EDB CloudNativePG Cluster 1.26.0 release notes . . . 50
2.15.1 の変更 . . . 50

第 3 章 GETTING_STARTED 51
3.1 Before you start . . . 51
3.1.1 Kubernetes の用語 . . . 51
3.1.2 PostgreSQL の用語 . . . 52
3.1.3 クラウド用語 . . . 53
3.1.4 次に何をするか . . . 54
3.2 Use cases . . . 54
3.2.1 ケース 1Kubernetes 内のアプリケーション . . . 54
3.2.2 ケース 2Kubernetes 外部のアプリケーション . . . 55
3.3 Architecture . . . 56
3.3.1 状態の同期 . . . 56
3.3.2 Kubernetes アーキテクチャ . . . 57
3.3.3 PostgreSQL アーキテクチャ . . . 60
3.3.4 Kubernetes クラスター全体のデプロイメント . . . 61
3.4 Installation and upgrades . . . 63
3.4.1 EDB サブスクリプショントークンの取得 . . . 63

3.4.2 Helm チャートを使用する . . . 63
3.4.3 オペレーターマニフェストを直接使用する . . . 63
3.4.4 EDB プルシークレットのインストール . . . 63
3.4.5 オペレーターのインストール . . . 64
3.4.6 kubectl の cnp プラグインを使用する . . . 64
3.4.7 スプレッドアップグレード . . . 65
3.4.8 インスタンスマネージャーのインプレース更新 . . . 66
3.4.9 バージョン間の互換性 . . . 66
3.4.10 1.30.0、1.29.2、または 1.28.4 へのアップグレード . . . 67
3.4.11 オペレーター側のパスワードエンコーディング CVE-2026-55765 . . . 67
3.4.12 search_path 強化 CVE-2026-55769 . . . 67
3.4.13 DatabaseRole リソースを使用した宣言的ロール管理 . . . 68
3.4.14 安全な予備選挙 . . . 68
3.5 Quickstart . . . 69
3.5.1 パート 1-ローカルの Kubernetes / Openshift ローカルプレイグラウンドのセットアップ 70
3.5.2 パート 2 CloudNativePG クラスターの EDB Postgres AI をインストールする . . . 71
3.5.3 パート 3 PostgreSQL クラスターを展開する . . . 71
3.5.4 パート 4 Prometheus と Grafana を使用してクラスターを監視する . . . 73
3.6 Preview Versions . . . 77
3.6.1 現在のプレビューバージョン . . . 77
3.6.2 リリース候補の目的 . . . 77
3.6.3 コミュニティへの参加: どのように支援するか . . . 77
3.6.4 使用アドバイザリー . . . 78

第 4 章 CONFIGURATION **79**
4.1 PostgreSQL Configuration . . . 79
4.1.1 postgresql セクション . . . 79
4.1.2 The pg_hba section . . . 85
4.1.3 Dynamic Shared Memory settings . . . 90
4.1.4 Dynamic Shared Memory settings . . . 90
4.2 動的 pg_hba アドレス解決のための名前付けポッドラベルセレクターを定義します。 # オペレーターは、一致するポッド IP を解決し、 ${podselector:NAME} # pg_hba ルールの参照を/32 IPv4 または/128 IPv6 マスクを使用して IP ごとに 1 行に展開します。 podSelectorRefs: - name: app-pods selector: matchLabels: app: myapp - name: モニタリングセレクター:matchLabels:role: モニタリング . . . 95
4.3 Operator configuration . . . 96
4.3.1 使用可能なオプション . . . 96
4.3.2 オペレーター構成マップの定義 . . . 99
4.3.3 オペレーターシークレットの定義 . . . 99
4.3.4 オペレーターを再起動して構成をリロードする . . . 100
4.3.5 受付 Webhook を使用しないデフォルトと検証 . . . 100
4.3.6 プロファイリングツール . . . 101
4.4 Instance Pod configuration . . . 102
4.4.1 予測される量 . . . 102
4.4.2 エフェメラルボリューム . . . 102
4.4.3 環境変数 . . . 103
4.5 Examples . . . 106

4.5.1 基本 . . . 106
4.5.2 セキュリティ . . . 107
4.5.3 バックアップ . . . 107
4.5.4 レプリカクラスター . . . 108
4.5.5 PostGIS . . . 108
4.5.6 管理対象ロール . . . 108
4.5.7 管理対象サービス . . . 109
4.5.8 宣言的テーブルスペース . . . 109
4.5.9 プーラー構成 . . . 109
4.5.10 宣言的なパブリケーションおよびサブスクリプションオブジェクトを介した論理レプリケーション . . . 109
4.5.11 Postgres データベースの宣言的管理 . . . 110

第 5 章 USING **141**

5.1 Bootstrap . . . 141
5.1.1 `bootstrap` セクション . . . 142
5.1.2 `externalClusters` セクション . . . 142
5.1.3 空のクラスターをブートストラップする `initdb` . . . 144
5.1.4 別のクラスターからのブートストラップ . . . 150
5.2 Image Catalog . . . 155
5.2.1 カタログのスコーピング . . . 156
5.2.2 カタログ構造 . . . 156
5.2.3 イメージボリューム拡張を使用したイメージカタログ . . . 158
5.2.4 EDB CloudNativePG クラスターカタログ . . . 159
5.3 Importing Postgres databases . . . 161
5.3.1 その仕組み . . . 161
5.3.2 `microservice` タイプ . . . 162
5.3.3 `monolith` タイプ . . . 164
5.3.4 実際の例 . . . 166
5.3.5 インポートの最適化 . . . 168
5.3.6 `pg_dump` および `pg_restore` 動作のカスタマイズ . . . 169
5.3.7 オンラインインポートとアップグレード . . . 170
5.4 Security . . . 171
5.4.1 コード . . . 171
5.4.2 コンテナ . . . 172
5.4.3 クラスター . . . 173
5.5 Postgres Instance Manager . . . 188
5.5.1 スタートアッププローブ . . . 188
5.5.2 Liveness Probe . . . 190
5.5.3 シャットダウン制御 . . . 194
5.5.4 フェイルオーバー . . . 195
5.5.5 ディスクフル障害 . . . 195
5.6 Scheduling . . . 195
5.6.1 ポッドアフィニティとアンチアフィニティ . . . 196
5.6.2 `nodeSelector` によるノード選択 . . . 198
5.6.3 許容範囲 . . . 198
5.6.4 PostgreSQL ワークロードの分離 . . . 198

5.7 Job scheduling with EDB Postgres AI for CloudNativePG . . . 199
5.7.1 スケジュールされたメンテナンスタスクの例 . . . 200
5.7.2 ビジネスクリティカルなジョブのベストプラクティス . . . 200
5.8 Resource management . . . 200
5.8.1 垂直ポッドオートスケーラー VPA との統合 . . . 202
5.9 Failure Modes . . . 204
5.9.1 自己修復 . . . 204
5.9.2 手動介入 . . . 204
5.10 Rolling updates . . . 205
5.10.1 自動更新 `unsupervised` . . . 206
5.10.2 手動更新 `supervised` . . . 207
5.11 Replication . . . 207
5.11.1 アプリケーションレベルのレプリケーション . . . 207
5.11.2 非常に成熟したテクノロジー . . . 207
5.11.3 ストリーミングレプリケーションのサポート . . . 208
5.11.4 同期レプリケーション . . . 209
5.11.5 同期レプリケーション非推奨 . . . 215
5.11.6 レプリケーションスロット . . . 217
5.12 Logical Replication . . . 221
5.12.1 出版物 . . . 222
5.12.2 サブスクリプション . . . 225
5.12.3 制限事項 . . . 228
5.12.4 論理レプリケーションを使用したライブ移行とメジャー Postgres アップグレードの例 229
5.13 Backup . . . 232
5.13.1 メインコンセプト . . . 232
5.13.2 使用可能なバックアップオプションの比較: オブジェクトストアとボリュームスナップショット . . . 234
5.13.3 スケジュールされたバックアップ . . . 235
5.13.4 オンデマンド バックアップ . . . 237
5.13.5 バックアップ方法 . . . 238
5.13.6 スタンバイからのバックアップ . . . 239
5.13.7 リテンションポリシー . . . 240
5.14 Recovery . . . 241
5.14.1 Barman Cloud プラグインを使用したオブジェクトストアからのリカバリー . . . 241
5.14.2 `VolumeSnapshot` オブジェクトからのリカバリー . . . 243
5.14.3 追加の考慮事項 . . . 245
5.14.4 ポイント インタイム リカバリー PITR . . . 245
5.14.5 アプリケーションデータベースの構成 . . . 250
5.14.6 リカバリーの内部的な仕組み . . . 251
5.14.7 バックアップセクションを使用したクラスターへの復元 . . . 251
5.15 WAL archiving . . . 252
5.15.1 プラグインベースのアーキテクチャ . . . 252
5.15.2 サポートされているプラグイン . . . 252
5.15.3 非推奨の通知 ネイティブ Barman Cloud . . . 252
5.15.4 アーカイブタイムアウトについて . . . 252
5.16 Service management . . . 252
5.16.1 デフォルトサービスの無効化 . . . 253

5.16.2 独自のサービスの追加 . . . 253
5.17 PostgreSQL Role management . . . 255
5.17.1 共存と優先 . . . 255
5.17.2 一般的なロール構成の注意事項 . . . 256
5.17.3 DatabaseRole リソース . . . 256
5.17.4 インライン管理対象ロール . . . 261
5.17.5 インライン管理ロールから DatabaseRole への移行 . . . 262
5.17.6 パスワード管理 . . . 263
5.18 PostgreSQL Database management . . . 268
5.18.1 主要なコンセプト . . . 269
5.18.2 予約されたデータベース名 . . . 270
5.18.3 調整とステータス . . . 270
5.18.4 データベースの削除 . . . 270
5.18.5 データベース内の拡張機能の管理 . . . 271
5.18.6 データベースでのスキーマの管理 . . . 273
5.18.7 データベースでの外部データラッパー FDW の管理 . . . 274
5.18.8 制限と注意事項 . . . 276
5.19 Tablespaces . . . 278
5.19.1 宣言的テーブルスペース . . . 278
5.19.2 宣言的テーブルスペースの使用 . . . 279
5.19.3 ストレージクラスとテーブルスペース . . . 280
5.19.4 テーブルスペースの所有権 . . . 281
5.19.5 バックアップ リカバリー . . . 282
5.19.6 レプリカクラスター . . . 282
5.19.7 一時テーブルスペース . . . 283
5.19.8 kubectl プラグインのサポート . . . 284
5.19.9 制限事項 . . . 284
5.20 Storage . . . 285
5.20.1 バックアップ リカバリー . . . 286
5.20.2 CloudNativePG クラスターの EDB Postgres AI のベンチマーク . . . 286
5.20.3 Persistent Volume Claim Claim PVC . . . 286
5.20.4 ストレージクラスを介した構成 . . . 286
5.20.5 PVC テンプレートを介した構成 . . . 287
5.20.6 WAL のボリューム . . . 288
5.20.7 テーブルスペースのボリューム . . . 289
5.20.8 ボリュームの拡張 . . . 289
5.20.9 ボリュームの削減 . . . 290
5.20.10 永続ボリュームの静的プロビジョニング . . . 291
5.20.11 ブロックストレージの考慮事項 Ceph/Longhorn . . . 292
5.21 Labels and Annotations . . . 293
5.21.1 事前定義されたラベル . . . 293
5.21.2 事前定義されたアノテーション . . . 296
5.21.3 前提条件 . . . 299
5.21.4 クラスターのメタデータの定義 . . . 299
5.21.5 現在の制限 . . . 300
5.22 Monitoring . . . 300
5.22.1 モニタリングインスタンス . . . 300

5.22.2 CloudNativePG クラスターオペレーターの EDB Postgres AI のモニタリング 319
5.22.3 エクスポートされたメトリックを検査する方法 321
5.22.4 補助リソース 323
5.22.5 OpenShift でのモニタリング 323
5.23 Logging 336
5.23.1 クラスターログ 337
5.23.2 オペレーターログ 337
5.23.3 PostgreSQL ログ 337
5.23.4 PG 監査ログ 338
5.23.5 EDB Audit log 340
5.23.6 その他のログ 342
5.24 Certificates 342
5.24.1 サーバー証明書 344
5.24.2 サーバー CA シークレット 344
5.24.3 サーバー TLS シークレット 344
5.24.4 サーバー代替 DNS 名 344
5.24.5 クライアント証明書 344
5.24.6 クライアント CA シークレット 344
5.24.7 クライアント `streaming_replica` 証明書 344
5.24.8 サーバー証明書 345
5.24.9 例 345
5.24.10 Cert-manager の例 346
5.24.11 クライアント証明書 347
5.24.12 `streaming_replica` クライアント証明書のカスタマイズ 348
5.24.13 Cert-manager の例 349
5.24.14 cluster-example-cert-manager.yaml 350
5.25 Client TLS/SSL connections 353
5.25.1 新しい証明書の発行 353
5.25.2 TLS 証明書を介した接続のテスト 354
5.25.3 TLS プロトコルバージョンについて 356
5.26 Connecting from an application 356
5.26.1 DNS 解決 357
5.26.2 環境変数 357
5.26.3 シークレット 357
5.27 Connection Pooling 358
5.27.1 アーキテクチャ 358
5.27.2 クイックスタート 358
5.27.3 リソースのライフサイクルのプール化 360
5.27.4 セキュリティ 360
5.27.5 認証 361
5.27.6 ポッドテンプレート 362
5.27.7 PgBouncer 画像 364
5.27.8 高可用性 HA 367
5.27.9 PgBouncer 構成オプション 367
5.27.10 モニタリング 369
5.27.11 ロギング 379
5.27.12 接続の一時停止 379

5.27.13 制限事項 . . . 380
5.28 Replica clusters . . . 380
5.28.1 基本概念 . . . 381
5.28.2 レプリカクラスターのセットアップ . . . 383
5.28.3 分散トポロジ . . . 383
5.28.4 スタンドアロンレプリカクラスター . . . 387
5.28.5 遅延レプリカ . . . 390
5.29 Kubernetes upgrade and maintenance . . . 391
5.29.1 定期的なアップデートの重要性 . . . 392
5.29.2 クラスター内のメンテナンス操作 . . . 392
5.29.3 一時的な PostgreSQL クラスターの低下 . . . 392
5.29.4 ポッドの中断予算 . . . 392
5.29.5 開発またはテストに使用される PostgreSQL クラスター . . . 393
5.29.6 ノードのメンテナンスウィンドウ . . . 393
5.30 PostgreSQL upgrades . . . 395
5.30.1 マイナーバージョンのアップグレード . . . 395
5.30.2 メジャーバージョンのアップグレード . . . 395
5.30.3 オフラインインプレースメジャーアップグレード . . . 395
5.31 EDB Postgres AI for CloudNativePG Cluster Plugin . . . 401
5.31.1 インストール . . . 401
5.31.2 使用する . . . 404
5.31.3 K9s との統合 . . . 428
5.31.4 プラグインに必要な権限 . . . 428
5.32 Automated failover . . . 430
5.32.1 安全な予備選挙 . . . 431
5.32.2 遅延フェイルオーバー . . . 434
5.32.3 ノードレベルの障害の検出 . . . 434
5.32.4 Failover Quorum クォーラムベースのフェールオーバー . . . 435
5.33 Fencing . . . 440
5.33.1 インスタンスをフェンスする方法 . . . 440
5.33.2 フェンシングのリフティング方法 . . . 441
5.33.3 フェンシングの仕組み . . . 442
5.34 Declarative hibernation . . . 442
5.34.1 ハイバネーション . . . 442
5.34.2 水分補給 . . . 443
5.35 PostGIS . . . 444
5.35.1 PostGIS クラスターに関する基本的な概念 . . . 444
5.35.2 PostGIS で新しい PostgreSQL クラスターを作成する . . . 445
5.36 Container Image Requirements . . . 448
5.36.1 イメージタグの要件 . . . 448
5.37 Image Volume Extensions . . . 449
5.37.1 公式拡張機能のイメージとカタログ . . . 450
5.37.2 その仕組み . . . 451
5.37.3 PostgreSQL クラスターからの拡張機能の削除 . . . 455
5.37.4 高度なトピック . . . 456
5.38 Custom Pod Controller . . . 462
5.38.1 PVC のサイズ変更 . . . 462

5.38.2 プライマリインスタンスとレプリカ . . . 462
5.38.3 PVC のコヒーレンス . . . 463
5.38.4 ローカルストレージ、リモートストレージ、およびデータベースのサイズ . . . 463
5.39 Networking . . . 464
5.39.1 オペレーターのクロス名前空間ネットワークポリシー . . . 464
5.39.2 クロスクラスターネットワーキング . . . 465
5.40 Benchmarking . . . 465
5.40.1 pgbench . . . 466
5.40.2 fio . . . 467

第 6 章 EDB ENHANCEMENTS 471
6.1 Free evaluation . . . 471
6.1.1 PostgreSQL を使用した評価 . . . 471
6.2 License and License keys . . . 471
6.2.1 会社レベルのライセンスキー . . . 471
6.2.2 クラスターレベルのライセンスキー . . . 472
6.2.3 クラスターレベルでのライセンスキーシークレット . . . 473
6.2.4 ライセンスの有効期限が切れるとどうなるか . . . 473
6.3 EDB private container registry . . . 473
6.3.1 `docker login` を使用した例 . . . 473
6.3.2 使用可能なイメージタグの参照 . . . 474
6.4 Red Hat OpenShift . . . 476
6.4.1 PostgreSQL ワークロードのノードの予約 . . . 477
6.4.2 クラスターサービスバージョン CSV . . . 479
6.4.3 EDB プライベートレジストリへのアクセスの保証 . . . 481
6.4.4 クラスター全体のインストール . . . 484
6.4.5 `oc` を使用したクラスター全体のインストール . . . 487
6.4.6 OpenShift CLI `oc` のインストール . . . 488
6.4.7 カスタムリソース定義 CRD . . . 488
6.4.8 サービスアカウント . . . 489
6.4.9 CloudNativePG™クラスター CRD の EDB Postgres AI のクラスターロール . . . 490
6.5 Iron Bank . . . 499
6.5.1 Iron Bank から EDB {{name.short}}とオペランドイメージを取得する . . . 500
6.5.2 Iron Bank イメージを使用した{{name.short}}オペレーターのインストール . . . 500
6.5.3 IronBank イメージを使用した EPAS オペランドを使用したクラスターの展開 . . . 501
6.6 Transparent Data Encryption (TDE) . . . 502
6.6.1 パスフレーズシークレット . . . 502
6.6.2 カスタムパスフレーズコマンド . . . 504
6.6.3 カスタム wrap/unwrap コマンド . . . 504
6.6.4 HashiCorp Vault を使用した例 . . . 505
6.7 Add-ons . . . 508
6.7.1 外部バックアップアダプター . . . 508
6.7.2 カステン . . . 517
6.7.3 ヴェレロ . . . 518

第 7 章 REFERENCE 521
7.1 Operator capability levels . . . 521

7.1.1 レベル 1 基本インストール . . . 521
7.1.2 レベル 2 シームレスなアップグレード . . . 528
7.1.3 レベル 3 完全なライフサイクル . . . 529
7.1.4 レベル 4 深い洞察 . . . 534
7.1.5 レベル 5 自動操縦 . . . 535
7.2 Frequently Asked Questions (FAQ) . . . 536
7.2.1 Kubernetes での PostgreSQL の実行 . . . 536
7.2.2 高可用性 . . . 539
7.2.3 データベース管理 . . . 540
7.3 Troubleshooting . . . 543
7.3.1 始める前に . . . 543
7.3.2 最初の一歩 . . . 543
7.3.3 バックアップはありますか？ . . . 544
7.3.4 緊急バックアップ . . . 544
7.3.5 ログ . . . 544
7.3.6 オペレーター情報 . . . 545
7.3.7 クラスター情報 . . . 547
7.3.8 ポッド情報 . . . 548
7.3.9 バックアップ情報 . . . 550
7.3.10 ストレージ情報 . . . 550
7.3.11 ノード情報 . . . 551
7.3.12 条件 . . . 551
7.3.13 ネットワーキング . . . 552
7.3.14 PostgreSQL コアダンプ . . . 553
7.3.15 プロファイリングデータの視覚化と分析 . . . 554
7.3.16 いくつかの既知の問題 . . . 555
7.4 API Reference - v1.30.0 . . . 558
7.4.1 Packages . . . 558
7.4.2 postgresql.k8s.enterprisedb.io/v1 . . . 558
7.5 Backup and Recovery . . . 684
7.6 CNCF Projects Integrations . . . 685
7.6.1 External Secrets . . . 685
7.6.2 Cilium . . . 689

第 8 章 APPENDIX **695**
8.1 Appendix A - Backup on volume snapshots . . . 695
8.1.1 標準のボリュームスナップショットについて . . . 695
8.1.2 要件 . . . 695
8.1.3 ボリュームスナップショットバックアップを構成する方法 . . . 696
8.1.4 ホットおよびコールドバックアップ . . . 697
8.1.5 ボリュームスナップショットオブジェクトの永続化 . . . 699
8.1.6 バックアップボリュームのスナップショットの期限 . . . 699
8.1.7 ボリュームスナップショットバックアップの例 . . . 701
8.2 Appendix B - Backup on object stores . . . 702
8.2.1 共通オブジェクトストア . . . 703
8.2.2 WAL アーカイブ . . . 703
8.2.3 リテンション ポリシー . . . 704

8.2.4 圧縮アルゴリズム . . . 705
8.2.5 バックアップオブジェクトのタグ付け . . . 706
8.2.6 backup および WAL コマンドの追加オプション . . . 706
8.2.7 オブジェクトストアからのリカバリー . . . 707
8.3 Appendix C - Common object stores for backups . . . 708
8.3.1 AWS S3 . . . 709
8.3.2 Azure Blob Storage . . . 712
8.3.3 Google Cloud Storage . . . 714
8.4 CNPG-I . . . 716
8.4.1 なぜ CNPG-I なのか？ . . . 716
8.4.2 プラグインの登録 . . . 716
8.4.3 プラグインを使用する . . . 719
8.4.4 コミュニティプラグイン . . . 720

CloudNativePG クラスターオペレーターの EDB Postgres AI は、 CloudNativePG™ に基づくフォークです。EDB Postgres Advanced Server や IBM Power、OpenShift などの追加サポート対象プラットフォームを使用して、Oracle との互換性などの追加価値を提供します。EDB によって設計、開発、およびサポートされており、ネイティブストリーミングレプリケーションを使用して、プライマリ/スタンバイアーキテクチャで可用性の高い Postgres データベースクラスターのライフサイクルを完全にカバーしています。

CloudNativePG クラスターの EDB Postgres AI は、2021 年 2 月 4 日に一般提供されました。以前のバージョンは、GA リリース前に一部の顧客が利用できるようになりました。

注釈

オペレーターの名前がクラウドネイティブ PostgreSQL から変更されました。基になるコンポーネントとリソースが変更されていないため、クラウドネイティブ PostgreSQL の既存のユーザーは何の変更も経験しません。

- 高可用性のための Kubernetes API 統合
- CloudNativePG™は `postgresql.cnpg.io/v1` API バージョンを使用します
- CloudNativePG クラスターの EDB Postgres AI は `postgresql.k8s.enterprisedb.io/v1` API バージョンを使用します
- フェイルオーバーを介した自己修復とレプリカの自動再作成
- スケールアップ/ダウン機能を備えたキャパシティ管理
- 定期メンテナンスに向けた計画的なスイッチオーバー
- 読み取り専用および読み取り/書き込みの Kubernetes サービス定義
- Postgres のマイナーバージョンとオペレーターのアップグレードのローリングアップデート
- 継続的なバックアップとポイント インタイム リカバリー
- PgBouncer を使用した接続プーリング
- すぐに使える統合メトリックエクスポーター
- 複数の Kubernetes クラスターにわたる PostgreSQL レプリケーション
- WAL ファイルの別冊

第1章　はじめに

1.1 Kubernetes の EDB Postgres に固有の機能

- Long Term Support
- IBM とのパートナーシップを介した IBM Power および z/Linux のサポート
- EDB Postgres Advanced Sever を介した Oracle compatibility
- EDB Postgres Advanced Server を介して Transparent Data Encryption (TDE)
- Kasten および Velero/OADP を使用したコールドバックアップのサポート
- サードパーティの Kubernetes バックアップツール用の汎用アダプター

トライアルサブスクリプションの一部として *evaluate EDB Postgres AI for CloudNativePG Cluster for free* を行うことができます。実稼働環境で CloudNativePG クラスターの EDB Postgres AI を使用するには、有効な EDB サブスクリプションが必要です。

> **注釈**
>
> *Operator Capability Levels model* に基づいて、ユーザーは CloudNativePG クラスターオペレーターの EDB Postgres AI から**「レベル V - 自動パイロット」**機能セットを期待できます。

EDB は、CloudNativePG クラスターの EDB Postgres AI の長期サポート LTS バージョンを毎年宣言することに取り組んでいます。1.28 が現在の LTS バージョンです。各 LTS バージョンはメンテナンスリリースを受け取り、最初のリリースから標準の 6 か月を超えてさらに 12 か月、合計 18 か月サポートされます。

たとえば、CloudNativePG クラスターの EDB Postgres AI の v1.25 は、2024 年 12 月にリリースされました。LTS バージョンとして宣言されたため、2026 年 6 月まで合計 18 か月間サポートされます。

現在サポートされているバージョンとそのステータスのリストについては、Platform Compatibility を参照してください。

新機能を利用するためにオペレーターの最新バージョンに定期的にアップグレードすることをお勧めしますが、LTS バージョンを使用すると、追加の安定性を必要とするお客様は、アップグレード前に 12〜18 か月間同じバージョンを使用できます。

1.2 ライセンス

CloudNativePG クラスターの EDB Postgres AI は、PostgreSQL、EDB Postgres Extended、および EDB Postgres Advanced サーバーの両方で動作し、EDB End User License Agreement の下で利用できます。

トライアルサブスクリプションの一部として *evaluate EDB Postgres AI for CloudNativePG Cluster for free* ができます。実稼働環境で CloudNativePG クラスターの EDB Postgres AI を使用するには、有効な EDB サブスクリプションが必要です。

1.3 サポートされているリリースと Kubernetes ディストリビューション

EDB でサポートされている CloudNativePG クラスターの EDB Postgres AI のマイナーリリースのリストについては、Platform Compatibility を参照してください。

ページ。ここでは、各 Kubernetes ディストリビューションとバージョン、および EOL 日付がサポートされているものを見つけることもできます。

1.3.1 マルチプルのアーキテクチャ

CloudNativePG Cluster Operator コンテナイメージの EDB Postgres AI は、次のプラットフォームのマルチアーチフォーマットをサポートしています。`linux/amd64`、`linux/arm64`、`linux/ppc64le`、`linux/s390x`。

> **警告**
>
> CloudNativePG クラスターの EDB Postgres AI では、Kubernetes クラスター内のすべてのノードが同じ CPU アーキテクチャである必要があります。したがって、ハイブリッド CPU アーキテクチャの Kubernetes クラスターはサポートされていません。さらに、EDB は、OpenShift でのみ `linux/ppc64le` および `linux/s390x` アーキテクチャをサポートしています。

Postgres の次のバージョンは、現在バージョン 1.28 でサポートされています。

- PostgreSQL 14〜18
- EDB Postgres Advanced 14 - 18
- EDB Postgres 拡張 14〜18

PostgreSQL および EDB Postgres Advanced は、次のプラットフォームで利用できます。`linux/amd64`、`linux/ppc64le`、`linux/s390x`。さらに、PostgreSQL は `linux/arm64` でもサポートされています。 EDB Postgres Extended は、`linux/amd64` でのみサポートされています。EDB は、OpenShift でのみ `linux/ppc64le` および `linux/s390x` アーキテクチャのオペランドイメージをサポートしています。

1.4 このガイドについて

Quickstart の指示に従って、Kind または Minikube を使用してローカル Kubernetes クラスターで CloudNativePG クラスターの EDB Postgres AI をテストします。

Kubernetes と PostgreSQL の基本的な用語に慣れていない場合は、*Before you start* を参照してください。

> **注釈**
>
> このガイドでは主に Kubernetes を取り上げていますが、すべての概念を OpenShift に拡張できます。

は、カナダ PostgreSQL コミュニティ協会の商標または登録商標であり、その許可を得て使用しています。

第2章　EDB CloudNativePG Cluster release notes

EDB Postgres for Kubernetes ドキュメントでは、マイナーリリースとパッチを含む EDB Postgres for Kubernetes のメジャーバージョンについて説明しています。リリースノートは、各リリースの新機能に関する情報を提供します。マイナーまたはパッチリリースで導入された新機能の場合、コンテンツには、機能を導入したリリースも示されます。

Version	Release date	Upstream merges
1.30.0、*1.29.2*、または *1.28.4* へのアップグレード	29 Jun 2026	Upstream *1.30.0*、*1.29.2*、または *1.28.4* へのアップグレード
1.30.0、*1.29.2*、または *1.28.4* へのアップグレード	29 Jun 2026	Upstream *1.30.0*、*1.29.2*、または *1.28.4* へのアップグレード
EDB CloudNativePG Cluster 1.29.1 release notes	08 May 2026	Upstream *EDB CloudNativePG Cluster 1.29.1 release notes*
EDB CloudNativePG Cluster 1.29.0 release notes	31 Mar 2026	Upstream *EDB CloudNativePG Cluster 1.29.0 release notes*
1.30.0、*1.29.2*、または *1.28.4* へのアップグレード	29 Jun 2026	Upstream *1.30.0*、*1.29.2*、または *1.28.4* へのアップグレード
1.28.3	08 May 2026	Upstream 1.28.3
EDB CloudNativePG Cluster 1.28.2 release notes	31 Mar 2026	Upstream *EDB CloudNativePG Cluster 1.28.2 release notes*
EDB CloudNativePG Cluster 1.28.1 release notes	10 Feb 2026	Upstream *EDB CloudNativePG Cluster 1.28.1 release notes*
EDB CloudNativePG Cluster 1.28.0 release notes	16 Dec 2025	Upstream *EDB CloudNativePG Cluster 1.28.0 release notes*
1.27.4	31 Mar 2026	Upstream 1.27.4
EDB CloudNativePG Cluster 1.27.3 release notes	10 Feb 2026	Upstream *EDB CloudNativePG Cluster 1.27.3 release notes*
EDB CloudNativePG Cluster 1.27.2 release notes	16 Dec 2025	Upstream *EDB CloudNativePG Cluster 1.27.2 release notes*
EDB CloudNativePG Cluster 1.27.1 release notes	24 Oct 2025	Upstream *EDB CloudNativePG Cluster 1.27.1 release notes*
EDB CloudNativePG Cluster 1.27.0 release notes	19 Aug 2025	Upstream *EDB CloudNativePG Cluster 1.27.0 release notes*
EDB CloudNativePG Cluster 1.26.3 release notes	16 Dec 2025	Upstream *EDB CloudNativePG Cluster 1.26.3 release notes*
EDB CloudNativePG Cluster 1.26.2 release notes	24 Oct 2025	Upstream *EDB CloudNativePG Cluster 1.26.2 release notes*

次のページに続く

表 1 – 前のページからの続き

Version	Release date	Upstream merges
EDB CloudNativePG Cluster 1.26.1 release notes	25 Jul 2025	Upstream *EDB CloudNativePG Cluster 1.26.1 release notes*
EDB CloudNativePG Cluster 1.26.0 release notes	23 May 2025	Upstream *EDB CloudNativePG Cluster 1.26.0 release notes*
1.25.9	29 Jun 2026	None
1.25.8	08 May 2026	None
1.25.7	02 Apr 2026	None
1.25.6	10 Feb 2026	None
1.25.5	16 Dec 2025	None
1.25.4	24 Oct 2025	Upstream 1.25.4
1.25.3	25 Jul 2025	Upstream 1.25.3
1.25.2	23 May 2025	Upstream 1.25.2
1.25.1	04 Mar 2025	Upstream 1.25.1
1.25.0	23 Dec 2024	Upstream 1.25.0
1.24.4	23 May 2025	Upstream 1.24.4
1.24.3	04 Mar 2025	Upstream 1.24.3
1.24.2	23 Dec 2024	Upstream 1.24.2
1.24.1	18 Oct 2024	Upstream 1.24.1
1.24.0	26 Aug 2024	Upstream 1.24.0
1.23.6	23 Dec 2024	Upstream 1.23.6
1.23.5	18 Oct 2024	Upstream 1.23.5
1.23.4	26 Aug 2024	Upstream 1.23.4
1.23.3	01 Aug 2024	Upstream 1.23.3
1.23.2	13 Jun 2024	Upstream 1.23.2
1.23.1	29 Apr 2024	Upstream 1.23.1
1.23.0	24 Apr 2024	Upstream 1.23.0
1.22.11	25 Jul 2025	None
1.22.10	23 May 2025	None
1.22.9	04 Mar 2025	None
1.22.8	23 Dec 2024	None
1.22.7	18 Oct 2024	None
1.22.6	26 Aug 2024	Upstream 1.22.6
1.22.5	01 Aug 2024	Upstream 1.22.5
1.22.4	13 Jun 2024	Upstream 1.22.4
1.22.3	24 Apr 2024	Upstream 1.22.3
1.22.2	22 Mar 2024	Upstream 1.22.2
1.22.1	02 Feb 2024	Upstream 1.22.1
1.22.0	22 Dec 2023	Upstream 1.22.0
1.21.6	13 Jun 2024	Upstream 1.21.6
1.21.5	24 Apr 2024	Upstream 1.21.5
1.21.4	22 Mar 2024	Upstream 1.21.4
1.21.3	02 Feb 2024	Upstream 1.21.3
1.21.2	22 Dec 2023	Upstream 1.21.2

次のページに続く

表 1 – 前のページからの続き

Version	Release date	Upstream merges
1.21.1	08 Nov 2023	Upstream 1.21.1
1.21.0	18 Oct 2023	Upstream 1.21.0
1.20.6	02 Feb 2024	Upstream 1.20.6
1.20.5	22 Dec 2023	Upstream 1.20.5
1.20.4	08 Nov 2023	Upstream 1.20.4
1.20.3	18 Oct 2023	Upstream 1.20.3
1.20.2	27 Jul 2023	Upstream 1.20.2
1.20.1	13 Jun 2023	Upstream 1.20.1
1.20.0	27 Apr 2023	Upstream 1.20.0
1.19.6	08 Nov 2023	Upstream 1.19.6
1.19.5	18 Oct 2023	Upstream 1.19.5
1.19.4	27 Jul 2023	Upstream 1.19.4
1.19.3	13 Jun 2023	Upstream 1.19.3
1.19.2	27 Apr 2023	Upstream 1.19.2
1.19.1	20 Mar 2023	Upstream 1.19.1
1.19.0	14 Feb 2023	Upstream 1.19.0
1.18.13	13 Jun 2024	None
1.18.12	24 Apr 2024	None
1.18.11	22 Mar 2024	None
1.18.10	02 Feb 2024	None
1.18.9	22 Dec 2023	None
1.18.8	08 Nov 2023	None
1.18.7	18 Oct 2023	None
1.18.6	27 Jul 2023	None
1.18.5	13 Jun 2023	Upstream 1.18.5
1.18.4	27 Apr 2023	Upstream 1.18.4
1.18.3	20 Mar 2023	Upstream 1.18.3
1.18.2	14 Feb 2023	Upstream 1.18.2
1.18.1	21 Dec 2022	Upstream 1.18.1
1.18.0	14 Nov 2022	Upstream 1.18.0
1.17.5	20 Mar 2023	Upstream 1.17.5
1.17.4	14 Feb 2023	Upstream 1.17.4
1.17.3	21 Dec 2022	Upstream 1.17.3
1.17.2	14 Nov 2022	Upstream 1.17.2
1.17.1	07 Oct 2022	Upstream 1.17.1
1.17.0	06 Sep 2022	Upstream 1.17.0
1.16.5	21 Dec 2022	Upstream 1.16.4
1.16.4	14 Nov 2022	Upstream 1.16.4
1.16.3	07 Oct 2022	Upstream 1.16.3
1.16.2	06 Sep 2022	Upstream 1.16.2
1.16.1	12 Aug 2022	Upstream 1.16.1
1.16.0	07 Jul 2022	Upstream 1.16.0
1.15.5	07 Oct 2022	Upstream 1.15.5

次のページに続く

表 1 – 前のページからの続き

Version	Release date	Upstream merges
1.15.4	06 Sep 2022	Upstream 1.15.4
1.15.3	12 Aug 2022	Upstream 1.15.3
1.15.2	07 Jul 2022	Upstream 1.15.2
1.15.1	27 May 2022	Upstream 1.15.1
1.15.0	21 Apr 2022	Upstream 1.15.0
1.14.0	25 Mar 2022	NA
1.13.0	17 Feb 2022	NA
1.12.0	11 Jan 2022	NA
1.11.0	15 Dec 2021	NA
1.10.0	11 Nov 2021	NA
1.9.2	15 Oct 2021	NA
1.9.1	30 Sep 2021	NA
1.9.0	28 Sep 2021	NA
1.8.0	13 Sep 2021	NA
1.7.1	11 Aug 2021	NA
1.7.0	28 Jul 2021	NA
1.6.0	12 Jul 2021	NA
1.5.1	11 Jun 2021	NA
1.5.0	17 Jun 2021	NA
1.4.0	18 May 2021	NA
1.3.0	23 Apr 2021	NA
1.2.1	06 Apr 2021	NA
1.2.0	31 Mar 2021	NA
1.1.0	03 Mar 2021	NA
1.0.0	04 Feb 2021	NA
0.8.0	29 Jan 2021	NA
0.7.0	31 Dec 2020	NA
0.6.0	04 Dec 2020	NA
0.5.0	20 Nov 2020	NA
0.4.0	05 Nov 2020	NA
0.3.0	25 Sep 2020	NA
0.2.0	11 Aug 2020	NA
0.1.0	03 Apr 2020	NA
0.0.1	05 Mar 2020	NA

2.1 EDB CloudNativePG Cluster 1.30.0 release notes

リリース日 2026 年 6 月 29 日

CloudNativePG™クラスターの EDB Postgres ® AI のこのリリースには、次のものが含まれます。

2.1.1 ハイライト

ネイティブツリー内 Barman Cloud サポートの非推奨通知を更新して、1.30.0 ではなく Kubernetes 1.31.0 の EDB Postgres で削除されることを反映しています。ユーザーは引き続き Barman Cloud プラグインに移行することをお勧めします。 #11083

`cluster` 参照は、`Database`、`Pooler`、`Publication`、`Subscription`、および `ScheduledBackup` リソースで不変になりました。別のクラスターでこれらのオブジェクトのいずれかをポイントすることには、明確に定義されたセマンティクスがなく、以前はコントローラを一貫性のない状態のままにしました。更新は、CEL 検証ルールを介して API サーバーで拒否されるようになりました。#10743

2.1.2 機能

Description	Addresses
安全なプライマリ選挙のためのプライマリ `Lease` プライマリプロモーションをシリアル化するミューテックスとして機能する Kubernetes *Lease* **オブジェクトクラスターにちなんで命名されました。インスタンスマネージャーは、プライマリとして機能する前にリースを保持する必要があり、クリーンなときにリリースしますシャットダウンするため、レプリカは完全な *TTL* を待たずに昇格できます。タイミングは、新しい**.spec.primaryLease`スタンザを介して構成できます。リースはプロモーションゲートであり、フェンスではありません。一次隔離は引き続きフェンシングを担当します。	#10627
宣言的ロール管理のための `DatabaseRole` CRD `Cluster`の `.spec.managed.roles`スタンザでインライン宣言するのではなく、スタンドアロンの Kubernetes オブジェクトとして PostgreSQL ロールを管理する `DatabaseRole`カスタムリソースを導入しました。各ロールは、独自のライフサイクル、ステータス、および RBAC を取得します。これにより、GitOps ワークフローに適合し、ロール定義を所有するアプリケーションの横に存在します。この仕様は、インラインメソッドと同じ `RoleConfiguration`構造を再利用するため、ロールの移行は、スタンザを独自のマニフェストに移動することです。 *databaseRoleReclaimPolicy* **`フィールド** `*retain*、デフォルト、または `delete`は、リソースが削除され、永続ボリュームをミラーリングしたときにロールに何が起こるかを制御します。	#6155
宣言的ロールの TLS クライアント証明書 `DatabaseRole`に `clientCertificate`ブロックを含めて、クラスターのクライアント CA によって署名され、`<databaserole-name>-client-cert`シークレットに保存される TLS クライアント証明書をオペレーターが自動的に生成および更新できるようになりました。これにより、パスワードなしの PostgreSQL `cert`認証が有効になります。シークレットは、機能が無効になるか、`DatabaseRole`が削除されるとクリーンアップされます。	#10896

次のページに続く

表 2 – 前のページからの続き

Description	Addresses
イメージカタログを介した PgBouncer イメージ管理 `Pooler`リソースは、新しい`spec.pgbouncer.imageCatalogRef`フィールドを介して`ImageCatalog`または`ClusterImageCatalog`のエントリを参照できるようになり、PgBouncer イメージ管理を一元化できます。カタログエントリが更新されると、参照しているすべての`Poolers`が自動的にリコンサイルされ、仕様を変更せずに新しいイメージをロールアウトします。解決された画像は`status.image`でレポートされ、新しい`status.phase` *active*、*paused*、*inactive*、または`failed`も`kubectl get pooler`の`Phase`列として表示され、ライフサイクルを概要します。	#10568

2.1.3 機能強化

Description	Addresses
PostgreSQL 19 の `pg_upgrade`に追加された拡張パスのサポートに基づいて、イメージボリューム拡張を使用するクラスターで PostgreSQL 19 以降への `pg_upgrade`インプレースメジャーアップグレードを有効にしました。 アップグレード中に`Job`、ソース-およびターゲットバージョンの拡張イメージはサイドバイサイドでマウントされるため、古いサーバーはライブラリを保持し、失敗したアップグレードはきれいに元に戻ります。	#10366
`.spec.monitoring.tls.enabled`を介して`Pooler`メトリックエンドポイントの TLS サポートを追加しました。 有効にすると、メトリックサーバーは HTTPS で提供され、`.spec.pgbouncer.clientTLSSecret`から証明書とキーを再利用し、ハンドシェイクのたびにリロードして、再起動なしのローテーションをサポートします。生成された`PodMonitor`は、それに応じて`https`をスクレイピングします。	#10466
ラベルセレクターを`Cluster`スケールサブリソース`status.selector`に追加し、`Cluster`を垂直ポッドオートスケーラー VPA および水平ポッドオートスケーラー HPA の有効な`targetRef`にします。`Cluster`をインスタンスポッドにマッピングできるようになりました。 @sebv004 による寄稿。	#8996
プライマリポッドが kubelet パースペクティブから`Ready`であるが、オペレーターの`/pg/status`チェックが失敗し、フェイルオーバーが遅延される場合、オペレーターは`Cluster`で`Warning` `PrimaryStatusCheckFailed`イベントを発行するようになり、ユーザーに`kubectl describe cluster`を介して遅延を可視化します。	#10509
`ENABLE_WEBHOOK_NAMESPACE_SUFFIX`フラグを追加しました。これは、オペレーターの Webhook 構成名に`-<OPERATOR_NAMESPACE>`のサフィックスを付けて、複数のオペレーターインスタンスが同じクラスターで共存できるようにします。 オペレーターは、これらの構成を検索するだけです。ユーザーはそれらを作成および保守する必要があります。 @maxlengdell による寄稿。	#10420

次のページに続く

表 3 – 前のページからの続き

Description	Addresses
オペレーターは、ポッドがロールされたときに CNPG-i プラグインを自動的にリロードするようになりました。`EndpointSlices`バッキングプラグイン`Services`を監視し、新しいポッドが`Ready`になるとプラグインを使用してすべてのクラスターを再エンキューするため、アップグレードされたプラグインは、次回の再同期。	#10836
インスタンスのシリアル番号は、常にグローバルカウンタをインクリメントするのではなく、既存のインスタンス名の中で最も低い空きスロットを再利用することにより割り当てられるようになりました。 ポッドと PVC の名前は、インスタンスの再作成にわたって安定したままですたとえば、後に再作成されたインスタンスノードドレインは同じ名前で戻ってきます)、削除されたインスタンスによって解放されたシリアルは再利用されます。新しい`Initialized`クラスター条件は、クラスターが最初のブートストラップを完了したかどうかをレポートし、`status.latestGeneratedNode`は非推奨です。もう書き込まれませんが、下位互換性のために CRD に保存されます。	#10548
デフォルトと検証は、アドミッション Webhook が使用できない場合、または障害を無視するように構成されている場合のフォールバックとして、調整中に実行されるようになりました。そのため、オペレーターは無効または不完全な仕様をリコンサイルしません。 欠落しているデフォルトは直接適用され、検証失敗後でサイレントに失敗するのではなく、リソースステータスで表面化されます。	#10874

2.1.4 セキュリティ修正

Description	Addresses
`CVE-2026-55769` / `GHSA-x8c2-3p4r-v9r6` オペレーター発行の接続での`search_path`ピニング データベース所有者は`public`スキーマにオーバーロードされたビルトイン演算子を植え、`search_path`を変更して、クラスタースーパーユーザーとして実行されるオペレーターのイントロスペクションプローブが解決されるようにすることができます。`pg_catalog`の前にあるこれらのオーバーロードは、`COPY ... FROM PROGRAM`を介してポッド内 RCE につながる可能性がある`CWE-426`特権エスカレーションチェーン`CVE-2018-1058`と同じクラスです。オペレーターは、プールされたすべての接続に`search_path = pg_catalog, public, pg_temp`を固定するため、起動メッセージで出荷され、テナント制御のデフォルトより優先されます。	#10774 , *#10743*

次のページに続く

表 4 – 前のページからの続き

Description	Addresses
`GHSA-7qwx-x8ff-3px9` 認証されたオペレーターからインスタンスマネージャーへの呼び出し インスタンスマネージャーのリモートウェブサーバーは、オペレーター専用の制御エンドポイントの認証ではなくネットワークの分離に依存しているため、ポッドのステータスポートに到達できる当事者はそれらを呼び出し、バックアップオーケストレーションと WAL アーカイブを中断し、運用メタデータを読み取ります。アップグレードエンドポイントは SHA-256 固定されているため、任意のコードの実行は許可されません。インスタンスマネージャーは、一致する証明書を提示しない機密エンドポイントへの要求を拒否します。この強化はバックポートされません。以前のリリースでは、`NetworkPolicy`でステータスポートを制限し続ける必要があります。	#10579 , *#10743*
`CVE-2026-55765` / `GHSA-w3gf-xc94-wvmj` オペレーターサイドの SCRAM-SHA-256 パスワードエンコーディング オペレーターは現在 SCRAM-SHA-256 が `CREATE`/`ALTER ROLE ... PASSWORD`を発行する前にクリアテキストロールパスワードをエンコードするため、リテラル PostgreSQL が解析およびそのような拡張機能 `pg_stat_statements`または `pgaudit`がキャプチャできるように) は、平文シークレットではなく SCRAM 検証ツールです。事前ハッシュされた MD5 または SCRAM 値は変更せずに転送され、シークレットごとのアノテーション `k8s.enterprisedb.io/passwordPassthrough: "enabled"`はオプトアウトします。	#10724 , *#10743*

2.1.5 の変更

Description	Addresses
Kubernetes 1.36 のサポートを追加しました。	#10900
デフォルトの PostgreSQL バージョンを 18.4 に更新しました。	#10719
パブリッククラウドプロバイダーでオペレーターのテストに使用される Kubernetes バージョンを更新しました。	#10720 , #10563 , #11033

2.1.6 バグ修正

Description	Addresses
クラスターがレプリカに降格された後、古いプライマリ側のステータスを報告する宣言的な `Database`、*Publication*、および `Subscription`オブジェクトを修正しました。コントローラはレプリカの状態を再チェックし、`Cluster`を監視するようにして、降格がすぐに検出されるようにします。	#10871

次のページに続く

表 6 – 前のページからの続き

Description	Addresses
対応する`Job`および PVC が作成される前にインスタンスシリアルカウンタが進められたことが原因で発生した非シークエンシャルのポッド名`-1`、`-3`などを修正した。バンプは、これらのリソースが存在した場合にのみ持続されるようになりました。	#10491
レプリカクラスターの`Terminating`でスタックしている`Database`、`Publication`、または`Subscription`の削除を修正しました。ファイナライザーリコンサイラーの前にレプリカゲートが実行され、ファイナライザーはリリースされません。 レプリカでは、PostgreSQL オブジェクトは次のとおりです。プライマリクラスターに任せます。	#10853
生き残った CR が所有する PostgreSQL オブジェクトをドロップする`delete`再利用ポリシーと競合する重複`Database`または`Subscription`を修正しました。ドロップは、記録された調整に基づいてゲートされるようになりました。	#10870
キャッシュされたシークレットバージョンが無効化されず、パスワードが再適用されなかったため、スーパーユーザーアクセスが無効になってから再有効化された後、`postgres`スーパーユーザーがロックアウトされたままになる問題を修正しました。 @mhartmann-jaconi による診断。	#10834
バックアップが`running`に到達する前に、バックアップを実行しているインスタンスマネージャーが再起動された場合たとえば、オペレーターのアップグレードに続くインプレースアップグレードによって、バックアップが`started`フェーズでスタックする問題を修正した。調整が再スケジュールされるため、失われたセッションが検出されます。	#10859
SPDY フォールバックと WebSocket をネゴシエートするように`exec`/`attach`ストリーミングを修正し、SPDY を削除した Kubernetes バージョンと、WebSocket exec アップグレードを拒否する OpenShift などのプラットフォームとの互換性を復元しました。 @bartscheers による寄稿。	#10876 , #10933
**同時に`Backup`オブジェクトが競合した場合のリソースリークを修正しました。バックアップは厳密に作成時の順序で実行されるようになったため、既に実行中のバックアップが新しいバックアップにプリエンプトされることはなく、そのレプリケーションスロットと PostgreSQL セッションがプライマリで孤立しなくなりました。* @GabriFedi97 による寄稿。	#10747
参照されたシークレットを取得できなかった場合に、PostgreSQL ロールのパスワードをクリアするロール調整を修正しました。シークレットが利用可能になるまでロールは変更されず、アクションごとのエラーが集計されて可視性が向上します。	#10053
メトリックエクスポーターのセットアップエラー—一般的にコントローラーとの重複キーレースがロールバックした`streaming_replica`作成とウェッジレプリカ参加が発生したブートストラップ障害を修正しました。 メトリックエクスポーターステップが別のトランザクションで実行されるようになりました。 @BlaiseAntony による寄稿。	#10749
`Backup`は作成されたが、そのステータスパッチが着地しなかったときに発生する`ScheduledBackup`コントローラループを修正しました。コントローラは、次の反復に`AlreadyExists`でループする代わりに、既存の`Backup`を採用するようになりました。	#10612

次のページに続く

表 6 – 前のページからの続き

Description	Addresses
`Cluster`が削除された`Pooler`をリコンサイルするときの nil ポインターパニックを修正しました。	#10667
ブートストラップログ処理を修正して、すべての名前付けログパイプ`postgres`、*postgres.csv*、および`postgres.json`が`WithActiveInstance`中にコンシューマーを取得し、名前付けパイプの代わりに通常のファイルが作成されないようにします。	#10043
アドレスを角かっこで囲むことにより、無効な IPv6 URL の生成を修正しました。 @Infinoid による寄稿。	#10682
構成が`enabled: false`を設定している場合、外部クラスタープラグインがアクティブなものとして扱われるのを修正しました。	#10932
復元ジョブが古い`Cluster`プライマリがまだ記録されておらず、タイムラインがまだ設定されていないを読み取り、その`.history`ファイルをスプリットブレインガードによって拒否される可能性があるオブジェクトストアからのブートストラップリカバリ中のレースを修正しました。 これが発生すると、リカバリはベースバックアップのタイムラインで停止し、後のタイムラインでコミットされたトランザクションをサイレントにドロップしました。クラスタータイムラインが設定されていないときに履歴ファイルを使用できるようになりました。 @dennispidun による寄稿。	#10818
インスタンスの PVC を削除すると、インスタンスが永久にスタックしたままになる可能性があるレースを修正しました。WAL PVC がまだ終了中にデータ PVC が削除された場合、オペレーターは終了ボリュームにバインドされたインスタンスを再作成し、ポッドをスケジュール不可のままにして、それ以上すべてをブロックします reconciliation. オペレーターは、インスタンスを再作成または再接続する前に終端 PVC が完全に削除されるのを待機し、ログラインとクラスターフェーズを介して待機を表面化します。	#11017
サーバーとクライアント CA が同じシークレットデフォルトに解決する場合、クラスター作成中のキャッシュレースを修正しました。古いインフォーマーキャッシュは、冗長`Create`をトリガーしましたが、`AlreadyExists`で障害が発生し、クラスターが`Unable to create required cluster objects`でスタックしたままになる可能性があります。 オペレーターは、名前が一致する場合、既に取得した CA シークレットを再利用するようになりました。	#10989
他のブートストラップ方法で既に適用されているのと同じプリフライトディレクトリチェックを強制することにより、既存の`PGDATA`たとえばレプリカポッドの再起動後を上書きまたは失敗する`pg_basebackup`ブートストラップパスを修正しました。これにより、静的にプロビジョニングされた PVC がサイレントに上書きされるのを防ぎます。	#11006
リコンサイル後のプラグインフックがエラーを返したときの`Healthy`とプラグイン障害フェーズ間の`Cluster`フェーズのフラッピングを修正しました。`Healthy`フェーズは、成功したリコンシリエーションの最後のステップとして登録されるようになったため、プラグインエラーで終了するループは`Healthy`. @GabriFedi97 による寄稿。	#10421

次のページに続く

表 6 – 前のページからの続き

Description	Addresses
外部サーバーのシークレットがローテーションされた後の古い証明書データと部分的な読み取りを修正しましたたとえば、2 つの証明書から 1 つに縮小する CA バンドル。ファイルはアトミックに書き込まれるようになったため、libpq は常に古い値または新しい値を読み取ります。 a mix. @Anand-240 による寄稿。	#10975
短い名前の代わりにプラグイン `Service` FQDN を使用するようにプラグインの接続を修正し、クラスターレベルのプロキシがポッドに自動的に注入される際の障害を回避します。 @kdautrey による寄稿。	#10921
要求ごとの `Cluster` 作成/更新検証 Webhook メッセージからの過剰なオペレーターログノイズが修正され、`info` ではなく `debug` にログに記録されるようになりました。	#10984
有効な PostgreSQL パラメーター名ではないキーを受け入れる `spec.postgresql.parameters` を修正しました。これにより、`postgresql.conf` に任意の指示子が注入される可能性があります。キー名が Webhook によって検証されるようになりました。	#11029
既存のクラスターで WAL アーカイバープラグインが有効になっている場合のスイッチオーバーデッドロックを修正しました。`primaryUpdateMethod: switchover` では、クリーンな降格にはまだ欠けているアーカイバーサイドカーが必要であるため、プライマリをロールアウトできませんでした。 オペレーターは今は、プライマリポッドを再作成するため、サイドカーが挿入され、アーカイブが再開されます。このチェックは、 Barman Cloud プラグインなどのネイティブサイドカー `restartPolicy: Always` を使用した init コンテナとしてアーカイバーを注入するプラグインもカバーしています。	#11032 , #11059
インスタンス作成ジョブがバックオフ制限を使い果たしたときに、クラスターが `Setting up primary` に無期限に留まる問題を修正した。オペレーターは、ターミナルジョブの障害を検出し、クラスターを回復不可としてマークし、失敗したジョブに名前を付け、そのログをポイントします。	#11035
データ PVC が作成された後、初期化ジョブが開始される前に、ステータスパッチが競合するファーストプライマリブートストラップデッドロックを修正しました。孤立した保留中の PVC がインスタンスとしてカウントされ、ブートストラップゲートをブロックしたままになります。 PVC 状態リコンサイラーは、割り当てられたシリアルを再利用してブートストラップジョブを再作成します。	#11039
検証なしでファイルシステムパスに結合される外部クラスター名とシークレットセレクター参照を修正し、インスタンスマネージャーが接続マテリアルをダンプするときに `..` コンポーネントまたはパスセパレーターが外部シークレットディレクトリをエスケープします。これらの値は検証 Webhook で拒否され、書き込みサイトで再チェックされます。 @r0binak によってレポートされます。	#11045

次のページに続く

表 6 – 前のページからの続き

Description	Addresses
`pending`で永久にスタックするバックアップを修正しました。コンカレントバックアップゲートはリコンサイルごとに実行され、インスタンスマネージャーによって非同期に書き込まれた既に完了したフェーズを上書きする可能性があります。 ゲートは、バックアップフェーズ中にのみ実行されるようになりました。はまだ未設定または`pending`です。	#11056
古いキャッシュにより、既にプロビジョニングされたスナップショットを再作成し、`AlreadyExists`で失敗したときに、失敗として永続的にマークされる宣言的`VolumeSnapshot`バックアップを修正しました。 オペレーターは、既存のスナップショットが次の場合に衝突を許容するようになりました。このバックアップのラベルを作成し、それを採用します。外部スナップショットとの衝突は引き続きエラーとして表面化します。	#11071
スナップショットが既にプロビジョニングされている場合でも、ファイナライズステップ中に一時的なインスタンスマネージャー接続エラーたとえば、短いポッドネットワークの中断からのダイヤルタイムアウトで破棄されるボリュームスナップショットバックアップを修正しました。このようなネットワークエラーは、ターミナルとして扱われる代わりに再試行されるようになりました。	#11069
トークンの保存と移行メタデータのクリーンアップの間でリコンサイルが再キューイングされたときに、`status.demotionToken`を失うレプリカスイッチオーバーを修正しましたたとえば、不安定な Webhook に対して失敗したクリーンアップパッチ。空の変更なしトークンは、格納されている値をパッチして戻されません。	#11075
Windows での`kubectl cnp psql`を修正しました。実行は Unix 専用のシステムコールに依存し、「Windows ではサポートされていません」で失敗しました。 Windows は子プロセスとして`kubectl exec`を起動するようになりました。 @Utkarsh-sharma47 による寄稿。	#10972
ロググループごとのタイマーが解放されない、ビジークラスターでの`kubectl cnp logs -f`の無制限のメモリリークを修正しました。タイマーは反復にわたって再利用されるようになりました。 @Anand-240 による寄稿。	#10976

2.2 EDB CloudNativePG Cluster 1.29.2 release notes

リリース日 2026 年 6 月 29 日

CloudNativePG™クラスターの EDB Postgres ® AI のこのリリースには、次のものが含まれます。

2.2.1 ハイライト

ネイティブツリー内 Barman Cloud サポートの非推奨通知を更新して、1.30.0 ではなく Kubernetes 1.31.0 の EDB Postgres で削除されることを反映しています。ユーザーは引き続き Barman Cloud プラグインに移行することをお勧めします。 #11083

`cluster` 参照は、`Database` 、`Pooler` 、`Publication` 、`Subscription` 、および `ScheduledBackup` リソースで不変になりました。別のクラスターでこれらのオブジェクトのいずれかをポイントすることには、明

確に定義されたセマンティクスがなく、以前はコントローラを一貫性のない状態のままにしました。更新は、CEL 検証ルールを介して API サーバーで拒否されるようになりました。#10743

2.2.2 機能強化

Description	Addresses
PostgreSQL 19 の `pg_upgrade`に追加された拡張パスのサポートに基づいて、イメージボリューム拡張を使用するクラスターで PostgreSQL 19 以降への `pg_upgrade`インプレースメジャーアップグレードを有効にしました。 アップグレード中に `Job`、ソース-およびターゲットバージョンの拡張イメージはサイドバイサイドでマウントされるため、古いサーバーはライブラリを保持し、失敗したアップグレードはきれいに元に戻ります。	#10366
ラベルセレクターを `Cluster`スケールサブリソース `status.selector`に追加し、`Cluster`を垂直ポッドオートスケーラー VPA および水平ポッドオートスケーラー HPA の有効な `targetRef`にします。`Cluster`をインスタンスポッドにマッピングできるようになりました。 @sebv004 による寄稿。	#8996
プライマリポッドが kubelet パースペクティブから `Ready`であるが、オペレーターの `/pg/status`チェックが失敗し、フェイルオーバーが遅延される場合、オペレーターは `Cluster`で `Warning` `PrimaryStatusCheckFailed`イベントを発行するようになり、ユーザーに `kubectl describe cluster`を介して遅延を可視化します。	#10509
オペレーターは、ポッドがロールされたときに CNPG-i プラグインを自動的にリロードするようになりました。`EndpointSlices`バッキングプラグイン `Services`を監視し、新しいポッドが `Ready`になるとプラグインを使用してすべてのクラスターを再エンキューするため、アップグレードされたプラグインは、次回の再同期。	#10836

2.2.3 セキュリティ修正

Description	Addresses
`CVE-2026-55769` / `GHSA-x8c2-3p4r-v9r6` オペレーター発行の接続での `search_path`ピニング データベース所有者は `public`スキーマにオーバーロードされたビルトイン演算子を植え、`search_path`を変更して、クラスタースーパーユーザーとして実行されるオペレーターのイントロスペクションプローブが解決されるようにすることができます。 `pg_catalog`の前にあるこれらのオーバーロードは、`COPY ... FROM PROGRAM`を介してポッド内 RCE につながる可能性がある `CWE-426`特権エスカレーションチェーン `CVE-2018-1058`と同じクラスです。オペレーターは、プールされたすべての接続に `search_path = pg_catalog, public, pg_temp`を固定するため、起動メッセージで出荷され、テナント制御のデフォルトより優先されます。	#10774 , *#10743*

次のページに続く

表 8 – 前のページからの続き

Description	Addresses
`CVE-2026-55765` / `GHSA-w3gf-xc94-wvmj` オペレーターサイドの SCRAM-SHA-256 パスワードエンコーディング オペレーターは現在 SCRAM-SHA-256 が `CREATE`/`ALTER ROLE ... PASSWORD`を発行する前にクリアテキストロールパスワードをエンコードするため、リテラル PostgreSQL が解析およびそのような拡張機能 `pg_stat_statements`または `pgaudit`がキャプチャできるように) は、平文シークレットではなく SCRAM 検証ツールです。事前ハッシュされた MD5 または SCRAM 値は変更せずに転送され、シークレットごとのアノテーション `k8s.enterprisedb.io/passwordPassthrough: "enabled"`はオプトアウトします。	#10724 , *#10743*

2.2.4 の変更

Description	Addresses
Kubernetes 1.36 のサポートを追加しました。	#10900
デフォルトの PostgreSQL バージョンを 18.4 に更新しました。	#10719
パブリッククラウドプロバイダーでオペレーターのテストに使用される Kubernetes バージョンを更新しました。	#10720 , #10563 , #11033

2.2.5 バグ修正

Description	Addresses
有効な PostgreSQL パラメーター名ではないキーを受け入れる `spec.postgresql.parameters`を修正しました。これにより、`postgresql.conf`に任意の指示子が注入される可能性があります。キー名が Webhook によって検証されるようになりました。	#11029
クラスターがレプリカに降格された後、古いプライマリ側のステータスを報告する宣言的な `Database`、*Publication*、および `Subscription`オブジェクトを修正しました。コントローラはレプリカの状態を再チェックし、`Cluster`を監視するようにして、降格がすぐに検出されるようにします。	#10871
対応する `Job`および PVC が作成される前にインスタンスシリアルカウンタが進められたことが原因で発生した非シークエンシャルのポッド名 `-1`、`-3`などを修正した。バンプは、これらのリソースが存在した場合にのみ持続されるようになりました。	#10491

次のページに続く

表 10 – 前のページからの続き

Description	Addresses
既存のクラスターで WAL アーカイバープラグインが有効になっている場合のスイッチオーバーデッドロックを修正しました。`primaryUpdateMethod: switchover`では、クリーンな降格にはまだ欠けているアーカイバーサイドカーが必要であるため、プライマリをロールアウトできませんでした。 オペレーターは今は、プライマリポッドを再作成するため、サイドカーが挿入され、アーカイブが再開されます。このチェックは、 Barman Cloud プラグインなどのネイティブサイドカー `restartPolicy: Always`を使用した init コンテナとしてアーカイバーを注入するプラグインもカバーしています。	#11032 , #11059
インスタンス作成ジョブがバックオフ制限を使い果たしたときに、クラスターが `Setting up primary`に無期限に留まる問題を修正した。オペレーターは、ターミナルジョブの障害を検出し、クラスターを回復不可としてマークし、失敗したジョブに名前を付け、そのログをポイントします。	#11035
レプリカクラスターの `Terminating`でスタックしている `Database`、`Publication`、または `Subscription`の削除を修正しました。ファイナライザーリコンサイラーの前にレプリカゲートが実行され、ファイナライザーはリリースされません。 レプリカでは、PostgreSQL オブジェクトは次のとおりです。プライマリクラスターに任せます。	#10853
生き残った CR が所有する PostgreSQL オブジェクトをドロップする `delete`再利用ポリシーと競合する重複`Database`または`Subscription`を修正しました。ドロップは、記録された調整に基づいてゲートされるようになりました。	#10870
キャッシュされたシークレットバージョンが無効化されず、パスワードが再適用されなかったため、スーパーユーザーアクセスが無効になってから再有効化された後、`postgres`スーパーユーザーがロックアウトされたままになる問題を修正しました。 @mhartmann-jaconi による診断。	#10834
バックアップが `running`に到達する前に、バックアップを実行しているインスタンスマネージャーが再起動された場合たとえば、オペレーターのアップグレードに続くインプレースアップグレードによって、バックアップが `started`フェーズでスタックする問題を修正した。調整が再スケジュールされるため、失われたセッションが検出されます。	#10859
SPDY フォールバックと WebSocket をネゴシエートするように `exec` / `attach`ストリーミングを修正し、SPDY を削除した Kubernetes バージョンと、WebSocket exec アップグレードを拒否する OpenShift などのプラットフォームとの互換性を復元しました。 @bartscheers による寄稿。	#10876 , #10933
**同時に `Backup`オブジェクトが競合した場合のリソースリークを修正しました。バックアップは厳密に作成時の順序で実行されるようになったため、既に実行中のバックアップが新しいバックアップにプリエンプトされることはなく、そのレプリケーションスロットと PostgreSQL セッションがプライマリで孤立しなくなりました。* @GabriFedi97 による寄稿。	#10747
参照されたシークレットを取得できなかった場合に、PostgreSQL ロールのパスワードをクリアするロール調整を修正しました。シークレットが利用可能になるまでロールは変更されず、アクションごとのエラーが集計されて可視性が向上します。	#10053

次のページに続く

表 10 – 前のページからの続き

Description	Addresses
メトリックエクスポーターのセットアップエラー一般的にコントローラーとの重複キーレースがロールバックした `streaming_replica`作成とウェッジレプリカ参加が発生したブートストラップ障害を修正しました。 メトリックエクスポーターステップが別のトランザクションで実行されるようになりました。 @BlaiseAntony による寄稿。	#10749
`Backup`は作成されたが、そのステータスパッチが着地しなかったときに発生する`ScheduledBackup`コントローラループを修正しました。コントローラは、次の反復に`AlreadyExists`でループする代わりに、既存の`Backup`を採用するようになりました。	#10612
`Cluster`が削除された`Pooler`をリコンサイルするときの nil ポインターパニックを修正しました。	#10667
ブートストラップログ処理を修正して、すべての名前付けログパイプ`postgres`、*postgres.csv*、および`postgres.json`が`WithActiveInstance`中にコンシューマーを取得し、名前付けパイプの代わりに通常のファイルが作成されないようにします。	#10043
アドレスを角かっこで囲むことにより、無効な IPv6 URL の生成を修正しました。 @Infinoid による寄稿。	#10682
構成が`enabled: false`を設定している場合、外部クラスタープラグインがアクティブなものとして扱われるのを修正しました。	#10932
復元ジョブが古い`Cluster`プライマリがまだ記録されておらず、タイムラインがまだ設定されていないを読み取り、その`.history`ファイルをスプリットブレインガードによって拒否される可能性があるオブジェクトストアからのブートストラップリカバリ中のレースを修正しました。 これが発生すると、リカバリはベースバックアップのタイムラインで停止し、後のタイムラインでコミットされたトランザクションをサイレントにドロップしました。クラスタータイムラインが設定されていないときに履歴ファイルを使用できるようになりました。 @dennispidun による寄稿。	#10818
サーバーとクライアント CA が同じシークレットデフォルトに解決する場合、クラスター作成中のキャッシュレースを修正しました。古いインフォーマーキャッシュは、冗長`Create`をトリガーしましたが、`AlreadyExists`で障害が発生し、クラスターが`Unable to create required cluster objects`でスタックしたままになる可能性があります。 オペレーターは、名前が一致する場合、既に取得した CA シークレットを再利用するようになりました。	#10989
他のブートストラップ方法で既に適用されているのと同じプリフライトディレクトリチェックを強制することにより、既存の`PGDATA`たとえばレプリカポッドの再起動後を上書きまたは失敗する`pg_basebackup`ブートストラップパスを修正しました。これにより、静的にプロビジョニングされた PVC がサイレントに上書きされるのを防ぎます。	#11006
リコンサイル後のプラグインフックがエラーを返したときの`Healthy`とプラグイン障害フェーズ間の`Cluster`フェーズのフラッピングを修正しました。`Healthy`フェーズは、成功したリコンシリエーションの最後のステップとして登録されるようになったため、プラグインエラーで終了するループは`Healthy`. @GabriFedi97 による寄稿。	#10421

次のページに続く

表 10 – 前のページからの続き

Description	Addresses
外部サーバーのシークレットがローテーションされた後の古い証明書データと部分的な読み取りを修正しましたたとえば、2 つの証明書から 1 つに縮小する CA バンドル。ファイルはアトミックに書き込まれるようになったため、libpq は常に古い値または新しい値を読み取ります。 a mix. @Anand-240 による寄稿。	#10975
短い名前の代わりにプラグイン `Service` FQDN を使用するようにプラグインの接続を修正し、クラスターレベルのプロキシがポッドに自動的に注入される際の障害を回避します。 @kdautrey による寄稿。	#10921
要求ごとの `Cluster` 作成/更新検証 Webhook メッセージからの過剰なオペレーターログノイズが修正され、`info` ではなく `debug` にログに記録されるようになりました。	#10984
データ PVC が作成された後、初期化ジョブが開始される前に、ステータスパッチが競合するファーストプライマリブートストラップデッドロックを修正しました。孤立した保留中の PVC がインスタンスとしてカウントされ、ブートストラップゲートをブロックしたままになります。 PVC 状態リコンサイラーは、割り当てられたシリアルを再利用してブートストラップジョブを再作成します。	#11039
検証なしでファイルシステムパスに結合される外部クラスター名とシークレットセレクター参照を修正し、インスタンスマネージャーが接続マテリアルをダンプするときに `..` コンポーネントまたはパスセパレーターが外部シークレットディレクトリをエスケープします。これらの値は検証 Webhook で拒否され、書き込みサイトで再チェックされます。 @r0binak によってレポートされます。	#11045
`pending` で永久にスタックするバックアップを修正しました。コンカレントバックアップゲートはリコンサイルごとに実行され、インスタンスマネージャーによって非同期に書き込まれた既に完了したフェーズを上書きする可能性があります。 ゲートは、バックアップフェーズ中にのみ実行されるようになりました。はまだ未設定または `pending` です。	#11056
古いキャッシュにより、既にプロビジョニングされたスナップショットを再作成し、`AlreadyExists` で失敗したときに、失敗として永続的にマークされる宣言的 `VolumeSnapshot` バックアップを修正しました。 オペレーターは、既存のスナップショットが次の場合に衝突を許容するようになりました。このバックアップのラベルを作成し、それを採用します。外部スナップショットとの衝突は引き続きエラーとして表面化します。	#11071
スナップショットが既にプロビジョニングされている場合でも、ファイナライズステップ中に一時的なインスタンスマネージャー接続エラーたとえば、短いポッドネットワークの中断からのダイヤルタイムアウトで破棄されるボリュームスナップショットバックアップを修正しました。このようなネットワークエラーは、ターミナルとして扱われる代わりに再試行されるようになりました。	#11069

次のページに続く

表 10 – 前のページからの続き

Description	Addresses
トークンの保存と移行メタデータのクリーンアップの間でリコンサイルが再キューイングされたときに、`status.demotionToken`を失うレプリカスイッチオーバーを修正しましたたとえば、不安定な Webhook に対して失敗したクリーンアップパッチ。空の変更なしトークンは、格納されている値をパッチして戻されません。	#11075
Windows での `kubectl cnp psql`を修正しました。実行は Unix 専用のシステムコールに依存し、「Windows ではサポートされていません」で失敗しました。 Windows は子プロセスとして `kubectl exec`を起動するようになりました。 @Utkarsh-sharma47 による寄稿。	#10972
ロググループごとのタイマーが解放されない、ビジークラスターでの `kubectl cnp logs -f` の無制限のメモリリークを修正しました。タイマーは反復にわたって再利用されるようになりました。 @Anand-240 による寄稿。	#10976

2.3 EDB CloudNativePG Cluster 1.29.1 release notes

リリース日 2026 年 5 月 8 日

CloudNativePG™クラスターの EDB Postgres ® AI のこのリリースには、次のものが含まれます。

2.3.1 ハイライト

``CVE-2026-44477`` / ``GHSA-423p-g724-fr39`` メトリックエクスポーター特権エスカレーション。メトリックエクスポーターは `postgres` スーパーユーザーとして認証されなくなりました。 `pg_monitor` 特権のみを持つ専用の `cnp_metrics_exporter` ロールを使用し、低特権のデータベースユーザーが PostgreSQL スーパーユーザーを取得するチェーンを閉じます。

GHSA-423p-g724-fr39

アップグレードの影響ユーザー所有テーブルを読み取るカスタムモニタリングクエリー、または `PUBLIC CONNECT` が取り消されたデータベースに対して `target_databases: '*'` を使用するカスタムモニタリングクエリーは、 `cnp_metrics_exporter` に対する明示的な `GRANT` ステートメントが必要です。**カスタムクエリーの特権と安全性** を参照

と **メトリックエクスポーターロールの手動作成**

モニタリング文書で。

レプリカクラスターの場合、ソースプライマリクラスターを使用するレプリカクラスターの前に、ソースプライマリクラスターをアップグレードします。`cnp_metrics_exporter` ロールはソースプライマリで作成され、ダウンストリームに複製されます。最初にアップグレードされたレプリカクラスターは、ソースプライマリがアップグレードされるまで、不足しているロールをスクレイピングします。上記のリンクのある manual-recovery セクションでは、レプリカクラスターについてもカバーしています。

2.3.2 セキュリティ修正

Description	Addresses
デフォルトのモニタリングクエリーのスキーマ修飾カタログ参照 すべての `pg_catalog`オブジェクトを明示的に修飾することにより、同梱のモニタリング構成とドキュメントサンプルを強化しました。非修飾の参照は、`search_path`を介して解決されます。データベースユーザーはこれを操作して、ビルトインオブジェクトをシャドウできます。	#10576
検出可能な SBOM と来歴証明書 オペレーターコンテナイメージに添付される SBOM および SLSA 来歴が OCI 1.1 リファラー仕様に従うようになったため、標準のレジストリツールとサプライチェーンスキャナーでそれらを自動的に検出できます。	#10601
`github.com/jackc/pgx/v5`の CVE 修正 v5.9.2 にバンプして、*CVE-2026-33816 pgproto3* **`のメモリセーフティと** *`GHSA-j88v-2chj-qfwx* 単純プロトコルのドル引用符で囲まれた文字列処理を介した SQL インジェクションのアップストリームの修正をピックアップします。	#10437 , #10499
Go ランタイムでの CVE 修復 `crypto/x509`、*crypto/tls*、*net/http*、および `net`のアップストリームの修正をピックアップするために Go 1.26.3 でビルドされています。CVE-2026-32280、CVE-2026-32281、CVE-2026 -33810、CVE-2026-33814、CVE-2026-33811、CVE-2026-39825)。	#10463 , #10647
ビルドパイプラインの強化 Go 1.26.3 バンプは、CVE-2026-42501 `cmd/go`モジュールチェックサム検証にも対処し、リリースビルド中のサプライチェーンのエクスポージャを削減します。影響を受けるコードパスには、実行中のオペレーターから到達できません。	#10647

2.3.3 の変更

Description	Addresses
TLS ピア検証を `VerifyPeerCertificate`から `VerifyConnection`に切り替えました。これは、完了したハンドシェイクごとに実行されます前者は、再開された TLS 1.3 セッションでスキップされます。 セッションの再開は、現在 Kubernetes の EDB Postgres で有効になっていないため、これは目に見える効果はありませんが、セッションキャッシュが後で導入された場合の将来性のある検証です。	#10478

2.3.4 バグ修正

Description	Addresses
以前のプライマリがプライマリラベルを保持したフェールオーバーウィンドウを修正しました。 以前のプライマリがフェールオーバー中に一時的なネットワーク分割の後などに戻った場合、`-rw`サービスがそれにルーティングし続け、レプリカが再接続できる場合があります、そしてコミットされた書き込みは`pg_rewind`に失われました。古いプライマリには`unhealthy`のラベルが付けられ、フェールオーバー中にサービストラフィックから分離されます。	#10409
プライマリをホストするノードが到達不能になったときにトリガーされないフェールオーバーを修正しました。 オペレーターは、ポッドの`Ready`条件 kubelet が報告を停止したときにノードコントローラによって`False`に反転されるを読み取るようになりました。そのシナリオでは`True`として古い。スプリアスフェールオーバーガード #10445 と組み合わせると、Kubernetes 自分自身がポッドを準備ができていないことをマークした場合にのみフェールオーバーがトリガーされます。	#10448
プライマリの HTTP ステータスエンドポイントでの一時的な障害によって発生した擬似フェイルオーバーを修正しました。	#10445
PostgreSQL 構成値のバックスラッシュと制御文字のエスケープを修正しました。 以前は、`log_line_prefix`のようなパラメーターのこのような文字は、構成ファイルを破損するか、実行時にサイレントに削除される可能性がありました。	#10515
各引数をシェルクォートするように`restore_command`コンストラクトを修正しました。 空白を含む`destinationPath`たとえば`s3://my bucket/wal`などの値は、POSIX シェルによってワード分割され、別の引数として WAL 復元ツールに渡されました。	#10518
入場 Web フックの `recoveryTarget`検証が強化されました。 `targetXID`は負でない 32 ビット整数であり、`targetName`は 64 バイト未満で ASCII 制御文字が含まれていない必要があります。不正な値は、後の PostgreSQL 復旧中に失敗するのではなく、受付時に拒否されます。	#10565
残りの`pgsql_tmp*`ディレクトリがデータディレクトリに存在する場合、スナップショットのリストアが失敗する問題を修正しました。	#10447
PVC ストレージサイズとリソース要求が同時に変更されたときに発生するデッドロックを修正しました。	#10427

2.4 EDB CloudNativePG Cluster 1.29.0 release notes

リリース日 2026 年 3 月 31 日

CloudNativePG™クラスターの EDB Postgres ® AI のこのリリースには、次のものが含まれます。

2.4.1 ハイライト

ネイティブツリー内 Barman Cloud サポートの非推奨通知を更新して、Kubernetes 1.30.0 の EDB Postgres で 1.29.0 ではなく 1.30.0 で削除されることを反映しています。ユーザーは引き続き Barman Cloud プラグインに移行することをお勧めします。 #10167

2.4.2 機能

Description	Addresses
イメージカタログの PostgreSQL 拡張機能は、 <code>ImageCatalog</code>機能を拡張して、PostgreSQL 拡張機能をサポートしました。これにより、ユーザーはカタログ内の拡張機能固有のイメージを定義および管理でき、カスタマイズされた PostgreSQL ビルドの展開が簡素化されます。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9781">#9781
ポッドセレクターを介した動的ネットワークアクセス制御では、 <code>podSelectorRefs</code>の宣言的定義を導入して、<code>pg_hba.conf</code>ルールを動的に管理します。ラベルセレクターを使用してクライアントポッドを識別することにより、オペレーターは自動的にエフェメラル IP アドレスを解決し、それに応じて PostgreSQL ホストベースの認証ルールを更新します。これにより、同じ名前空間内の許可されたワークロードのみがデータベースに接続できるようになり、手動の IP 管理または静的 CIDR レンジの必要がなくなります。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9781">#9781
共有<code>ServiceAccount</code>サポートは、オプションの<code>serviceAccountName</code>フィールドを<code>Cluster</code>と<code>Pooler</code>仕様の両方に追加しました。これにより、複数のリソースが既存のサービスアカウントを共有でき、すべてのクラスターとプーラーにわたってワンタイム IAM 構成 AWS IRSA、GCP Workload Identity、または Azure Workload Identity などを容易にします。 @bozkayasalihx による投稿。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9287">#9287

2.4.3 機能強化

Description	Addresses
TLS 暗号スイートと最小/最大 TLS バージョンの詳細な構成のサポートにより、<code>Pooler</code> CRD を改善しました。 これにより、管理者は、プーラーからクライアント、およびプーラーからサーバーへの接続に関する厳密なセキュリティコンプライアンス要件を満たすことができます。 @alex1989hu による投稿。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9571">#9571
アップグレードジョブで<code>BackoffLimit=0</code>を設定することにより、メジャーアップグレードの信頼性が向上し、失敗した<code>pg_upgrade</code>の不必要な再試行を防ぎます。 ユーザーがコンテナイメージを元に戻すときに、オペレーターは失敗したジョブを自動的に削除するようになりました。これにより、クラスターは元のバージョンで正常に再起動できます。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10104">#10104 , <a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10298">#10298
調整ループの主要なフェーズ中にネイティブ Kubernetes イベントを発行することにより、オペレーターの可観測性が向上しました。 これにより、<code>kubectl get events</code>を介して直接、オペレーターの意思決定プロセスと管理リソースのライフサイクルの可視性が提供されます。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10040">#10040

次のページに続く

表 15 – 前のページからの続き

Description	Addresses
バックアップリソースの<code>k8s.enterprisedb.io/reconciliationDisabled</code>アノテーションのサポートを拡張しました。 これにより、管理者は、特定のバックアップオブジェクトのオペレーターの調整ロジックを一時的にフリーズできます。 @GabriFedi97 による寄稿。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10020">#10020
<code>bin_path</code>フィールドを<code>postgresql.extensions</code>スタンザと、<code>ImageCatalog</code>および<code>ClusterImageCatalog</code>リソースに追加しました。 これにより、拡張機能は外部バイナリのディレクトリパスを指定できます。これらは Postgres プロセスの<code>PATH</code>環境変数に自動的に追加されます。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10250">#10250
<code>env</code>フィールドを<code>postgresql.extensions</code>スタンザと、<code>ImageCatalog</code>および<code>ClusterImageCatalog</code>リソースに追加しました。 これにより、クラスター管理者は Postgres プロセスのカスタム環境変数を定義できます。このフィールドは、拡張機能の絶対マウントパスに動的に解決するための<code>${image_root}</code>プレースホルダーをサポートしています。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10375">#10375
プラグインのファイナライザーを実装して、対応するサービスが削除されたときにプラグインによって管理されるリソースが正常にクリーンアップされるようにします。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9560">#9560
各調整サイクルの前にインスタンスがプライマリであることを確認することにより、ロール管理を向上させます。読み取り専用レプリカでの不必要な調整試行と偽のエラーメッセージを回避します。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9971">#9971
オペレーターは、クラスターに新しい PVC を追加するときに `primaryUpdateMethod`を優先するようになりました、ストレージの拡張または追加中にロールアウト戦略たとえば、スイッチオーバー対リスタートが尊重されるようにします。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9720">#9720
`alpha.k8s.enterprisedb.io/unrecoverable`アノテーションロジックを洗練して、`Ready`状態にまだ到達していないポッドでも機能できるようにし、スタックしたインスタンスのリカバリーを容易にします。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9968">#9968
回復不能な問題が発生したバックアップ用の「ターミナルエラー」フェーズを導入しました無効な資格情報または存在しないクラウドバケットなど。 これにより、オペレーターは運命の操作の再試行を停止し、リソースの枯渇を防ぎ、ステータスですぐに明確なフィードバックを提供します。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9353">#9353
<code>reconciliationStartedAt</code>および<code>reconciliationTerminatedAt</code>フィールドを<code>Backup</code>ステータスに導入することにより、長時間実行バックアップのモニタリングが向上しました。 この変更により、オペレーターの内部ライフサイクルが実際のバックアップツールの実行タイミング<code>startedAt</code>/<code>stoppedAt</code>から分離され、ユーザーはオペレーターが要求の処理を開始したときを追跡できます。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9351">#9351

次のページに続く

表 15 – 前のページからの続き

Description	Addresses
`Pending`フェーズを`Backup`ステータスに追加して、バックアップがキューに入れられ、使用可能なワーカーまたはインスタンスの可用性を待機しているときを明示的に示します。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9364">#9364

2.4.4 セキュリティ修正

Description	Addresses
セキュリティのベストプラクティスの統合** OpenSSF ベースラインスキャナーを統合し、`SECURITY-INSIGHTS.yaml`ファイルをリポジトリに追加して、業界標準のセキュリティレポートと調整します。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10054">#10054 , <a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10062">#10062
SLSA の来歴と SBOM** リリースバイナリとコンテナイメージに SLSA サプライチェーンレベルソフトウェアアーティファクトの来歴を追加しました。さらに、GoReleaser パイプライン内のソフトウェア部品表 SBOM 生成を有効にして、依存関係の透明性を向上させました。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10048">#10048 , <a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10074">#10074
パスワード漏洩防止** PostgreSQL が特定の調整フェーズ中にログにロールパスワードを漏洩する可能性がある潜在的なセキュリティリスクを修正しました。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9950">#9950

2.4.5 の変更

Description	Addresses
デフォルトの PostgreSQL バージョンを 18.3 イメージ`18.3-standard-ubi9`に更新しました。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10090">#10090

2.4.6 バグ修正

Description	Addresses
同期レプリケーションを使用するクラスターに影響するオペレーターのアップグレード中のデッドロックを修正しました ... 新旧のオペレーターバージョンを実行しているポッドが異なる PostgreSQL 構成ハッシュを計算し、均一性チェックが無期限にブロックされ、ローリング更新とインプレースアップグレードの両方の進行を防ぎます。 /p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10342">#10342
WAL ディスクが満杯のときにフェンシングアノテーションを処理できない問題を修正しました ... ディスク領域チェックによりインスタンスマネージャーの起動がブロックされたため。チェックは、ライフサイクルループの後半で、フェンシングが評価された後、実行されるようになりました。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10302">#10302
初期ブートストラップに使用される<code> VolumeSnapshot</code>が削除されている場合、レプリカが<code>Pending</code>状態でスタックする問題を修正しました。 オペレーターは、使用前にスナップショットの存在を検証するようになりました。スナップショットが見つからない場合は、次の利用可能な候補を使用しようとするか、<code>pg_basebackup</code>にフォールバックします。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10192">#10192
「監視対象プライマリ」ロールアウト戦略が、以前にスケジュールされた更新の遅延の原因となっていた、使用可能なすべてのロールアウトスロットを消費しないようにしました。 @ermakov-oleg による寄稿。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9977">#9977
特定のホットスタンバイパラメーターの変更がレプリカクラスターに正しく適用されなかった問題を修正しました。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9952">#9952
「Continue」結果が返されたときに後続のプラグインをスキップする可能性がある CNPG-I リコンサイラーフックのバグを修正しました。 @sharifmshaker による寄稿。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9978">#9978
現在ポッドに接続されていない PVC のファイルシステムのサイズを変更しようとしたときに発生したデッドロックシナリオを修正しました。 @jmealo による寄稿。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9981">#9981
<code>ConnectionParameters</code>で構成された外部クラスターを受け入れるようにブートストラップリカバリソースの Webhook 検証を修正しました <code>pg_basebackup</code>ベースのリカバリーの場合。以前は、Barman オブジェクトストアまたは CNPG-i プラグインも設定されていない限り、これらは誤って拒否されました。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10268">#10268

次のページに続く

表 18 – 前のページからの続き

Description	Addresses
拡張機能とテーブルスペースのボリューム名に接頭辞が付けられ、標準のクラスターボリュームとの名前の衝突を回避するようになりました。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9973">#9973
正常でないクラスターを冬眠させると、オペレーターは`WaitingForHealthy`条件を報告するようになり、遅延されたハイバーネーション状態を可視化します	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10193">#10193
ターゲットポッドが存在しない場合でも、フェンシングが正しく動作するように修正しました。 フェンシングはクラスターレベルのアノテーションで動作し、ポッドの存在に依存しないでください。インスタンス名の検証は、<code>cnp fencing on</code>コマンドでのみ実行されるようになりました。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10035">#10035
パッチ更新戦略を使用するときにすべての仕様フィールドの変更を正しく処理するようにクラスターおよびプーラーサービスリコンサイラーを修正しました。リコンサイラーは RFC 7386 JSON マージパッチを使用するようになり、クラウドプロバイダー設定フィールド `loadBalancerClass`など 誤って削除されるのを防ぎます。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10190">#10190 , <a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10311">#10311
パラレル WAL リストアに影響を与える非推奨のツリー内 Barman Cloud バックアップ実装の競合状態を修正しました。ここで、プリフェッチされたファイルはダウンロード中に読み取られる可能性があり、「invalid Checkpoint Record」エラーで PostgreSQL のリカバリーが失敗します。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10285">#10285
プラグインベースの WAL リストアにも適用されるようにタイムライン履歴ファイルの検証を修正しました。 以前は、<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9650">#9650 で導入された保護はツリー内リストアのみをカバーしており、プラグインはチェックをバイパスし、将来のダウンロードを許可していましたタイムライン履歴ファイル、レプリカでタイムライン不一致エラーの原因となります。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9849">#9849
cnp プラグインは、ImagePullSecret を`pgbench`ジョブポッドテンプレートに正しく伝播するようになりました。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10174">#10174

2.5 EDB CloudNativePG Cluster 1.28.2 release notes

リリース日 2026 年 3 月 31 日

CloudNativePG™クラスターの EDB Postgres ® AI のこのリリースには、次のものが含まれます。

2.5.1 ハイライト

ネイティブツリー内 Barman Cloud サポートの非推奨通知を更新して、Kubernetes 1.30.0 の EDB Postgres で 1.29.0 ではなく 1.30.0 で削除されることを反映しています。ユーザーは引き続き Barman Cloud プラグインに移行することをお勧めします。 #10167

2.5.2 機能強化

Description	Addresses
TLS 暗号スイートと最小/最大 TLS バージョンの詳細な構成のサポートにより、<code>Pooler</code> CRD を改善しました。 これにより、管理者は、プーラーからクライアント、およびプーラーからサーバーへの接続に関する厳密なセキュリティコンプライアンス要件を満たすことができます。 @alex1989hu による投稿。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9571">#9571
アップグレードジョブで<code>BackoffLimit=0</code>を設定することにより、メジャーアップグレードの信頼性が向上し、失敗した<code>pg_upgrade</code>の不必要な再試行を防ぎます。 ユーザーがコンテナイメージを元に戻すときに、オペレーターは失敗したジョブを自動的に削除するようになりました。これにより、クラスターは元のバージョンで正常に再起動できます。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10104">#10104 , <a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10298">#10298
各調整サイクルの前にインスタンスがプライマリであることを確認することにより、ロール管理を向上させます。読み取り専用レプリカでの不必要な調整試行と偽のエラーメッセージを回避します。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9971">#9971
オペレーターは、クラスターに新しい PVC を追加するときに `primaryUpdateMethod`を優先するようになりました、ストレージの拡張または追加中にロールアウト戦略たとえば、スイッチオーバー対リスタートが尊重されるようにします。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9720">#9720
`alpha.k8s.enterprisedb.io/unrecoverable`アノテーションロジックを洗練して、`Ready`状態にまだ到達していないポッドでも機能できるようにし、スタックしたインスタンスのリカバリーを容易にします。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9968">#9968

2.5.3 セキュリティ修正

Description	Addresses
セキュリティのベストプラクティスの統合** OpenSSF ベースラインスキャナーを統合し、`SECURITY-INSIGHTS.yaml`ファイルをリポジトリに追加して、業界標準のセキュリティレポートと調整します。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10054">#10054 , <a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10062">#10062
SLSA の来歴と SBOM** リリースバイナリとコンテナイメージに SLSA サプライチェーンレベルソフトウェアアーティファクトの来歴を追加しました。さらに、GoReleaser パイプライン内のソフトウェア部品表 SBOM 生成を有効にして、依存関係の透明性を向上させました。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10048">#10048 , <a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10074">#10074
パスワード漏洩防止** PostgreSQL が特定の調整フェーズ中にログにロールパスワードを漏洩する可能性がある潜在的なセキュリティリスクを修正しました。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9950">#9950

2.5.4 の変更

Description	Addresses
デフォルトの PostgreSQL バージョンを 18.3 イメージ`18.3-standard-ubi9`に更新しました。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10090">#10090

2.5.5 バグ修正

Description	Addresses
同期レプリケーションを使用するクラスターに影響するオペレーターのアップグレード中のデッドロックを修正しました ... 新旧のオペレーターバージョンを実行しているポッドが異なる PostgreSQL 構成ハッシュを計算し、均一性チェックが無期限にブロックされ、ローリング更新とインプレースアップグレードの両方の進行を防ぎます。 /p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10342">#10342

次のページに続く

表 22 – 前のページからの続き

Description	Addresses
WAL ディスクが満杯のときにフェンシングアノテーションを処理できない問題を修正しました ... ディスク領域チェックによりインスタンスマネージャーの起動がブロックされたため。チェックは、ライフサイクルループの後半で、フェンシングが評価された後、実行されるようになりました。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10302">#10302
初期ブートストラップに使用される<code> VolumeSnapshot</code>が削除されている場合、レプリカが<code>Pending</code>状態でスタックする問題を修正しました。 オペレーターは、使用前にスナップショットの存在を検証するようになりました。スナップショットが見つからない場合は、次の利用可能な候補を使用しようとするか、<code>pg_basebackup</code>にフォールバックします。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10192">#10192
「監視対象プライマリ」ロールアウト戦略が、以前にスケジュールされた更新の遅延の原因となっていた、使用可能なすべてのロールアウトスロットを消費しないようにしました。 @ermakov-oleg による寄稿。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9977">#9977
特定のホットスタンバイパラメーターの変更がレプリカクラスターに正しく適用されなかった問題を修正しました。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9952">#9952
「Continue」結果が返されたときに後続のプラグインをスキップする可能性がある CNPG-I リコンサイラーフックのバグを修正しました。 @sharifmshaker による寄稿。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9978">#9978
現在ポッドに接続されていない PVC のファイルシステムのサイズを変更しようとしたときに発生したデッドロックシナリオを修正しました。 @jmealo による寄稿。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9981">#9981
<code>ConnectionParameters</code>で構成された外部クラスターを受け入れるようにブートストラップリカバリソースの Webhook 検証を修正しました <code>pg_basebackup</code>ベースのリカバリーの場合。以前は、Barman オブジェクトストアまたは CNPG-i プラグインも設定されていない限り、これらは誤って拒否されました。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10268">#10268
拡張機能とテーブルスペースのボリューム名に接頭辞が付けられ、標準のクラスターボリュームとの名前の衝突を回避するようになりました。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9973">#9973
正常でないクラスターを冬眠させると、オペレーターは`WaitingForHealthy`条件を報告するようになり、遅延されたハイバーネーション状態を可視化します	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10193">#10193

次のページに続く

表 22 – 前のページからの続き

Description	Addresses
ターゲットポッドが存在しない場合でも、フェンシングが正しく動作するように修正しました。 フェンシングはクラスターレベルのアノテーションで動作し、ポッドの存在に依存しないでください。インスタンス名の検証は、<code>cnp fencing on</code>コマンドでのみ実行されるようになりました。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10035">#10035
パッチ更新戦略を使用するときにすべての仕様フィールドの変更を正しく処理するようにクラスターおよびプーラーサービスリコンサイラーを修正しました。リコンサイラーは RFC 7386 JSON マージパッチを使用するようになり、クラウドプロバイダー設定フィールド `loadBalancerClass`など 誤って削除されるのを防ぎます。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10190">#10190 , <a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10311">#10311
パラレル WAL リストアに影響を与える非推奨のツリー内 Barman Cloud バックアップ実装の競合状態を修正しました。ここで、プリフェッチされたファイルはダウンロード中に読み取られる可能性があり、「invalid Checkpoint Record」エラーで PostgreSQL のリカバリーが失敗します。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10285">#10285
プラグインベースの WAL リストアにも適用されるようにタイムライン履歴ファイルの検証を修正しました。 以前は、<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9650">#9650 で導入された保護はツリー内リストアのみをカバーしており、プラグインはチェックをバイパスし、将来のダウンロードを許可していましたタイムライン履歴ファイル、レプリカでタイムライン不一致エラーの原因となります。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9849">#9849
cnp プラグインは、ImagePullSecret を`pgbench`ジョブポッドテンプレートに正しく伝播するようになりました。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/10174">#10174

2.6 EDB CloudNativePG Cluster 1.28.1 release notes

リリース日 2025 年 2 月 10 日

CloudNativePG™クラスターの EDB Postgres ® AI のこのリリースには、次のものが含まれます。

2.6.1 機能強化

Description	Addresses
バックアップおよびリカバリ操作のための Azure の`DefaultAzureCredential`認証メカニズムのサポートを追加しました。 これは、バックアップ構成で`azureCredentials.useDefaultAzureCredentials: true`を設定することにより有効にでき、明示的なストレージアカウントキーまたは SAS トークンを必要とせずに Azure 環境での認証を簡素化します。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9468">#9468

2.6.2 バグ修正

Description	Addresses
アンダースコアを含む PostgreSQL 拡張機能名の検証を修正しました *pg_partman*、`pg_ivm`など。 アンダースコアを含む拡張機能名は、マウントパスに元の名前を保持しながら、Kubernetes ボリューム名にハイフンを使用するように自動的にサニタイズされます。 Webhook 検証は、サニタイズ後の名前の競合を防ぎます。 @shusaan さんからの投稿。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9386">#9386
メジャーバージョンのアップグレード後に、クラスターステータスの`TimelineID`が 1 にリセットされない重要な問題を修正しました。 `pg_upgrade`は新しいタイムラインを初期化するため、古い ID たとえばタイムライン 2 を保持すると、レプリカがオブジェクトストレージから互換性のない履歴ファイルを復元しようとし、致命的な「要求されたタイムラインはこのサーバーの履歴の子ではありません」エラーが発生しました。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9830">#9830
***Pooler*`の古い *TLS* ステータスフィールドが仕様から削除された後、クリアされない問題を修正しました。 これは、`*ServerTLS*`フィールドが再利用され、*PgBouncer* が誤った証明書を使用し、すべてのアプリケーションの接続をブロックする「*unsupportedcertificate*」エラーが発生した *v1.28.0* にアップグレードする場合に特に重要でした。オペレーターは、`*ServerCA*、*ClientCA*、*ClientTLS*、および`ServerTLS`ステータスフィールドが構成されなくなったときに明示的にクリアするようになりました。**	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9397">#9397
将来のタイムラインからタイムライン履歴ファイルをダウンロードしようとすることにより、レプリカがクラッシュループに入る可能性があるバグを修正しました。 これは、古いファイルが以前のクラスターライフから WAL アーカイブに残り、レプリカがリカバリ中に誤ってそれらを取得しようとした場合に発生しました。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9650">#9650
指定されたプライマリ移行中の`replica_cluster`セットアップの競合状態を修正し、レプリカクラスターでの一時的な「プライマリなし」状態を防止します。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9601">#9601
バックアップコントローラは、一意のインスタンスセッション ID を使用して、インスタンスマネージャーの再起動を検出するようになりました。 これにより、基になるコンテナがクラッシュして再起動した場合、オペレーターがバックアップがまだ進行中であると誤って仮定するのを防ぎます。これにより、以前は孤立したバックアップオブジェクトが発生しました。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9370">#9370

次のページに続く

表 24 – 前のページからの続き

Description	Addresses
明示的な資格情報ストレージキーや SAS トークンなどを使用する場合に`storageAccount`が必要とされなかった Azure オブジェクトストア構成の検証ギャップを修正しました。 オペレーターは、これらの場合にストレージアカウント名が提供され、`connectionString`が他の認証パラメーターと相互排他的であることを強制するようになりました。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9604">#9604
クラスターが削除対象としてマークされると、オペレーターがすぐにリソースのクリーンアップを開始するように、削除パスを最適化しました。 これにより、内部調整ループの待機中にクラスターが `Terminating`ステータスに留まる時間が大幅に削減されます。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9555">#9555
機能が無効になっている場合、またはクラスターが再構成されているときに、レプリケーションスロットがレプリカから適切にドロップされない問題を修正しました。 これにより、未使用のスロットがプライマリで WAL のビルドアップを発生させないようになります。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9381">#9381
`Pooler`用に作成された `ServiceAccount`に `imagePullSecrets`が追加されない問題を修正しました。 以前は、これらのシークレットは展開には適用されましたが、SA には適用されなかったため、特定のセキュリティポリシーを使用して制限された環境でイメージのプルに失敗しました。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9427">#9427
オペレーターが `PodMonitor`を削除する前に所有権を確認するチェックを追加しました。 これにより、オペレーターが、予想される CNP リソースと名前を共有する手動管理モニタリングリソースを誤って削除するのを防ぎます。 @juliamertz による寄稿。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9340">#9340
`pg_stat_archiver`メトリックが、スイッチオーバー後にスタンバイインスタンスで古いデータを報告し続けるバグを修正しました。 PostgreSQL はプライマリで有効なアーカイバー統計のみを提供するため、エクスポーターはスタンバイでこれらのメトリックをスキップするようになりました。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9411">#9411
リカバリー `targetTime`のタイムスタンプ形式の解釈を明確にしました。 明示的なタイムゾーンなしで提供されたタイムスタンプは、一貫して UTC として解釈されるようになりました。 @pchovelon による寄稿。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/8937">#8937
バックアップステータスの更新を修正して、コンカレント更新中の「リソースが変更されました」エラーを防止します。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9551">#9551
バックアップポッドが見つからない場合に、正しいポッド名を使用するようにイベントレポートを修正しました。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9552">#9552
非常に多くの履歴バックアップがあるクラスターのスケジュールされたバックアップ操作のパフォーマンスの向上。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9489">#9489

次のページに続く

表 24 – 前のページからの続き

Description	Addresses
`Database`オブジェクトのファイナライザーを削除する際のエラー処理を修正しました。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9431">#9431
クラスターに`skipWalArchiving`アノテーションが存在する場合に「Disabled」を表示するように`status`コマンドを更新しました。 これは、WAL アーカイブが意図的にバイパスされる場合の、混乱を招く「起動中」または「不明」状態を置き換えます。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9709">#9709
`logs --follow`コマンドを修正して、現在のログストリームがすべて完了したときに途中で終了するのではなく、新しいポッドのポーリングを継続します。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9599">#9599

2.7 EDB CloudNativePG Cluster 1.28.0 release notes

リリース日 2025 年 12 月 16 日

CloudNativePG™クラスターの EDB Postgres ® AI のこのリリースには、次のものが含まれます。

2.7.1 ハイライト

EDB private container registry を 1 つの場所に統合しました。この変更により、アクセスと展開が簡素化されます。v1.28.0 以降、オペレーターのアップグレードを試行するお客様は、新しい統合リポジトリのプルシークレットを構成する必要があります。詳細と手順については、 Migrating to the unified private EDB container registry に従ってください。

2.7.2 機能

Description	Addresses
クォーラムベースのフェールオーバーが安定版に昇格 1.27.0 で実験的に導入されたクォーラムベースのフェールオーバー機能を安定した API に昇格させました。このデータ駆動型フェイルオーバーメカニズムは、`spec.postgresql.synchronous.failoverQuorum`フィールドを介して構成され、以前の`alpha.k8s.enterprisedb.io/failoverQuorum`アノテーションから卒業しました。	#8589
宣言的な外部データ管理 *Database* CRD を拡張することにより、外部データラッパー FDW の包括的な宣言管理を導入しました。この機能は`.spec.fdws`および`.spec.servers`フィールドを追加し、FDW 拡張機能とそれに対応する外部サーバーを`Database`リソースから直接管理できます。この作業は、LFX メンターシッププログラム 2025 第 2 期の一部として、Ying Zhu@EdwinaZhu によって実装されました。	#7942 , #8401

2.7.3 機能強化

Description	Addresses
primaryUpdateMethod: restart を使用する場合のイメージと構成の同時変更が有効になり、同じ操作でコンテナイメージ PostgreSQL バージョンまたは拡張機能を含むと PostgreSQL 構成設定を更新できます。 `primaryUpdateMethod: switchover`を使用する場合でも、イメージと構成の変更を個別に実行して、スイッチオーバープロセス中の構成の不一致を回避する必要があることに注意してください。	#8241
デフォルトの `tcp_user_timeout`を 5 秒に設定することにより、レプリカインスタンスのネットワーク障害の検出が向上しました。 この変更により、レプリカはサイレントネットワークドロップをより迅速に検出および回復できます。以前は、レプリカはこのような障害を検出するまでに最大 127 秒待機することができました。新しいタイムアウトでは、5 秒以内にプライマリに再接続します。以前の動作を保持するには、オペレーター構成で `STANDBY_TCP_USER_TIMEOUT`を `0`に設定します。	#9317
Kubernetes の EDB Postgres によって生成されたすべてのリソースクラスター、バックアップ、プーラーなどに、標準の Kubernetes 推奨ラベル `app.kubernetes.io/name`などを採用しました。 @JefeDavis による寄稿。	#8087
ポッドレベルで `securityContext`を導入し、個々のコンテナごとに `container-SecurityContext`を導入しました *postgres*、*init*、サイドカーを含む これにより、セキュリティ設定の詳細な制御が提供され、以前のクラスター全体の `postgres`および `operator`ユーザー設定を置き換えます。 @x0ddf による投稿。	#6614
レプリカポッドの `alpha.k8s.enterprisedb.io/unrecoverable=true`アノテーションを導入しました。 このアノテーションは、適用されると、ポッドと PVC を削除することによりインスタンスを完全に削除するようにオペレーターに指示します。その後、プライマリからレプリカを再作成します。	#8178
ユーザー定義のモニタリングクエリーの新しいキャッシュレイヤーを導入して、PostgreSQL データベースの負荷を削減します。	#8003
`auth_dbname`を `pgbouncer`データベースに自動的に設定することにより、PgBouncer の統合が強化され、認証設定が簡素化されます。	#8671
データベースのインポート中にステージ固有の `pg_restore`オプションを提供できます。 ステージ固有のオプションは `preRestore`、*postRestore*、`dataRestore`です。	#7690
識別と管理が簡単になるように、PostgreSQL `majorVersion`を `Backup`オブジェクトのステータスに追加しました。	#8464
復元プロセスを開始する前に、すべての init コンテナが完了するのを待機するようにクラスターの復元を強化しました。 これにより、init コンテナで実行されているバックアップツールは、復元が開始される前にデータの準備を終了します。実装は、 `RestartPolicy=Always`を使用したサイドカーを無視することにより、Kubernetes init コンテナサイドカーを正しく処理します。	#9026

次のページに続く

表 26 – 前のページからの続き

Description	Addresses
`PGBOUNCER_IMAGE_NAME`オペレーター構成パラメーターを追加して、デフォルトの PgBouncer イメージをオーバーライドできるようにします。 これは、エアギャップ環境または内部レジストリを使用する場合に役立ちます。	#9232
CNP プラグイン クラスターサイズの計算などのファイルシステム操作のタイムアウトを構成するための`kubectl cnp status`コマンド に`--timeout`フラグを追加しました。デフォルトは 10 秒のままですが、操作に時間がかかる場合がある大規模なクラスターの場合、調整できます。	#9201
`cnp report`を改善して、よりシェルフレンドリーなファイル名を生成します。	#8984

2.7.4 セキュリティ修正

Description	Addresses
PgBouncer の詳細なカスタム TLS 構成を提供できます *Pooler* CRD は、*.spec.pgbouncer*`の下の `*clientTLSSecret*、*clientCASecret*、*serverTLSSecret*、および`serverCASecret`フィールドで拡張されました。これらのフィールドにより、ユーザーはクライアントからプーラー、およびプーラーからサーバーの両方の接続に独自の証明書を提供でき、オペレーターが生成した証明書よりも優先されます。	#8692
オペレーターのメトリックサーバーポート 8080 のオプションの TLS サポートを追加しました。この機能はオプトインであり、`METRICS_CERT_DIR`環境変数を設定することにより有効にされます。これは、指定されたディレクトリで`tls.crt`および`tls.key`ファイルを探すようにオペレーターに指示します。設定しない場合、サーバーは下位互換性のために HTTP を使用し続けます。	#8997
オペレーターの展開のみを必要とすることにより、`cnp_report_operator`が最小限の権限で動作できるようにしました 他のすべてのリソースポッド、シークレット、構成マップ、イベント、Webhook、および OLM データはオプショナルであり、ベストエフォートベースで収集されます。コマンドは、完全に失敗するのではなく、明確な警告をログに記録し、使用可能なデータでレポート生成を続行することにより、これらのリソースの権限エラーを適切に処理します。これにより、ユーザーが名前空間スコープの権限を制限できる最小特権アクセスが有効になります。	#8982

2.7.5 の変更

Description	Addresses
デフォルトの PostgreSQL バージョンを`18.1-standard-ubi9`に更新しました。	#9178

次のページに続く

表 28 – 前のページからの続き

Description	Addresses
新しいプーラー展開用に、デフォルトの PgBouncer バージョンを 1.25.1 に更新しました。	#9367

2.7.6 バグ修正

Description	Addresses
一時的な Kubernetes API サーバーの接続問題に対するすべてのプローブタイプライブネス、準備状況、およびスタートアップの復元力の向上。 プローブは、短時間のネットワークの中断中にキャッシュされたクラスター定義にフォールバックするキャッシュメカニズムを使用するようになり、不必要なポッドの再起動とプローブの失敗を防ぎます。	#9148
CNP-I バックアップ/WAL プラグイン `plugin-barman-cloud`などを使用してボリュームスナップショットから復元するときに正しく実行されるように`CheckEmptyWalArchive`セーフガードを修正しました。 以前、このチェックはプラグインベースの実装でスキップされていました。	#9306
ImageCatalog の取得が失敗した場合のエラー検出とレポートが改善されました。 オペレーターは Warning イベントを発行し、`NotFound`エラーだけでなくすべての障害タイプのエラーをログに記録するようになり、構成の問題の可視性が向上します。	#9266
`k8s.enterprisedb.io/pluginServerName`アノテーションを追加することにより、CNP-I プラグインに接続する際の TLS 証明書検証の問題を修正しました。 これにより、プラグインの証明書がサービス名とは別の DNS 名を使用する環境で、証明書の検証に使用される DNS 名をカスタマイズできます。	#9222
インスタンスマネージャーコントローラがエラーの後に再起動に失敗し、「コントローラは既に存在します」を報告する問題を修正しました。メッセージ。 コントローラは、後続の初期化試行に `SkipNameValidation`を使用するようになりました。 @mateusoliveira43 による寄稿。	#9123
宛先パスが絶対である場合、プラグインでの誤った WAL 復元パスの処理を修正し、パスの重複の問題を防止します。 @Endevir による寄稿。	#9093
PostgreSQL 13+との下位互換性があるように、マルチテーブルパブリケーションの `CREATE PUBLICATION` SQL 生成を修正しました。 以前に生成された構文は PostgreSQL 15+でのみ有効であり、古いバージョンでは構文エラーが発生しました。	#8888
インデックスではなく名前で `postgres`コンテナを確実に選択することにより、複雑なポッド定義でのバックアップの失敗を修正しました。	#8964
`cnp`プラグイン `cnp report`ログ収集、特に以前のログを取得する場合のバグを修正しました。 コレクターは、別の要求で以前のログと現在のログを正しく取得し、不足している以前のログたとえば、再起動履歴のないコンテナを適切に処理し、現在のログが常に収集されるようにします。	#8992

2.8 EDB CloudNativePG Cluster 1.27.3 release notes

リリース日 2025 年 2 月 10 日

EDB CloudNativePG クラスターのこのリリースには、次のものが含まれます。

2.8.1 機能強化

Description	Addresses
バックアップおよびリカバリ操作のための Azure の <code>DefaultAzureCredential</code>認証メカニズムのサポートを追加しました。 これは、バックアップ構成で <code>azureCredentials.useDefaultAzureCredentials:true</code>を設定することにより有効にでき、明示的なストレージアカウントキーまたは SAS トークンを必要とせずに Azure 環境での認証を簡素化します。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9468">#9468

2.8.2 バグ修正

Description	Addresses
アンダースコアを含む PostgreSQL 拡張機能名の検証を修正しました例、<code>pg_partman</code>、<code>pg_ivm</code> 。 アンダースコアを含む拡張機能名は、マウントパスに元の名前を保持しながら、Kubernetes ボリューム名にハイフンを使用するように自動的にサニタイズされます。 Webhook 検証は、サニタイズ後の名前の競合を防ぎます。 @shusaan による寄稿。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9386">#9386
メジャーバージョンのアップグレード後に、クラスターステータスの <code>TimelineID</code>が 1 にリセットされない重要な問題を修正しました。 <code>pg_upgrade</code>は新しいタイムラインを初期化するため、古い ID たとえばタイムライン 2 を保持すると、レプリカがオブジェクトストレージから互換性のない履歴ファイルを復元しようとし、致命的な「要求されたタイムラインはこのサーバーの子ではありません」が発生します。 history" エラー。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9830">#9830
将来のタイムラインからタイムライン履歴ファイルをダウンロードしようとすることにより、レプリカがクラッシュループに入る可能性があるバグを修正しました。 これは、古いファイルが以前のクラスターライフから WAL アーカイブに残り、レプリカがリカバリ中に誤ってそれらを取得しようとした場合に発生しました。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9650">#9650
指定されたプライマリ移行中の `replica_cluster`セットアップの競合状態を修正し、レプリカクラスターでの一時的な「プライマリなし」状態を防止します。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9601">#9601

次のページに続く

表 31 – 前のページからの続き

Description	Addresses
バックアップコントローラは、一意のインスタンスセッション ID を使用して、インスタンスマネージャーの再起動を検出するようになりました。 これにより、基になるコンテナがクラッシュして再起動した場合、オペレーターがバックアップがまだ進行中であると誤って仮定するのを防ぎます。これにより、以前は孤立したバックアップオブジェクトが発生しました。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9370">#9370
明示的な資格情報ストレージキーや SAS トークンなどを使用する場合、<code>storageAccount</code>が必要とされなかった Azure オブジェクトストア構成の検証ギャップを修正しました。 オペレーターは、これらの場合にストレージアカウント名を指定すること、および<code>connectionString</code>が他の認証パラメーターと相互排他的であることを強制するようになりました。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9604">#9604
クラスターが削除対象としてマークされると、オペレーターがすぐにリソースのクリーンアップを開始するように、削除パスを最適化しました。 これにより、内部調整ループを待機している間、クラスターが<code>Terminated</code>ステータスに留まる時間が大幅に削減されます。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9555">#9555
機能が無効になっている場合、またはクラスターが再構成されているときに、レプリケーションスロットがレプリカから適切にドロップされない問題を修正しました。 これにより、未使用のスロットがプライマリで WAL のビルドアップを発生させなくなります。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9381">#9381
<code>imagePullSecrets</code>が、 <code>Pooler</code>用に作成された<code>ServiceAccount</code>に追加されなかった問題を修正しました。 以前は、これらのシークレットは展開には適用されましたが、SA には適用されなかったため、特定のセキュリティポリシーを使用して制限された環境でイメージのプルに失敗しました。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9427">#9427
オペレーターが<code>PodMonitor</code>を削除する前に所有権を確認するチェックを追加しました。 これにより、オペレーターが、予想される CNP リソースと名前を共有する手動管理モニタリングリソースを誤って削除するのを防ぎます。 @juliamertz による寄稿。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9340">#9340
スイッチオーバー後に<code>pg_stat_archiver</code>メトリックがスタンバイインスタンスの古いデータを報告し続けるバグを修正しました。 PostgreSQL はプライマリで有効なアーカイバー統計のみを提供するため、エクスポーターはスタンバイでこれらのメトリックをスキップするようになりました。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9411">#9411
リカバリー<code>targetTime</code>のタイムスタンプ形式の解釈を明確にしました。 明示的なタイムゾーンなしで提供されたタイムスタンプは、一貫して UTC として解釈されるようになりました。 @pchovelon による寄稿。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/8937">#8937
バックアップステータスの更新を修正して、コンカレント更新中の「リソースが変更されました」エラーを防止します。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9551">#9551

次のページに続く

表 31 – 前のページからの続き

Description	Addresses
バックアップポッドが見つからない場合に、正しいポッド名を使用するようにイベントレポートを修正しました。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9552">#9552
非常に多くの履歴バックアップがあるクラスターのスケジュールされたバックアップ操作のパフォーマンスの向上。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9489">#9489
`Database`オブジェクトのファイナライザーを削除する際のエラー処理を修正しました。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9431">#9431
クラスターに<code>skipWalArchiving</code>アノテーションが存在する場合に「Disabled」を表示するように<code>status</code>コマンドを更新しました。これは、WAL アーカイブが意図的にバイパスされる場合の、紛らわしい「起動中」または「不明」状態を置き換えます。</p>	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9709">#9709
`logs --follow`コマンドを修正して、現在のログストリームがすべて完了したときに途中で終了するのではなく、新しいポッドのポーリングを継続します。	<a href="https://github.com/cloudnative-pg/cloudnative-pg/pull/9599">#9599

2.9 EDB CloudNativePG Cluster 1.27.2 release notes

リリース日 2025 年 12 月 16 日

EDB CloudNativePG クラスターのこのリリースには、次のものが含まれます。

2.9.1 ハイライト

EDB private container registry を 1 つの場所に統合しました。この変更により、アクセスと展開が簡素化されます。v1.27.2 以降、オペレーターのアップグレードを試行するお客様は、新しい統合リポジトリのプルシークレットを構成する必要があります。詳細と手順については、 Migrating to the unified private EDB container registry に従ってください。

2.9.2 の変更

Description	Addresses
新しいプーラー展開用に、デフォルトの PgBouncer バージョンを 1.25.1 に更新しました。	#9367
コミュニティ CloudNativePG 1.27.2 アップストリームリリースとマージされました。コミュニティを見る *EDB CloudNativePG Cluster 1.26.0 release notes*	

次のページに続く

表 32 – 前のページからの続き

Description	Addresses
デフォルトの PostgreSQL バージョンを `18.1-standard-ubi9`に更新しました。	

2.10 EDB CloudNativePG Cluster 1.27.1 release notes

リリース日 2025 年 10 月 24 日

このリリースには次のものが含まれます。

2.10.1 機能強化

Description	Addresses
以前にハードコードされた PgBouncer *auth_type*、*server_tls_sslmode*、および `client_tls_sslmode`設定 をオーバーライドするサポートを追加しました。デフォルト値は以前の動作と一貫性のあるままですが、必要に応じてカスタマイズできるようになりました。	#8674
PostgreSQL のスマートで高速なシャットダウン の前に `CHECKPOINT`ステップを追加して、特に `checkpoint_timeout`が高いシステムで、シャットダウン時間とレプリカプロモーション時間を削減します。	#8867
公式 `postgres-containers`プロジェクトに基づいて、非推奨またはサポートされていない OS バージョン のインスタンスマネージャーに警告を追加しました。	#8601
証明書解析エラーレポートの向上 障害は、汎用メッセージの代わりに特定のエラーをログに記録するようになり、トラブルシューティングを支援します。これは、Go 1.25.2 および 1.24.8 の CVE-2025-58187 修正後に特に関連しており、無効な DNS SAN の解析失敗をトリガーする可能性があります。	#8801
他のブートストラップ方法と一貫した `pg_basebackup`メソッド を使用してクラスターをブートストラップするときに、宛先 WAL アーカイブパスが空であることを確認するチェックを追加しました。	#8895
休止状態のクラスターでバックアップが実行されないように検証を追加しました。このようなクラスターで試行されたバックアップは、標準の前提条件チェックパターンに従って、`ClusterIsHibernated`の理由で失敗するようになりました。	#8870
`pprof`プロファイリングのサポートを追加しました インスタンスは、高度なデバッグのために `Cluster`リソースに `alpha.k8s.enterprisedb.io/enableInstancePprof`アノテーションを追加することにより、`pprof`ツールを有効にできるようになりました。	#7876
Ryan Wallner が提供するように、`cnp`プラグイン のフレキシブル I/O テスター FIO イメージを `wallnerryan/fiotools-aio:v2`に更新しました。	#8847

次のページに続く

表 33 – 前のページからの続き

Description	Addresses
`barman-cloud`ベースのバックアッププラグインを使用する場合、より詳細なステータス情報を提供するように`cnp status backup`コマンドを強化しました	#8780 , #8690

2.10.2 の変更

Description	Addresses
Barman Cloud のネイティブインコアサポートの廃止を少なくともバージョン 1.29 に遅延しました。	#8670
`postgres-containers`および `postgis-containers`イメージとイメージカタログアーティファクト の新しい形式を採用し、デフォルトの PostgreSQL バージョンを `18.0-system-trixie`に **更新** PostgreSQL 18 がサポートされるようになりました。	#8578 , #8760 , #8558
`Cluster`および `Pooler`リソースの `monitoring.enablePodMonitor`フィールド を非推奨にしました。このフィールドは、将来のリリースでは削除される予定です。 `PodMonitor`リソースに依存するユーザーは、代わりに手動で作成する必要があります。	#8753

2.10.3 バグ修正

Description	Addresses
パラレル WAL リカバリーでカスタム WAL セグメントサイズを使用する場合のバックアップ復元の失敗を修正しました。 オペレーターは、復元中に WAL の終わりファイルマーカーを管理しないため、バックアップが複数の WAL セグメントにまたがる場合のエラーを防ぎます。	#8873
以前のマイナーバージョンのボリュームスナップショットが誤って使用され、レプリカの作成を最適化する可能性があるメジャーアップグレードのバグを修正しました。	#8475
新しいクラスター をブートストラップする前にアプリケーションユーザーのシークレットを待機するように`initdb`を修正し、潜在的な競合状態を防ぎます。	#8663
同期レプリケーションを使用するインスタンスが 2 つのみのクラスターで正常に動作するように、クォーラムベースのフェイルオーバーを修正しました。	#8680
内部構成フィールド を無視するように構成ハッシュ計算を修正して、不必要な調整を防ぎます。	#8868
`cnpgi`プラグインの接続再試行ロジックを修正しました リコンシリエーションループは、接続プールの変更を正しく検出し、指数バックオフを使用して「クローズドプール」を削減するようになりました。エラー。	#8554

次のページに続く

表 35 – 前のページからの続き

Description	Addresses
バックアッププラグインと連携するように、レプリカのスケーリング中のボリュームスナップショットの使用を修正しました。 以前は、この最適化はツリー内バックアップ実装でのみ利用できましたが、バックアッププラグインを使用するクラスターは、新しいレプリカを作成するときにボリュームスナップショットを活用できるようになりました。	#8506
ブートストラップコントローラの init コンテナの設定を正しく継承するように `Pooler` テンプレートを修正しました。	#8394
`Pooler` の正しい API グループ `postgresql.k8s.enterprisedb.io` およびバックアップ Webhook を使用するように Webhook エラーを修正し、一貫した API エラーレポートを保証します。	#8485
エラーを処理する際のハイバーネーションリコンサイラー の潜在的な nil ポインター逆参照を修正しました。 @PascalBourdier による寄稿。	#8756
呼び出し元が誤って共有データを変更する可能性がある環境キャッシュの問題を修正しました。 `LoadEnv` ファンクションは、キャッシュされた環境スライスのコピーを結果、ミューテーションがキャッシュに影響を与えるのを防ぐようになりました。	#8880

2.11 EDB CloudNativePG Cluster 1.27.0 release notes

リリース日 2025 年 8 月 19 日

2.11.1 重要な変更

Liveness Probe のデフォルト動作が更新されました。 *isolated primary is now forcibly shut down*

設定された `livenessProbeTimeout` デフォルト 30 秒以内。このリリースには次のものが含まれます。

機能

Description	Addresses
OCI 準拠のコンテナイメージとしてパッケージされた PostgreSQL 拡張機能の動的ロードがサポートされるようになりました。 `.spec.postgresql.extensions` スタンザを使用すると、PostgreSQL 拡張機能をインスタンスポッド内の読み取り専用の不変ボリュームとしてマウントできます。これにより、ベースイメージを再構築せずに動的な拡張機能管理が可能になります。	#7991
HA クラスターでの論理デコードスロット同期。 `spec.replicationSlots.highAvailability` の下に `synchronizeLogicalDecoding` フィールドを追加して、高可用性クラスター間の論理デコードスロットの自動同期を有効にします。これにより、論理レプリケーションサブスクライバーは、パブリッシャーのフェールオーバー後もシームレスに継続されます。	#7931

次のページに続く

表 36 – 前のページからの続き

Description	Addresses
プライマリ分離チェック。 1.26 で導入された liveness pinger 実験機能を安定版に昇格させ、 `.spec.probes.liveness.isolationCheck`セクションを追加して、デフォルトで liveness プローブのプライマリ分離チェックを有効にします。これにより、Kubernetes 環境での主要な接続問題の検出と処理が向上します。	#7845

機能強化

Description	Addresses
クォーラムベースのフェールオーバーを有効にするオプトイン実験機能を導入して、フェールオーバーイベント中の安全性とデータの耐久性を向上させました。この機能はフェイルオーバークォーラムとも呼ばれ、 `alpha.k8s.enterprisedb.io/failoverQuorum`アノテーションを介してアクティブ化できます。	#7572
`streaming_replica`などの事前定義されたユーザーのユーザーマップのサポートを追加し、厳密なポリシーまたは共有 CA を持つ環境で異なるコモンネームを持つ自己管理クライアント証明書を使用できますが、レプリカは`streaming_replica`ロールを使用してクラスターに参加できます。	#7725
`Cluster`ステータスに新しい `PhaseFailurePlugin`フェーズを追加して、プラグイン関連の障害のオブザーバビリティを向上させました。	#7988
`Backup.spec`フィールドを作成後に不変にして、バックアップ操作の一貫性と予測可能性を保証します。	#7904
ユーザーシークレットに `fqdn-uri`および `fqdn-jdbc-uri`フィールドを追加して、完全修飾ドメイン名ベースの接続文字列の取得を簡素化しました。	#7852
`Postgres`インターフェイスのサポートを CNP-I オペレーターに追加し、プラグインベースのアーキテクチャへの移行を継続します。	#7179
`metrics`機能を CNP-I インスタンスウェブサーバーに追加し、インスタンスから直接メトリックの公開を有効にして、可観測性を向上させます。	#8033

バグ修正

Description	Addresses
`switchover`ストラテジーを使用して`barman-cloud`プラグインに移行する場合のブロック解除されたロールアウト。プラグインサポートの欠如により WAL アーカイブが失敗した後、元のプライマリポッドが正常に再起動するようになりました。	#8236

2.12 EDB CloudNativePG Cluster 1.26.3 release notes

リリース日 2025 年 12 月 16 日

EDB CloudNativePG クラスターのこのリリースには、次のものが含まれます。

2.12.1 ハイライト

EDB private container registry を 1 つの場所に統合しました。この変更により、アクセスと展開が簡素化されます。v1.26.3 以降、オペレーターのアップグレードを試行するお客様は、新しい統合リポジトリのプルシークレットを構成する必要があります。詳細と手順については、 Migrating to the unified private EDB container registry に従ってください。

2.12.2 の変更

Description	Addresses
新しいプーラー展開用に、デフォルトの PgBouncer バージョンを 1.25.1 に更新しました。	#9367
コミュニティ CloudNativePG 1.26.3 アップストリームリリースとマージされました。コミュニティを見る *EDB CloudNativePG Cluster 1.26.0 release notes*	
デフォルトの PostgreSQL バージョンを`18.1-standard-ubi9`に更新しました。	

2.13 EDB CloudNativePG Cluster 1.26.2 release notes

リリース日 2025 年 10 月 24 日

EDB CloudNativePG クラスターのこのリリースには、次のものが含まれます。

2.13.1 の変更

Description	Addresses
コミュニティ CloudNativePG 1.26.2 アップストリームリリースとマージされました。コミュニティを見る *EDB CloudNativePG Cluster 1.26.0 release notes*	

2.14 EDB CloudNativePG Cluster 1.26.1 release notes

リリース日 2025 年 7 月 25 日

EDB CloudNativePG クラスターのこのリリースには、次のものが含まれます。

2.14.1 の変更

Description	Addresses
コミュニティ CloudNativePG 1.26.1 アップストリームリリースとマージされました。コミュニティ *EDB CloudNativePG Cluster 1.26.0 release notes* を参照してください。	

2.15 EDB CloudNativePG Cluster 1.26.0 release notes

リリース日 2025 年 5 月 23 日

EDB CloudNativePG クラスターのこのリリースには、次のものが含まれます。

2.15.1 の変更

Description	Addresses
コミュニティ CloudNativePG 1.26.0 アップストリームリリースとマージされました。コミュニティ *EDB CloudNativePG Cluster 1.26.0 release notes* を参照してください。	

第3章　GETTING_STARTED

3.1 Before you start

始める前に、Kubernetes と PostgreSQL に固有の用語を確認することが重要です。

3.1.1 Kubernetes の用語

Node

_node_は、Kubernetes の仮想または物理的なワーカーマシンであり、ポッドを実行するために必要なすべてのサービスがコントロールプレーンノードによって管理されます。

Postgres Node

_Postgres node_は、PostgreSQL ワークロードの実行専用の Kubernetes ワーカーノードです。これは、`node-role.kubernetes.io` ラベルとテイントを *proposed by EDB Postgres AI for CloudNativePG Cluster* として適用することにより実現されます。`postgres` ノードとも呼ばれます。

Pod

_pod_は、Kubernetes クラスターに展開できる最小のコンピューティングユニットであり、ネットワークとストレージを共有する 1 つ以上のコンテナで構成されます。

Service

_service_は、ポッドのグループで実行され、アプリケーション全体のサービス検出、ロードバランシング、フェイルオーバーなどの重要な機能を標準化するアプリケーションをネットワークサービスとして公開する抽象化です。

Secret

_secret_は、パスワード、アクセスキー、トークンなどの少量の機密データを保存し、ポッドで使用するように設計されたオブジェクトです。

Storage Class

_storage class_を使用すると、管理者は、プロビジョナー AWS EBS など、再利用ポリシー、マウントオプション、ボリューム拡張などを含むクラスター内のストレージのクラスを定義できます。

Persistent Volume

_永続ボリューム_PV は、管理者によって手動でプロビジョニングされたストレージ、または_storage class_コントローラによって動的にプロビジョニングされたストレージを表す Kubernetes クラスター内のリソースです。PV は、_persistent volume resume_を使用してポッドに関連付けられ、そのライフサイクルはそれを使用するポッドから独立しています。通常、PV はネットワークボリューム、特にパブリッククラウドでは。

local persistent volume(LPV) は、それを使用するポッドが実行されている特定のノードにのみ存在する永続ボリュームです。

Persistent Volume Claim

_persistent volume resume_PVC は、ストレージの要求を表します。これには、サイズ、アクセスモード、または特定のストレージクラスが含まれる場合があります。ポッドがノードリソースを消費する方法と同様に、PVC は PV のリソースを消費します。

Namespace

_namespace_は、Kubernetes クラスターの論理的で分離されたサブセットであり、より広範な物理クラスター内の_virtual cluster_として見ることができます。名前空間を使用すると、管理者はプロジェクト、部門、チームなどに基づいて分離された環境を作成できます。

RBAC

_role-ベースのセキュリティ_としても知られる_ロールベースのアクセス制御_RBAC は、コンピューターシステムのセキュリティで使用され、システムのネットワークとリソースへのアクセスを許可されたユーザーのみに制限する方法です。Kubernetes には、名前空間とクラスターレベルでロールを制御し、特定のリソースと個人に関連付けるネイティブ API があります。

CRD

_カスタムリソース定義_CRD は、Kubernetes API の拡張機能であり、開発者は_カスタムリソース_と呼ばれる新しいデータ型とオブジェクトを作成できます。

Operator

_operator_は、1 つ以上のアプリケーションまたは特定のサービスを管理するときに通常人間のオペレーターが実行する手順を自動化するカスタムリソースです。オペレーターは、Kubernetes を支援して、リソースの定義された状態が観察された状態と常に一致するようにします。

kubectl

`kubectl` は、Kubernetes クラスターの管理に使用されるコマンドラインツールです。

CloudNativePG クラスターの EDB Postgres AI には、コミュニティでサポートされている Kubernetes バージョンが必要です。詳細は Supported releases ページを参照してください。

3.1.2 PostgreSQL の用語

インスタンス

「IP アドレス」と「TCP ポート」のペア通常は 5432 で実行およびリッスンする Postgres サーバープロセス。

プライマリー

読み取り操作と書き込み操作の両方を受け入れることができる PostgreSQL インスタンス。

レプリカ

クラスター内の唯一のプライマリインスタンスから複製する PostgreSQL インスタンスは、先行書き込みログ WAL レコードのストリームを読み取ることにより更新され続けます。レプリカは、_standby_または

_secondary_server とも呼ばれます。PostgreSQL は、物理的ストリーミングレプリケーション非同期/同期とファイルベースのログ配布非同期に依存しています。

ホットスタンバイ

_replica_が読み取り専用のワークロードを受け入れられるようにする PostgreSQL 機能。

クラスター

高可用性 HA クラスターとして使用されるため、単一のプライマリとオプションの任意の数のレプリカで構成される PostgreSQL インスタンスのセット。

レプリカクラスター

選択した PostgreSQL クラスターから継続的リカバリモードにある CloudNativePG クラスター `Cluster` の EDB Postgres AI は、通常 Kubernetes クラスターの外部にあります。これは、プライベート、パブリック、ハイブリッド、およびマルチクラウドのコンテキストでのマルチクラスター展開を有効にする機能です。

指定されたプライマリ

別の PostgreSQL クラスターから継続的リカバリーにあり、レプリカクラスターがプライマリになった場合にプライマリになるように指定されているレプリカクラスター内の PostgreSQL スタンバイインスタンス。

スーパーユーザー

PostgreSQL では、 _superuser_は、`LOGIN` と `SUPERUSER` の両方の特権を持つロールです。セキュリティ上の理由から、CloudNativePG クラスターの EDB Postgres AI は、ローカルの Unix ドメインソケットを介した `peer` 認証を介して `postgres` ユーザーとして `postgres` データベースに接続することにより、管理タスクを実行します。

WAL

先行書き込みロギング WAL は、データベース管理システムのデータの整合性を保証する標準の方法です。

PVC グループ

EDB Postgres AI for CloudNativePG Cluster 用語の PVC グループは、同じ PostgreSQL インスタンス、つまり PGDATA を含むメインボリューム `storage` と WAL 用のボリューム `walStorage` に属する関連 PVC のグループです。

RTO

「目標復旧時間」の頭字語で、アプリケーションに悪影響を与えることなくシステムを利用できない時間。

</aRPO

「目標復旧ポイント」の頭字語、ディザスターリカバリーシナリオに従った許容可能なデータ損失のレベルの計算。

3.1.3 クラウド用語

リージョン

クラウドの_region_は、_アベイラビリティーゾーン_で編成された分離された独立した地理的エリアです。リージョン内のゾーンの往復ネットワーク遅延はほとんどありません。

ゾーン

クラウドの_アベイラビリティーゾーン__zone_としても知られるは、リソースを展開できるリージョンのエリアです。通常、アベイラビリティーゾーンは、データセンターまたは同じデータセンターの分離された建物に対応します。

3.1.4 次に何をするか

用語を理解したところで、選択したクラウド環境にオペレーターを展開する前に *test EDB Postgres AI for CloudNativePG Cluster on your laptop using a local cluster* を決定できます。

3.2 Use cases

CloudNativePG クラスターの EDB Postgres AI は、完全なクラウドネイティブエクスペリエンスを実現するために、同じ Kubernetes クラスター内にあるアプリケーションと連携するように設計されています。

ただし、データベースは Kubernetes クラスター内でホストできますが、アプリケーションは同時にコンテナ化できず、VM などの_従来の環境_で実行する必要がある場合があります。

3.2.1 ケース 1Kubernetes 内のアプリケーション

通常、アプリケーションとデータベースは、Kubernetes クラスター内の同じ名前空間で実行されます。

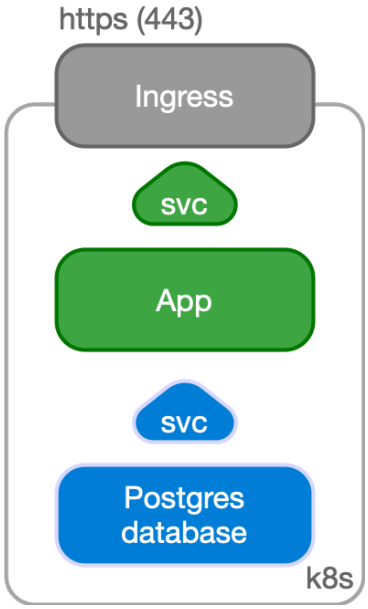


図 1: Application and Database inside Kubernetes

アプリケーションは通常ステートレスで、標準の `Deployment` として管理され、複数のレプリカが異なる Kubernetes ノードに分散され、`ClusterIP` サービスを介して内部公開されます。

サービスは、`Ingress` およびプロバイダーのロードバランサー機能を介して、HTTPS を介してエンドユーザーに外部公開されます。

アプリケーションは、バックエンドの PostgreSQL データベースを使用して、信頼性の高い永続的な方法で状態を追跡します。アプリケーションは、TLS 接続を介して現在のプライマリインスタンスをポイントす

る、CloudNativePG クラスターの EDB Postgres AI によって定義された `Cluster` リソースによって公開される読み取り/書き込みサービスを参照します。`Cluster` リソースは、単一のプライマリと複数のスタンバイアーキテクチャのロジックを埋め込み、Postgres での高可用性クラスターの管理の複雑さを隠蔽します。

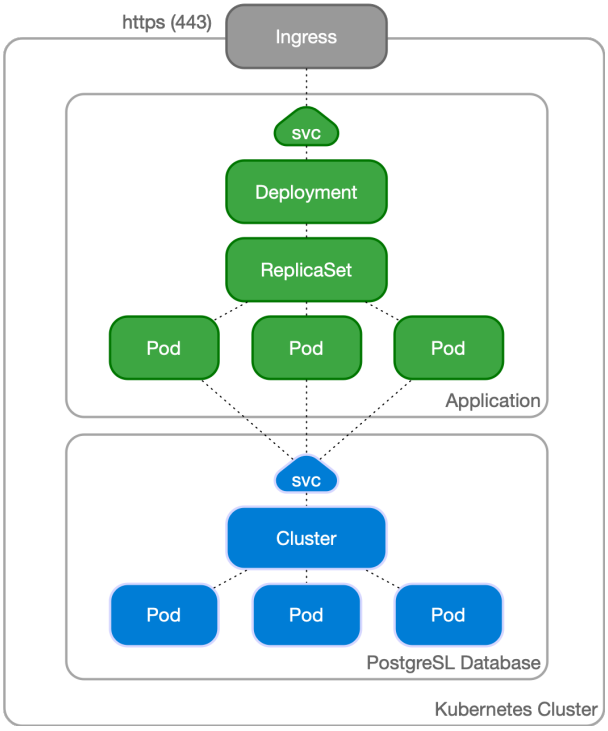


図 2: Close-up view of application and database inside Kubernetes

3.2.2 ケース 2Kubernetes 外部のアプリケーション

もう 1 つの考えられるユースケースは、Kubernetes 内で PostgreSQL データベースを管理し、アプリケーションをその外部たとえば仮想化環境に持たせることです。この場合、PostgreSQL は、Kubernetes で定義された Ingress リソース通常は *Service management* ページで説明している `LoadBalancer` サービスタイプに対応する IP アドレスまたはホスト名と TCP ポートで表されます。

アプリケーションは、PostgreSQL への TLS 接続の恩恵を引き続き受けられます。

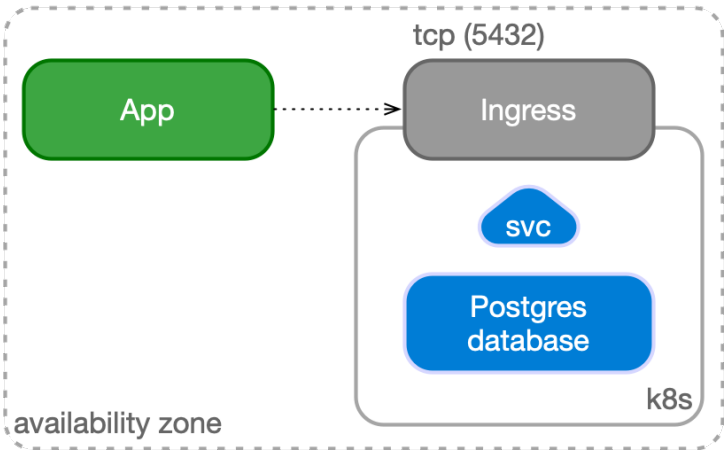


図 3: Application outside Kubernetes

3.3 Architecture

Recommended Architectures for PostgreSQL in Kubernetes というタイトルの CNCF ブログ投稿の記事を読むことをお勧めします。これは、Kubernetes での PostgreSQL 展開のベストプラクティスとデザインの考慮事項に関する貴重な洞察を提供します。

このドキュメントページでは、Kubernetes に PostgreSQL を展開する際に、堅牢なビジネス継続戦略を実装するための主要なアーキテクチャの考慮事項の概要を提供します。これらの考慮事項には次のものが含まれます。

- ** **マルチアベイラビリティーゾーン *Kubernetes* クラスター**

vs. **単一のアベイラビリティーゾーン *Kubernetes* クラスター** ** ストレッチクラスター 3 つ以上のアベイラビリティーゾーン全体での展開と非ストレッチクラスター単一のアベイラビリティーゾーン内での展開の違いを評価します。

- ***PostgreSQL* ワークロードのノードの予約** 特定のワーカーノードを `postgres` タスク専用にすることにより PostgreSQL ワークロードを分離し、最適なパフォーマンスを保証し、他のワークロードからの干渉を最小限に抑えます。
- ***PostgreSQL* アーキテクチャ**単一の Kubernetes クラスター内での効果的な PostgreSQL 展開を設計して、高可用性とパフォーマンス要件を満たす。
- ***Kubernetes* クラスター全体のデプロイメント**複数の Kubernetes クラスターにまたがる PostgreSQL アーキテクチャを計画および実装して、包括的なディザスターリカバリー機能を提供します。

3.3.1 状態の同期

PostgreSQL はデータベース管理システムであるため、Kubernetes で**ステートフルワークロード**として扱う必要があります。ステートレスアプリケーションは、主にトラフィックリダイレクションを使用して高可用性 HA とディザスターリカバリー DR を実現しますが、データベースの場合、次のいずれかを採用して、できれば連続的かつ瞬間的な方法で、状態を複数の場所にレプリケートする必要があります 2 つの戦略

- **ストレージレベルのレプリケーション**、通常は永続ボリューム
- **アプリケーションレベルのレプリケーション**、この特定の場合 PostgreSQL

CloudNativePG クラスターの EDB Postgres AI は、簡単な理由でアプリケーションレベルのレプリケーションに依存しています。PostgreSQL データベース管理システムには、**書き込み先行ログ WAL 配送**に基づいた堅牢で信頼性の高いビルトイン物理レプリケーション 機能が付属しています。**、これは 10 年以上、世界中の何百万ものユーザーがプロダクションで使用してきました。

PostgreSQL は、ネットワークを介した非同期および同期ストリーミングレプリケーションの両方と、非同期ファイルベースのログ配布通常はフォールバックオプションとして使用される、オブジェクトストアに WAL ファイルを保存するなどをサポートしています。レプリカは通常_スタンバイサーバー_と呼ばれ、_ホットスタンバイ_機能のおかげで、読み取り専用のワークロードに使用することもできます。

このトピックが詳細にカバーされている場所。

3.3.2 Kubernetes アーキテクチャ

Kubernetes は、冗長、低遅延、プライベートネットワーク接続を介して相互に接続されている別の物理的な場所データセンター、障害ゾーン、またはより頻繁に**アベイラビリティーゾーン**としても知られる場所にまたがる可能性をネイティブに提供します。

分散システムであるため、単一ゾーンの障害に対するコントロールプレーンの復元力を高めるために、Kubernetes クラスターの推奨最小アベイラビリティーゾーンの数は 3 です。詳細については、Running in multiple zones を参照してください。これは、**各データセンターが常にアクティブ**であり、ワークロードを同時に実行できることを意味します。

> **注釈**
>
> パブリッククラウドプロバイダーのマネージド Kubernetes サービスのほとんどは、各リージョンに 3 つ以上のアベイラビリティーゾーンを既に提供しています。

マルチアベイラビリティーゾーン Kubernetes クラスター

3 つ以上のゾーンを備えたマルチアベイラビリティーゾーンの Kubernetes アーキテクチャは、PostgreSQL の使用にお勧めします。このシナリオは、クラウドプロバイダーによって管理される Kubernetes サービスの一般的なシナリオです。

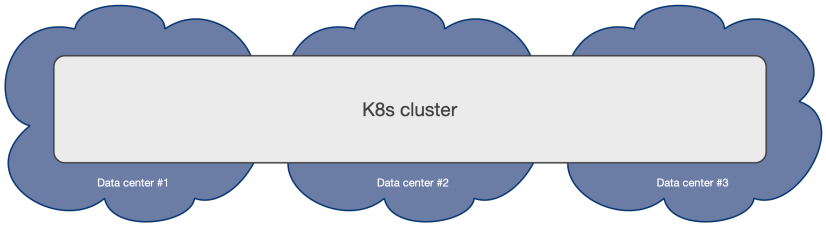


図 4: Kubernetes cluster spanning over 3 independent data centers

このようなアーキテクチャにより、CloudNativePG クラスターオペレーターの EDB Postgres AI は、すべてのアベイラビリティーゾーンをアクティブとして扱うことにより、単一の Kubernetes クラスター内のゾーンにわたって `Cluster` リソースの完全なライフサイクルを制御できます。これには、他の操作の中でも、ワークロードの *Job scheduling with EDB Postgres AI for CloudNativePG* が含まれます宣言的方法アフィニティルール、許容範囲、ノードセレクターに基づいて、自動フェイルオーバー、自己修復、および更新。すべては、単一の Kubernetes クラスター内のゾーンを超えてシームレスに動作します。

PostgreSQL **アーキテクチャ** を参照してください。

ストレージ、ワーカーノード、およびアベイラビリティーゾーンレベルでのシェアードナッシング展開を介して、同じ Kubernetes クラスター内で PostgreSQL クラスターをデザインする方法の詳細については、以下のセクションを参照してください。

さらに、 *Kubernetes* **クラスター全体のデプロイメント** を活用できます。

さまざまなリージョンで「パッシブ」 *PostgreSQL replica clusters* をホスティングし、宣言的構成を介して管理する分散 PostgreSQL トポロジ。このセットアップは、ディザスターリカバリー DR、読み取り専用操作、またはクロスリージョンの可用性に最適です。

単一のアベイラビリティーゾーン Kubernetes クラスター

Kubernetes クラスターにアベイラビリティーゾーンが 1 つだけある場合でも、CloudNativePG クラスターの EDB Postgres AI は、PostgreSQL データベースの HA および DR の結果を向上させるための多くの機能を提供し、単一点障害 SPoF をゾーンのレベルに押し上げます可能な限り-つまり、CloudNativePG Cluster クラスターの EDB Postgres AI に障害が発生する前に、ゾーンを停止する必要があります。

このシナリオは、1 つのデータセンターのみが使用できる自己管理のオンプレミスの Kubernetes クラスターの一般的なシナリオです。

単一のアベイラビリティーゾーン Kubernetes クラスターは、低遅延接続の範囲内で **2 つのデータセンター**のみが使用可能な場合、通常は同じ大都市圏で、唯一の実行可能なオプションです。2 つのゾーンのみがあると、マルチアベイラビリティーゾーンの Kubernetes クラスターを作成できません。これには、少なくとも 3 つのゾーンが必要です。その結果、ユーザーはアクティブ/パッシブ構成で 2 つの別個の Kubernetes クラスターを作成し、2 番目のクラスターは主にディザスターリカバリーに使用される *replica cluster feature* を参照してください。

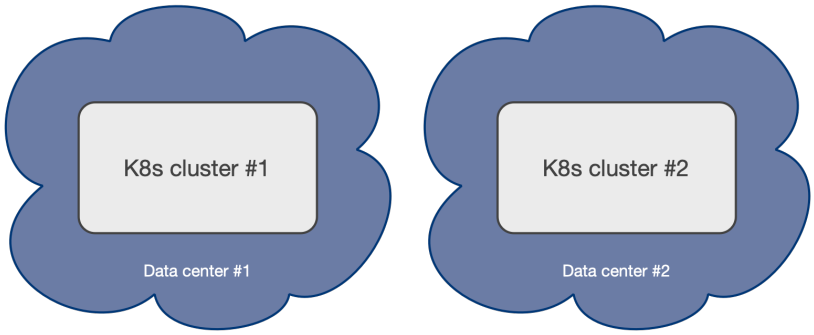


図 5: Example of a Kubernetes architecture with only 2 data centers

PostgreSQL アーキテクチャ を参照してください。

ストレージおよびワーカーノードレベルのみでのシェアードナッシング展開を介して、単一のアベイラビリティーゾーン Kubernetes クラスター内で PostgreSQL クラスターをデザインする方法の詳細については、以下のセクションを参照してください。HA の場合、このようなシナリオでは、PostgreSQL インスタンスが別のワーカーノードに配置され、同じストレージを共有しないことがさらに重要になります。

DR の場合、追加の *Kubernetes クラスター全体のデプロイメント* を使用して、“passive” *PostgreSQL replica clusters* をホスティングする分散トポロジを定義することにより、単一ゾーンより上に SPoF をプッシュできます。このシナリオの他の Kubernetes ワークロードと同様に、Kubernetes クラスターのプライマリとしての昇格を手動で行う必要があります。

replica cluster feature を介して、最初にプライマリクラスターを降格し、次にレプリカクラスターを昇格させることにより、分散 PostgreSQL トポロジを定義し、データセンター間の制御されたスイッチオーバーを調整できます。以前のプライマリを再クローンする必要はありません。フェールオーバーは完全に宣言型になりましたが、オペレーターは単一の Kubernetes クラスター内でのみ機能できるため、Kubernetes クラスター間の自動フェイルオーバーは CloudNativePG クラスターの EDB Postgres AI のスコープ内ではありません。

PostgreSQL ワークロードのノードの予約

マルチアベイラビリティーゾーン環境で動作しているか、より重要には、単一のアベイラビリティーゾーン内で動作しているかにかかわらず、運用環境で特定のワーカーノードを postgres 専用にすることにより、PostgreSQL ワークロードを分離することを強くお勧めします。PostgreSQL ワークロードの実行専用の Kubernetes ワーカーノードは、**Postgres ノード** または postgres ノードと呼ばれます。このアプローチにより、データベース操作の最適なパフォーマンスとリソース割り当てが保証されます。

Kubernetes では、これは宣言的な方法でノードラベルとテイントを使用して、インフラストラクチャ・アズ・コード IaC の実践に合わせて実現できます。ラベルはノードが postgres ワークロードを実行できることを保証し、テイントは非 postgres ワークロードが実行されないようにするのに役立ちます。そのノードでスケジュールされています。

提案されたノードラベル

CloudNativePG クラスターの EDB Postgres AI は、node-role.kubernetes.io/postgres ラベルを使用することをお勧めします。これは予約ラベル*.kubernetes.io であるため、ワーカーノードが作成された後にのみ適用できます。

postgres ラベルをノードに割り当てるには、次のコマンドを使用します。

```
kubectl label node <NODE-NAME> node-role.kubernetes.io/postgres=
```

Cluster リソースが postgres ノードでスケジュールされるようにするには、マニフェストで.spec.affinity.nodeSelector スタンザを正しく構成する必要があります。次に例を示します。

```
spec:
  # <snip>
  affinity:
    # <snip>
    nodeSelector:
      node-role.kubernetes.io/postgres: ""
```

提案されたノードテイント

CloudNativePG クラスターの EDB Postgres AI は、node-role.kubernetes.io/postgres テイントを使用することをお勧めします。

postgres テイントをノードに割り当てるには、次のコマンドを使用します。

```
kubectl taint node <NODE-NAME> node-role.kubernetes.io/postgres=:NoSchedule
```

postgres テイントを持つノードで Cluster リソースがスケジュールされるように保証には、マニフェストで.spec.affinity.tolerations スタンザを正しく構成する必要があります。次に例を示します。

```
spec:
  # <snip>
  affinity:
```

(次のページに続く)

(前のページからの続き)

```
  # <snip>
  tolerations:
    - key: node-role.kubernetes.io/postgres
      operator: Exists
      effect: NoSchedule
```

3.3.3 PostgreSQL アーキテクチャ

CloudNativePG クラスターの EDB Postgres AI は、非同期および同期ストリーミングレプリケーションに基づいてクラスターをサポートし、次の仕様で同じ Kubernetes クラスター内の複数のホットスタンバイレプリカを管理します。

- 1 つのプライマリ、HA 用のオプションの複数のホットスタンバイレプリカ
- アプリケーションで使用可能なサービス
- -rw アプリケーションはクラスターのプライマリインスタンスにのみ接続します b_tran_6 -ro アプリケーションは読み取り専用ワークロードオプショナルのホットスタンバイレプリカにのみ接続します
- -r アプリケーションは読み取り専用のインスタンスに接続しますワークロードのみオプショナル
- PostgreSQL クラスターの復元力を向上させるために推奨されるシェアードナッシングアーキテクチャ
- PostgreSQL インスタンスは別の Kubernetes ワーカーノードに存在し、ネットワークのみを共有する必要があります。その結果、インスタンスはストレージを共有せず、できれば自分が作成したノードに接続されたローカルボリュームを使用する必要があります。run on
- PostgreSQL インスタンスは、同じ Kubernetes クラスター/リージョン内の異なるアベイラビリティーゾーンに存在する必要があります

以下。

次の図は、3 つの異なるアベイラビリティーゾーンにまたがる PostgreSQL クラスターに推奨されるシェアードナッシングアーキテクチャの単純化したビューを提供し、それぞれが PostgreSQL データ専用のローカルストレージを備えた別のノードで実行されます。

CloudNativePG クラスターの EDB Postgres AI は、クラスターのトポロジが変更された場合に、上記のサービスの更新を自動的に処理します。たとえば、フェールオーバーが発生した場合、-rw サービスを自動的に更新して昇格したプライマリをポイントし、アプリケーションからのトラフィックがシームレスにリダイレクトされるようにします。

読み取り/書き込みワークロード

アプリケーションは、次の図に示すように、Kubernetes オペレーターによって_currentprimary_として選択された PostgreSQL インスタンスに接続することを決定できます。

アプリケーションは、 -rw サフィックスサービスを使用できます。

プライマリが一時的または永続的に使用不可になった場合、高可用性の目的で、CloudNativePG クラスターの EDB Postgres AI はフェイルオーバーをトリガーし、-rw サービスをクラスターの別のインスタンスをポイントします。

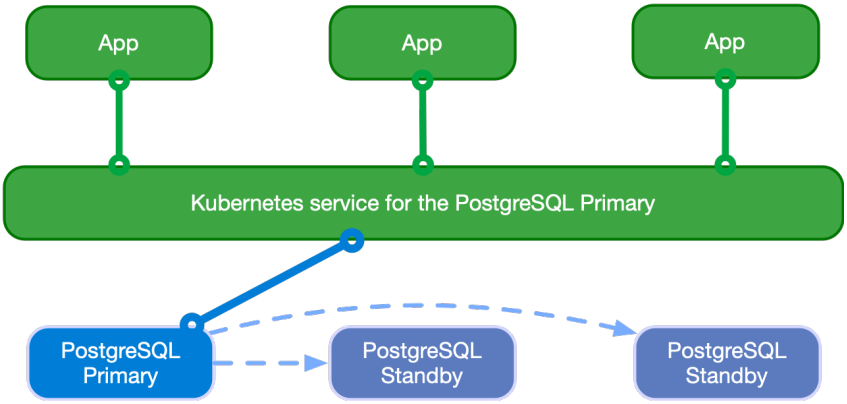


図 6: Applications writing to the single primary

読み取り専用ワークロード

は、これらのワークロードを処理するときに PostgreSQL が動作する方法を示し、精通しています。

アプリケーションは、オペレーターが利用できる `-ro` サービスを介してホットスタンバイレプリカにアクセスできます。このサービスにより、アプリケーションはプライマリノードから読み取り専用クエリをオフロードできます。

次の図は、アーキテクチャを示しています。

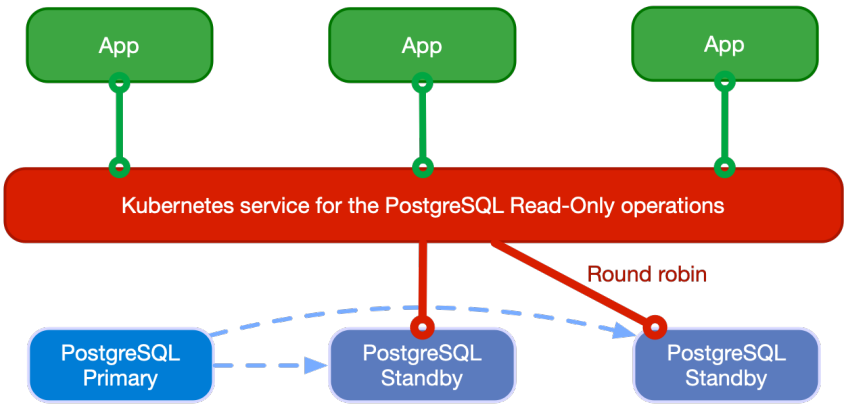


図 7: Applications reading from hot standby replicas in round robin

アプリケーションは、 `-r` サービスを介して PostgreSQL インスタンスにアクセスすることもできます。

3.3.4 Kubernetes クラスター全体のデプロイメント

> **注釈**
>
> CloudNativePG クラスターの EDB Postgres AI は、このセクションで説明するように、レプリカクラスターを使用して分散 PostgreSQL トポロジを定義できる機能を介して、複数の Kubernetes クラスター全体での PostgreSQL の展開をサポートしています。

分散 PostgreSQL クラスターでは、常にプライマリとして機能する 1 つの PostgreSQL インスタンスのみが存在できます。これは、アプリケーションが常に単一の Kubernetes クラスター内にのみ書き込むことがで

きることを意味します。

ただし、ビジネス継続性の目標に関しては、次のことが基本です。

- PostgreSQL バックアップデータを複数の場所、リージョンに保存し、場合によっては異なるプロバイダーを使用することにより、グローバル**リカバリポイント目標** リリース候補の目的 を削減しますディザスターリカバリー
- プライマリ Kubernetes クラスター高可用性を超えた PostgreSQL レプリケーションを利用することにより、グローバル**リカバリ時間目標** *RTO* を削減します。

上記の懸念事項に対処するために、CloudNativePG クラスターの EDB Postgres AI は、さまざまな Kubernetes クラスターに分散され、プライマリ PostgreSQL クラスターと 1 つ以上の PostgreSQL レプリカクラスターで構成される PostgreSQL トポロジの概念を導入します。この機能は、**レプリカクラスターを使用した分散 PostgreSQL トポロジ**と呼ばれ、プライベート、パブリック、ハイブリッド、およびマルチクラウドのコンテキストでのマルチクラスター展開が有効になります。

レプリカクラスターは、継続的リカバリーにある別個の `Cluster` リソースであり、WAL アーカイブからの WAL シッピングを介して、またはプライマリまたはスタンバイカスケードからのストリーミングレプリケーションを介して、別のソースからレプリケートされます。

次の図は、2 つの異なる Kubernetes クラスターにまたがる PostgreSQL クラスターを示しています。プライマリクラスターは最初の Kubernetes クラスターにあり、レプリカクラスターは 2 番目にあります。2 番目の Kubernetes クラスターは、会社のディザスターリカバリークラスターとして機能し、災害が発生して最初のクラスターが使用不可になった場合にアクティブ化される準備が整います。

レプリカクラスターは、プライマリクラスターと同じアーキテクチャを持つことができます。プライマリインスタンスの代わりに、レプリカクラスターには**指定されたプライマリ**インスタンスがあります。これは、ストリーミングレプリケーション対称アーキテクチャで任意の数のカスケードスタンバイサーバーを持つスタンバイサーバーです。

指定されたプライマリは、いつでも昇格でき、レプリカクラスターを、書き込み接続を受け入れることができるプライマリクラスターに変換できます。これは通常、次のことによりトリガーされます。

- **人間の決定**他の PostgreSQL クラスターまたは Kubernetes クラスター全体をプライマリにすることを選択します。データの損失を回避し、元のプライマリが再クローンを作成せずに続くことができるように保証特に大規模なデータセットの場合、最初に現在のプライマリを降格し、次に CloudNativePG クラスターの EDB Postgres AI が提供する API を使用して、指定されたプライマリを昇格させます。
- **予期しない障害** Kubernetes クラスター全体に障害が発生すると、データ損失が発生する可能性がありますが、PostgreSQL レプリカクラスターを昇格させることにより、他の Kubernetes クラスターにフェールオーバーする必要があります。

> **警告**
>
> CloudNativePG クラスターの EDB Postgres AI は、単一の Kubernetes クラスターを超える権限がないため、クラスター間の自動フェイルオーバーを実行できません。このような操作は手動で実行するか、マルチクラスター/フェデレーションクラスター対応機関に委任する必要があります。

上記の例では、指定されたプライマリは、ストリーミングレプリケーション `primary_conninfo` を介して WAL 更新を受信します。フォールバックとして、ファイルベースの WAL 出荷を使用してオブジェクトストア

から WAL セグメントを取得できます。たとえば、`restore_command` および `barman-cloud-wal-restore` を介して Barman Cloud プラグインを使用します。

CloudNativePG クラスターの EDB Postgres AI を使用すると、マルチプルのレプリカクラスターを使用したトポロジを定義できます。より少ない数のレプリカクラスターを定義し、クラスターがプライマリに昇格したときにこの数を増やすこともできます。

3.4 Installation and upgrades

セクション。

3.4.1 EDB サブスクリプショントークンの取得

次に、Repos 2.0 トークンを環境変数 `EDB_SUBSCRIPTION_TOKEN` として設定します。

```
EDB_SUBSCRIPTION_TOKEN=<your-token>
```

> **警告**
>
> トークンは機密情報です。未承認のユーザーに公開しないようにしてください。

3.4.2 Helm チャートを使用する

オペレーターは、提供されている Helm chart を使用してインストールできます。

3.4.3 オペレーターマニフェストを直接使用する

3.4.4 EDB プルシークレットのインストール

CloudNativePG クラスターに EDB Postgres AI をインストールする前に、`postgresql-operator-system` 名前空間で EDB ソフトウェアのプルシークレットを作成する必要があります。

プルシークレットは、オペレーターが存在する名前空間に保存する必要があります。次のコマンドを使用して `postgresql-operator-system` 名前空間を作成します。

```
kubectl create namespace postgresql-operator-system
```

プルシークレット自分自身を作成するには、次のコマンドを実行します。

```
kubectl create secret -n postgresql-operator-system docker-registry edb-pull-secret \
  --docker-server=docker.enterprisedb.com \
  --docker-username=k8s \
  --docker-password="$EDB_SUBSCRIPTION_TOKEN"
```

3.4.5 オペレーターのインストール

プルシークレットが名前空間に追加されたため、kubectl を介して適用される YAML マニフェストを介して、Kubernetes の他のリソースと同様に、オペレーターをインストールできます。

latest operator manifest をインストールできます

このマイナーリリースでは次のようになります。

```
kubectl apply --server-side -f \
  https://get.enterprisedb.io/pg4k/pg4k-1.30.0.yaml
```

これは次の方法で確認できます。

```
kubectl rollout status deployment \
  -n postgresql-operator-system postgresql-operator-controller-manager
```

3.4.6 kubectl の cnp プラグインを使用する

cnp プラグインを使用して、静的マニフェストにあるデフォルトの構成オプションをオーバーライドできます。

たとえば、デフォルトの最新のマニフェストを生成しますが、ウォッチ名前空間を特定の名前空間のみに変更するには、次のコマンドを実行できます。

```
kubectl cnp install generate \
  --watch-namespace "specific-namespace" \
  > cnp_for_specific_namespace.yaml
```

より包括的な例については、 *CNPG-I* ドキュメントを参照してください。

警告

GKE で CloudNativePG クラスターの EDB Postgres AI を展開しており、エラー`... failed to call webhook...`を取得した場合、公式で説明しているように、デフォルトでは、ワーカーノードとコントロールプレーン間のトラフィックは、いくつかの特定のポートを除きてファイアウォールによってブロックされていることに注意してください docs

そしてこの issue によって。 Webhook サービスの targetPort を許可されるもののいずれかに変更するか、ファイアウォールで Webhook のポート 9443 を開く必要があります。

Kubernetes では、オペレーターはデフォルトで Kubernetes Deployment として postgresql-operator-system 名前空間にインストールされます。この展開の名前は、インストール方法によって異なります。マニフェストまたは cnp プラグインを介してインストールされる場合、デフォルトでは、postgresql-operator-controller-manager と呼ばれます。Helm を介してインストールされる場合、デフォルトでは、展開名は helm リリース名にサフィックス-edb-postgres-for-kubernetes たとえば <name>-edb-postgres-for-kubernetes を追加したものから取得されます。

注釈

Helm を使用すると、 _"values.yaml"_ file の `fullnameOverride` フィールドを介して展開の名前をカスタマイズできます。

```
$ kubectl get deployments -n postgresql-operator-system
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
<deployment-name>   1/1     1            1           18m
```

```
kubectl describe deploy \
  -n postgresql-operator-system \
  <deployment-name>
```

他の展開と同様に、レプリカセットの上にあり、ローリングアップグレードをサポートします。CloudNativePG クラスターオペレーターの EDB Postgres AI のデフォルト構成には、単一のレプリカの展開が付属しており、ほとんどのインストールに適しています。ポッドが実行されているノードに到達できなくなった場合、ポッドは別のノードで再スケジュールされます。

オペレーターレベルでの高可用性が必要な場合、オペレーターがリーダー選択をサポートしていることを考慮すると、展開構成で複数のレプリカを指定できます。また、テイントと寛容を利用して、実際の PostgreSQL クラスターが実行されているノードでオペレーターが実行されないことを確認できます。これには、自己管理の Kubernetes インストールのコントロールプレーンが含まれる場合があります。

一部のバージョンは追加の手順を必要とする場合があるため、アップグレードを実行する前に。

1. コントローラと関連する Kubernetes リソースをアップグレードする
2. すべての PostgreSQL ポッドで実行されているインスタンスマネージャーをアップグレードする

リリースノートに別の記載がない限り、最初のステップは、通常、プレーンな Kubernetes インストールの新しいバージョンのマニフェストを適用するか、使用するディストリビューションのネイティブパッケージマネージャーを使用して実行されます上記のセクションの指示に従ってください。

2 番目のステップは、コントローラの更新後に自動的にトリガーされます。デフォルトでは、これにより、展開されたすべての PostgreSQL クラスターのローリング更新が開始され、新しいインスタンスマネージャーを使用するように一度に 1 つのインスタンスをアップグレードします。ローリング更新は、`primaryUpdateStrategy` オプションによって制御されるスイッチオーバーで終了します。デフォルト値 `unsupervised` は、スイッチオーバーを自動的に完了します。 `supervised` に設定している場合、ユーザーは `kubectl` の `cnp` プラグインを使用して、新しいプライマリインスタンスを手動でプロモートする必要があります。

3.4.7 スプレッドアップグレード

デフォルトでは、すべての PostgreSQL クラスターは同時にロールアウトされるため、特に複数のクラスターを管理する場合、リソース使用量の急増につながる可能性があります。CloudNativePG クラスターの EDB Postgres AI は、 *operator level* で 2 つの構成オプションを提供します

これにより、クラスターのロールアウト間、または同じクラスター内のインスタンス間で遅延を導入でき、時間の経過とともにリソース使用量を分散するのに役立ちます。

- `CLUSTERS_ROLLOUT_DELAY` さまざまな PostgreSQL クラスターデフォルト `0` のロールアウトの間で待機する秒数を定義します。
- `INSTANCES_ROLLOUT_DELAY` 同じ PostgreSQL クラスター内の個々のインスタンスのロールアウトの間で待機する秒数を定義しますデフォルト `0`。

3.4.8 インスタンスマネージャーのインプレース更新

デフォルトでは、CloudNativePG クラスターの EDB Postgres AI は、オペレーターが更新されるたびに、クラスターのローリング更新を発行します。オペレーターに同梱される新しいインスタンスマネージャーは、init コンテナを介して各 PostgreSQL ポッドに追加されます。

ただし、この動作は構成を介して変更でき、インスタンスマネージャーのインプレース更新を有効にできます。これは、コンテナを生きたままにする PID 1 プロセスです。

内部的には、CloudNativePG クラスターの EDB Postgres AI の各インスタンスマネージャーは、完全性検証フェーズが正常に完了し、すべての内部プロセスを正常に終了した後に、既存の実行可能ファイルを置き換える新しい実行可能ファイルのインジェクションをサポートしています。新しいバイナリで再起動すると、インスタンスマネージャーは既に実行されている_postmaster_をシームレスに採用します。

その結果、PostgreSQL プロセスは更新の影響を受けず、スイッチオーバーを実行する必要はありません。コインの反対側は、Pod がスタート後に変更され、不変性の純粋な概念を壊すということです。

Operator configuration で `ENABLE_INSTANCE_MANAGER_INPLACE_UPDATES` 環境変数を`'true'` に設定することにより、この機能を有効にできます。

インプレースアップグレードプロセスでは、ポッド内の init コンテナイメージは変更されません。したがって、ポッド定義はオペレーターの現在のバージョンを反映しません。

3.4.9 バージョン間の互換性

CloudNativePG クラスターの EDB Postgres AI は、セマンティックバージョニングに従います。同じ API バージョン内のオペレーターのすべてのリリースは、以前の API バージョンと互換性があります。現在の API バージョンは v1 で、オペレーターのバージョン 1.xy に対応します。

新機能に加えて、オペレーターの新しいバージョンにはバグ修正と安定性の強化が含まれています。このため、各バージョンは、最も安全で安定した Postgres 環境を維持するためにリリースされるため、**オペレーターの最新バージョンにアップグレードすることを強くお勧めします。

CloudNativePG クラスターの EDB Postgres AI は、現在少なくとも毎月オペレーターの新しいバージョンをリリースしています。各バージョンが利用可能になったときに更新を適用できない場合は、バージョンをスキップせずに、各バージョンを順番にアップグレードして、定期的に最新の状態になることをお勧めします。

EDB CloudNativePG Cluster 1.26.0 release notes ページには、CloudNativePG クラスターの EDB Postgres AI のすべてのリリースバージョンで導入された変更の詳細なリストが含まれており、ソフトウェアの新しいバージョンにアップグレードする前に読む必要があります。

ほとんどのバージョンは直接アップグレード可能であり、その場合、プレーンな Kubernetes インストールに新しいマニフェストを適用するか、選択したディストリビューションのネイティブパッケージマネージャーを使用するだけで十分です。

バージョンを直接アップグレードできない場合、新しいバージョンをインストールする前に古いバージョンを削除する必要があります。これはユーザーデータに影響を与えるのではなく、オペレーター自分自身にのみ影響を与えます。

3.4.10 1.30.0、1.29.2、または 1.28.4 へのアップグレード

3.4.11 オペレーター側のパスワードエンコーディング CVE-2026-55765

バージョン 1.30.0、1.29.2、および 1.28.4 から、セキュリティ上の理由から `CVE-2026-55765 / GHSA-w3gf-xc94-wvmj` 、CloudNativePG クラスター SCRAM-SHA-256 の EDB Postgres AI は、ロールパスワードをエンコードします **オペレーターサイド**クライアントサイド `CREATE /ALTER ROLE` ステートメントを発行する前に、PostgreSQL の観点から)。その結果、PostgreSQL パーサーに到達するリテラルおよび `pg_stat_statements` や `pgaudit` などの拡張機能が観察するリテラルは、`pg_authid.rolpassword` に到達するハッシュと同じであり、平文シークレットではありません。エンコードは、オペレーターが使用するすべての基本認証 `Secret` 、`postgres` スーパーユーザーシークレット、アプリケーションユーザーシークレット、および管理対象ロールパスワードシークレットに適用されます。MD5 または SCRAM-SHA-256 シャドウフォームで既に指定されたパスワードは、変更せずにパススルーされます。

PostgreSQL 14 以降、 `password_encryption` はデフォルトで `scram-sha-256` になるため、既存のインストールがこの変更の影響を受けることはないと思われます。

クラスターが `password_encryption` を `scram-sha-256` 以外の値に明示的にオーバーライドしていて、PostgreSQL 演算子ではなくにパスワードをハッシュする方法を決定したい場合は、各 basic-auth `Secret` にアノテーション `k8s.enterprisedb.io/passwordPassthrough: "enabled"` を設定してオプトアウトしますオペレーターが消費します。オペレーターはパスワード値をそのまま転送し、PostgreSQL は独自の `password_encryption` GUC に従ってそれをエンコードします。

> **警告**
>
> *k8s.enterprisedb.io/passwordPassthrough* アノテーションは、*Cluster* リソースではなく、**basic-auth Secret**自分自身に設定する必要があります。 *Cluster* に配置しても効果はなく、オペレーターは PostgreSQL に送信する前にパスワードに SCRAM-SHA-256 エンコードを適用し続けます。

詳細については。

3.4.12 search_path 強化 CVE-2026-55769

また、バージョン 1.30.0、1.29.2、および 1.28.4 以降、セキュリティ上の理由から `CVE-2026-55769 / GHSA-x8c2-3p4r-v9r6` 、CloudNativePG クラスターの EDB Postgres AI は、PostgreSQL への接続が開くすべての接続で `search_path` を固定 `pg_catalog, public, pg_temp` に固定します。テナント制御の `ALTER DATABASE /ALTER ROLE` 設定は、オペレーター発行のクエリが非修飾のオブジェクト名を解決する方法に影響を与えることができません。PgBouncer 統合で使用される `SECURITY DEFINER` ルックアップファンクションは、アップグレード後の最初のリコンシリエーション中に、独自の固定された `search_path` を使用して自動的に再作成されます。 スキーマの解決と `search_path の強化 <スキーマの解決と `search_path` の強化>` を参照

理論的根拠のため。

この変更は、 search_path が pg_catalog, public, pg_temp に固定されているトランザクション内で実行されるカスタムモニタリングクエリーにも影響します。カスタムメトリックが、非修飾名を介して他のユーザー定義スキーマにあるオブジェクトを参照している場合、アップグレード後も解決し続けるようにスキーマ修飾たとえば myschema.mytable 。ユーザー作成のブートストラップ postInit* および論理インポートインポート後 SQL は影響を受けません。これらは標準の"$user", public 解像度で実行され続けます。

3.4.13 DatabaseRole リソースを使用した宣言的ロール管理

バージョン 1.30.0 以降、クラスターの.spec.managed.roles スタンザでインライン宣言するのではなく、スタンドアロンの :ref:DatabaseRole` リソースを使用した宣言的ロール管理 <`DatabaseRole` リソースを使用した宣言的ロール管理>` リソースとして PostgreSQL ロールを管理することもできます。これはオプトイン機能の強化であり、アップグレード時のアクションは必要ありません。既存のインラインロールは変更せずに動作し続け、インライン\ ``managed.roles メソッドは完全にサポートされたままです。

中断せずに既存のインラインロールを DatabaseRole の下に移動するには、最初に DatabaseRole を作成してから、.spec.managed.roles から一致するエントリーを削除します。両方が存在しますが、クラスター仕様が常に優先されるため、インラインエントリがなくなった場合にのみ管理が引き渡されます。インライン管理ロールから `DatabaseRole への移行 <インライン管理ロールから `DatabaseRole` への移行>` を参照

完全な手順については。

DatabaseRole は ensure: absent をサポートしていません。インライン managed.roles スタンザが ensure: absent を設定することによりロールをドロップする場合、 DatabaseRole は代わりに databaseRoleReclaimPolicy フィールドに依存します。databaseRoleReclaimPolicy: delete でリソースを削除して PostgreSQL からロールを削除するか、デフォルトの retain をそのままにしてロールをそのままにします。

3.4.14 安全な予備選挙

バージョン 1.30.0 から、CloudNativePG クラスターの EDB Postgres AI は、クラスターごとの Kubernetes Lease を介してプライマリプロモーションを調整し、任意の時点で、最大 1 つのインスタンスが自分自身をプロモーションするように保証します。この動作は自動的に有効になり、構成は必要ありません。リースタイミングは、.spec.primaryLease を介してオプションで調整できます安全な予備選挙 を参照してください。

> **⚠ 警告**
>
> この機能を使用するには、オペレーターが `coordination.k8s.io` API グループの `Lease` オブジェクトを管理する必要があります。バンドルされたオペレーターマニフェストと Helm チャートは、必要な権限を自動的に付与します。オペレーターの RBAC を自分で管理する場合、アップグレードする前に、*coordination.k8s.io* API グループの `leases` に `create` 、*get* 、*list* 、*update* 、および `watch` 動詞を追加する必要があります。そうしないと、プライマリはプロモーションできません。

バージョン 1.30.0 から、オペレーターは、メモリ内クライアント証明書をピン留めすることにより、インスタンスマネージャーのステータスポート backup、pg_controldata 、部分的な WAL アーカイブ、およ

びインスタンスマネージャーのアップグレードの機密エンドポイントへの呼び出しを認証します。これは自動的に有効になり、構成は必要ありません。ステータス、状態、およびプローブエンドポイントは認証されないままです。この強化はバックポートされません。1.30.0 より前のリリースでは、`NetworkPolicy` でステータスポート TCP 8000 を制限し続けますが、これはセキュリティ境界のままです。**オペレーターからインスタンスへの認証** を参照

詳細については。

> **警告**
>
> この保護には、ハード要件があります。ステータスポート**は TLS で提供される必要があります**。これは、v1.24 以降のデフォルトです。 v1.24 より古いオペレーターによって作成されたインスタンスは、プレーンな HTTP でステータスポートを提供します。インスタンスマネージャーが 1.30.0 にアップグレードされると、オペレーターはそれらに対して認証できなくなり、保護されたエンドポイントへのすべての呼び出しは `401 Unauthorized` で**永久に**拒否されます。このようなインスタンスをまだ実行している場合は、ローリング更新を実行して、ポッドが TLS 対応ステータスポートで再作成されます。 v1.24 以降で作成されたインスタンスは影響を受けません。

これは、1.29.1、1.28.3、または 1.25.8 より古いリリース**たとえば、1.29.0、1.28.2、1.25.7、または以前のバージョンからアップグレードする場合にのみ適用されます。1.29.1、1.28.3、1.25.8 以降に既にインストールされている場合には、既にこの変更が適用されています。

`CVE-2026-44477` / `GHSA-423p-g724-fr39` の修正により、メトリックエクスポーターは、`postgres` スーパーユーザーではなく、`pg_monitor` 特権のみを持つ専用の `cnp_metrics_exporter` ロールとして認証します。

ユーザー所有テーブルを読み取るカスタムモニタリングクエリー、または `PUBLIC CONNECT` が取り消されたデータベースに対して `target_databases: '*'` を使用するカスタムモニタリングクエリーは、`cnp_metrics_exporter` に対する明示的な `GRANT` ステートメントが必要です。**カスタムクエリーの特権と安全性** を参照

と **メトリックエクスポーターロールの手動作成**

モニタリング文書で。

3.5 Quickstart

このセクションでは、Kind または Minikube のいずれかを使用してローカル Kubernetes クラスターに CloudNativePG クラスターの EDB Postgres AI を展開することにより、ローカルマシンで PostgreSQL クラスターをテストする方法をガイドします。

Red Hat OpenShift Container Platform ユーザーは、Red Hat OpenShift Local 旧 Red Hat CodeReady Containers で CloudNativePG クラスターの EDB Postgres AI の認定オペレーターをテストできます。

> **警告**
>
> このセクションに含まれる手順は、デモンストレーション、テスト、および練習のみを目的としているため、実稼働環境では使用しないでください。

他の Kubernetes アプリケーションと同様に、CloudNativePG クラスターの EDB Postgres AI は、YAML で書かれた通常のマニフェストを使用して展開されます。

これらの手順に従うことにより、ローカルの Kubernetes / Openshift インストールで PostgreSQL クラスターを起動し、実験できるはずです。

または *how to install `oc* <https://docs.openshift.com/container-platform/4.6/cli_reference/openshift_cli/getting-started-cli.html>`_ の Openshift ドキュメント

> **注釈**
>
> Openshift を実行している場合は、このドキュメントで `kubectl` が記載されているたびに `oc` を使用してください。 *kubectl* コマンドは、 *oc* コマンドと互換性があります。

3.5.1 パート 1-ローカルの Kubernetes / Openshift ローカルプレイグラウンドのセットアップ

最初の部分は、Minikube、Kind、または OpenShift Local のインストールに関するものです。時間をかけてシステムについて読み、どれを続行するかを決定してください。いずれかの設定を行ったら、パート 2 に進んでください。

パート *4 Prometheus* と *Grafana* を使用してクラスターを監視する では、ローカルテスト/評価のために Prometheus と Grafana を使用したモニタリングを設定する手順も提供します

Minikube

Minikube は、Kubernetes をローカルで簡単に実行できるツールです。Minikube は、Kubernetes を試してみるか、日常的に開発しようとしているユーザーのために、ラップトップの仮想マシン VM 内でシングルノード Kubernetes クラスターを実行します。通常、VirtualBox と組み合わせて使用します。

詳細については、ローカルの個人環境の公式 Kubernetes documentation on how toinstall Minikube で見つけることができます。インストールしたら、次のコマンドを実行して minikube クラスターを作成します。

```
minikube start
```

これにより、Kubernetes クラスターが作成され、使用する準備が整います。次のコマンドで動作することを確認します。

```
kubectl get nodes
```

`minikube` という名前のノードが 1 つ表示されます。

種類

仮想マシンハイパーバイザーを使用したくない場合、Kind は、Docker コンテナ「ノード」を使用してローカル Kubernetes クラスターを実行するためのツールです。Kind は、確かに「Kubernetes IN Docker」の略です。

Quickstart の指示に従って、環境に `kind` をインストールし、次の Kubernetes クラスターを作成します。

```
kind create cluster --name pg
```

OpenShift Local 旧 CodeReady ContainersCRC

1. Download OpenShift Local

そして、バイナリを PATH のディレクトリ内に移動します。

2. 次のコマンドを実行します。

```
crc setup
crc start
```

crc start 出力は、続行方法を説明します。

3. crc oc-env コマンドの出力を実行します。
4. 印刷した oc login コマンドを使用して、 kubeadmin としてログインします。 crc console を実行している Web コンソールを開くこともできます。ユーザーとパスワードは oc login コマンドの場合と同じです。
5. OpenShift Local には StorageClass が付属していないため、構成する必要があります。Dynamic volume provisioning wiki page をフォローする

rancher/local-path-provisioner をインストールします。

3.5.2 パート 2 CloudNativePG クラスターの EDB Postgres AI をインストールする

ラップトップで Kubernetes インストールをアップして実行しているため、CloudNativePG Cluster の EDB Postgres AI のインストールを続行できます。

クラスター構成ファイルで指定されない限り、 CloudNativePG クラスターの EDB Postgres AI は、現在デフォルトで Community Postgresql オペランドを展開します。詳細については、*EDB Postgres* サーバーの展開セクションを参照してください。

YAML マニフェストを介した *Kubernetes* のインストール セクションを参照して、PostgreSQL クラスターの展開を続行します。

3.5.3 パート 3 PostgreSQL クラスターを展開する

クラスター構成ファイルで指定されない限り、 CloudNativePG クラスターの EDB Postgres AI は、現在デフォルトで Community Postgresql オペランドを展開します。詳細については、*EDB Postgres* サーバーの展開 セクションを参照してください。

Kubernetes の他の展開と同様に、PostgreSQL クラスターを展開するには、目的の Cluster を定義する構成ファイルを適用する必要があります。

cluster-example.yaml サンプルファイルは、デフォルトのストレージクラスを使用して単純な Cluster を定義して、ディスク領域を割り当てます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3

  storage:
    size: 1Gi
```

3 ノード Community PostgreSQL クラスターを作成するには、次のコマンドを実行する必要があります。

```
kubectl apply -f cluster-example.yaml
```

`get pods` コマンドでポッドが作成されていることを確認できます。

```
kubectl get pods
```

これにより、デフォルトの名前空間でポッドが検索されます。Kubernetes インストールの他のワークロードからクラスターを分離するには、クラスターを展開する新しい名前空間をいつでも作成できます。または、ラベルを使用できます。オペレーターは、特定のクラスターに関連するすべてのオブジェクトに `k8s.enterprisedb.io/cluster` ラベルを適用します。例

```
kubectl get pods -l k8s.enterprisedb.io/cluster=<CLUSTER>
```

EDB Postgres サーバーの展開

デフォルトでは、オペレーターは、オペレーターがリリースされたときに Community PostgreSQL の最新メジャーバージョンの利用可能な最新のマイナーバージョンをインストールします。`Cluster` 定義の `spec` セクションで `imageName` キーを設定することにより、これをオーバーライドできます。たとえば、 EDB Postgres Advanced 18.1 をインストールするには、次を使用できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  # [...]
spec:
  # [...]
  imageName: docker.enterprisedb.com/k8s/edb-postgres-advanced:18.1-standard-ubi9
  #[...]
```

EDB Postgres Extended 18 をインストールするには、次を使用できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
```

(次のページに続く)

(前のページからの続き)

```
  # [...]
spec:
  # [...]
  imageName: docker.enterprisedb.com/k8s/edb-postgres-extended:18-standard-ubi9
  #[...]
```

3.5.4 パート 4 Prometheus と Grafana を使用してクラスターを監視する

このセクションでは、可観測性を実現するために Prometheus と Grafana を展開する方法、CloudNativePG Cluster クラスターの EDB Postgres AI、およびアラート条件を定義する Prometheus ルールセットをモニタするための Grafana ダッシュボードを作成する方法を示します。

Kube-Prometheus stack を活用します

Helm チャートは、Prometheus Community によって維持されます。追加のドキュメントと背景については、プロジェクトの Web サイトを参照してください。

Kube-Prometheus-stack Helm チャートは、 Prometheus Operator Alert Manager を含む Grafana 展開をインストールします。

この Helm チャートの展開用の構成ファイルは、CloudNativePG Cluster クラスターの EDB Postgres AI のオブザーバビリティに関する有用な初期設定を提供します。

インストール

Helm をまだインストールしていない場合は、instructions に従って、システムにインストールしてください。

`prometheus-community` Helm チャートリポジトリを追加し、サンプル構成

kube-stack-config.yaml で_Kube Prometheus stack_をインストールする必要があります。

これは、次のコマンドで実行できます。

```
helm repo add prometheus-community \
  https://prometheus-community.github.io/helm-charts

helm upgrade --install \
  -f https://raw.githubusercontent.com/EnterpriseDB/docs/main/product_docs/docs/
↪postgres_for_kubernetes/1/https://raw.githubusercontent.com/EnterpriseDB/docs/main/
↪product_docs/docs/postgres_for_kubernetes/1/samples/monitoring/kube-stack-config.
↪yaml \
  prometheus-community \
  prometheus-community/kube-prometheus-stack
```

完了すると、Prometheus、Grafana、および Alert Manager が `kube-stack-config.yaml` ファイルで構成されます。

- Prometheus インストールから、 **any** `PodMonitor` *_monitoring_* を参照してください。
- Alert Manager と Grafana は両方とも有効になっています。

ドキュメント。

いくつかのカスタムリソースが作成されたことがわかります。

```
% kubectl get crds
NAME                                        CREATED AT
…
alertmanagers.monitoring.coreos.com         <timestamp>
…
prometheuses.monitoring.coreos.com          <timestamp>
prometheusrules.monitoring.coreos.com       <timestamp>
…
```

および一連のサービス

```
% kubectl get svc
NAME                                      TYPE        PORT(S)
…                                          …           …
prometheus-community-grafana              ClusterIP   80/TCP
prometheus-community-kube-alertmanager    ClusterIP   9093/TCP
prometheus-community-kube-operator        ClusterIP   443/TCP
prometheus-community-kube-prometheus      ClusterIP   9090/TCP
```

Prometheus での表示

この時点で、モニタリングがアクティブ化された状態で展開された CloudNativePG クラスタークラスターの EDB Postgres AI は、Prometheus を介して観察可能です。

たとえば、`PodMonitor` を有効にした単純なクラスターを展開できます。

```
kubectl apply -f - <<EOF
- --
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-with-metrics
spec:
  instances: 3

  storage:
    size: 1Gi
- --
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: cluster-with-metrics
spec:
```

(次のページに続く)

(前のページからの続き)

```
  selector:
    matchLabels:
      k8s.enterprisedb.io/cluster: cluster-with-metrics
  podMetricsEndpoints:
  - port: metrics
EOF
```

Prometheus にアクセスするには、Prometheus サービスをポート転送します。

```
kubectl port-forward svc/prometheus-community-kube-prometheus 9090
```

次に、次の Prometheus コンソールにローカルにアクセスします。 http://localhost:9090/

CloudNativePG Cluster クラスターの EDB Postgres AI に関連する一連のメトリックを見つけます。詳細については、*monitoring section* を参照してください。

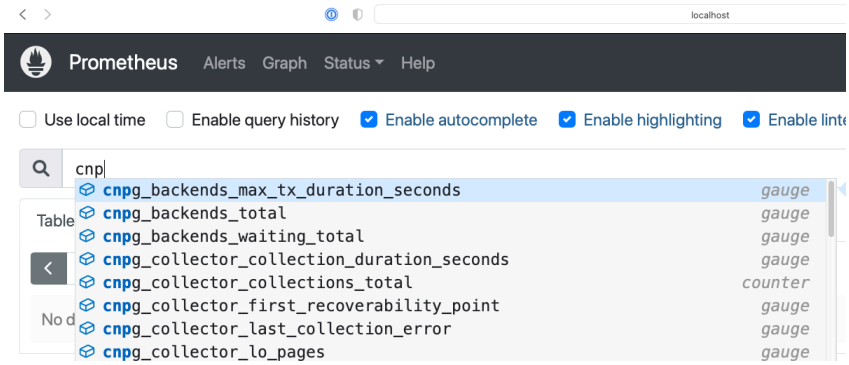


図 8: local prometheus

CloudNativePG クラスターオペレーターの EDB Postgres AI をターゲットとする PodMonitor を作成して、それをモニタすることもできます。*monitoring page* の関連セクションを参照してください。

`prometheusRule` を作成することにより、いくつかのアラートを定義できます。

```
kubectl apply -f \
  https://raw.githubusercontent.com/EnterpriseDB/docs/main/product_docs/docs/postgres_
↪for_kubernetes/1/https://raw.githubusercontent.com/EnterpriseDB/docs/main/product_
↪docs/docs/postgres_for_kubernetes/1/samples/monitoring/prometheusrule.yaml
```

デフォルトのアラートが表示されます。

```
% kubectl get prometheusrules
NAME                                                AGE
postgresql-operator-default-alerts                                  3m27s
```

Prometheus コンソールで、_Alerts_メニューをクリックして、インストールしたばかりのアラートを表示できます。

Grafana ダッシュボード

これまでのインストールでは、Grafana は事前定義されたダッシュボードなしで展開されます。

Grafana を開くには、grafana サービスをポート転送できます。

```
kubectl port-forward svc/prometheus-community-grafana 3000:80
```

http://localhost:3000/ で Grafana にローカルにアクセスします

ユーザー名として資格情報が `admin`、パスワードとして `prom-operator` を提供します `kube-stack-config.yaml` で定義。

CloudNativePG クラスターの EDB Postgres AI は、専用の grafana-dashboards で Grafana のデフォルトのダッシュボードを提供します。ファイル grafana-dashboard.json をダウンロードできます。

GUI から手動でインポートしますメニュー Dashboards > New > Import 。作成したばかりの `EDB Postgres AI for CloudNativePG Cluster` ダッシュボードをクリックできるようになりました。



図 9: local grafana

> **警告**
>
> 前のダッシュボードの一部のグラフは、*No data* を表示するいくつかのグラフを生成する `kubelet_volume_stats_available_bytes` や `kubelet_volume_stats_capacity_bytes` など、これが作成された時点でアルファ段階にあったメトリックを利用しています。

ローカル設定では、Prometheus と Grafana は、モニタリング機能を有効にして展開された CloudNativePG Cluster クラスターの EDB Postgres AI を自動的に検出および監視するように構成されていることに注意してください。

cluster-example.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
```

(次のページに続く)

(前のページからの続き)

```
  instances: 3

  storage:
    size: 1Gi
```

3.6 Preview Versions

CloudNativePG クラスター候補リリースの EDB Postgres AI は、コミュニティが新しい一般利用可能な GA リリースを発行する前に、テストに利用できるプレリリースバージョンです。これらのバージョンは機能が凍結されており、新しい機能が追加されず、最終リリースの前に公開テストを対象としています。

3.6.1 現在のプレビューバージョン

現在、利用可能なプレビューバージョンはありません。

3.6.2 リリース候補の目的

リリース候補は、公式リリースの前に広範なテストのためにコミュニティに提供されます。リリース候補は、CloudNativePG クラスターの EDB Postgres AI の新しいマイナーバージョンの最初のリリースと同一であることを目指していますが、GA リリースの前に追加の変更が実装される場合があります。

重大な問題の軽減

RC の主な目的は、自動テストと内部手動チェックが見逃す可能性のある重大な欠陥と回帰をキャッチすることです。**特に、コミュニティテストは、次のような現実のシナリオを検証するために非常に貴重です。**

- **アップグレードパス** CloudNativePG クラスタークラスターの既存の EDB Postgres AI を、以前の安定したマイナーバージョンから RC バージョンにアップグレードするプロセスのテスト。これにより、多くの場合、PVC の所有権に影響を与える問題など、既存の多様なユーザー設定でのみ確実にテストできる重要な互換性または移行の問題が明らかになります。
- **一意の構成**標準テストマトリックスでカバーされていない特定の Kubernetes バージョン、ストレージプロバイダー、ネットワークセットアップ、またはカスタム PostgreSQL 拡張機能との互換性の検証。
- **現実のワークロード**実際のアプリケーションのワークロードとトラフィックパターンで RC を実行して、パフォーマンスの低下または競合状態を明らかにします。

3.6.3 コミュニティへの参加: どのように支援するか

CloudNativePG Cluster マイナーリリースの各 EDB Postgres AI の安定性は、ワークロードとツールで次のバージョンをテストするコミュニティの努力に大きく依存しています。ユーザーテストを介してバグとリグレッションを特定することは、リリースを完成できる時期を決定する上で重要です。

行動喚起

リリース候補が利用可能な場合は、非実稼働環境で次のテストを実行することをお勧めします。

- **新しい展開テスト** RC バージョンを使用して、CloudNativePG Cluster クラスター用の新しい EDB Postgres AI を展開します。
- **アップグレードテスト重要な 既存のクラスター**を以前の安定したマイナーバージョンから RC バージョンにアップグレードし、すべての主要な操作面スケーリング、フェールオーバー、バックアップ/リストアが正しく機能することを確認します。
- **基本操作**ローリングアップデート、バックアップ/復元、インスタンスフェイルオーバー、接続プーリング機能などの主要機能が期待どおりに動作することを確認します。
- **問題の報告** GitHub の問題を開き、 ``**Release Candidate**`` タグまたはラベル、および RC バージョン番号たとえば `1.31.0-rc1` で明確にマーキングすることにより、問題または予期せぬ動作をすぐに報告します** 。

3.6.4 使用アドバイザリー

CloudNativePG Cluster コミュニティの EDB Postgres AI は、運用環境またはアクティブな開発プロジェクトで CloudNativePG Cluster の EDB Postgres AI のプレビューバージョンを使用しないことを強くお勧めします。CloudNativePG クラスターの EDB Postgres AI は、大規模な自動および手動テストを受けていますが、ベータリリースには重大なバグが含まれている可能性があります。プレビューバージョンの機能は、下位互換性のない方法で変更される場合があり、完全に削除される可能性があります。

リリース候補をテストすることにより、潜在的に重大なバグが GA リリース時にコミュニティ全体に影響を与えるのを防ぐことに役立ちます。

第4章　CONFIGURATION

4.1 PostgreSQL Configuration

PostgreSQL に慣れているユーザーは、インスタンスを構成するための次の 3 つのファイルの存在に気づいています。

- `postgresql.conf` PostgreSQL のメインランタイム構成ファイル
- `pg_hba.conf` クライアント認証ファイル
- `pg_ident.conf` 外部ユーザーを内部ユーザーにマッピング

PostgreSQL コンテナの宣言的構成と不変性の概念により、ユーザーはこれらのファイルに直接触れることはできません。`parameters` 、`pg_hba` 、および `pg_ident` キーを介してカスタム `postgresql.conf` 、`pg_hba.conf` 、および `pg_ident.conf` 設定を定義することにより、`Cluster` リソース定義の `postgresql` セクションから構成が可能です。

これらの設定は、すべてのインスタンスで同じです。

> **警告**
>
> *ALTER SYSTEM* クエリを使用して、命令的方法で PostgreSQL インスタンスの構成を変更しないでください。通常、オペレーターによって制御されるオプションの一部を変更すると、実際にクラスターの予測不能/回復不能な状態が発生する可能性があります。さらに、*ALTER SYSTEM* の変更はクラスター全体でレプリケートされません。詳細については、以下の Enabling ALTER SYSTEM を参照してください。

カスタム設定の使用に関するリファレンスはサンプルに含まれています。cluster-example-custom.yaml を参照してください。

> **警告**
>
> **OpenShift ユーザー** OpenShift ユーザーインターフェイスの現在の制限により、 YAML ペインからのみ PostgreSQL 設定を変更できます。

4.1.1 `postgresql` セクション

ポッド内の PostgreSQL インスタンスは、デフォルトの `postgresql.conf` ファイルで始まり、これらの設定が自動的に追加されます。

```
listen_addresses = *
include custom.conf
```

custom.conf ファイルには、次の例にあるように、postgresql セクションのユーザー定義設定が含まれます。

```
#  ...

postgresql:
  parameters:
    shared_buffers: "1GB"

#  ...
```

custom.conf のコンテンツは、次のセクションをこの順序で適用することにより、オペレーターによって自動的に生成および維持されます。

- グローバルデフォルトパラメーター
- PostgreSQL メジャーバージョンに依存するデフォルトパラメーター
- ユーザー指定パラメーター
- 固定パラメーター

グローバルデフォルトパラメーター は次のとおりです。

```
archive_timeout = 5min
dynamic_shared_memory_type = posix
full_page_writes = on
logging_collector = on
log_destination = csvlog
log_directory = /controller/log
log_filename = postgres
log_rotation_age = 0
log_rotation_size = 0
log_truncate_on_rotation = false
max_parallel_workers = 32
max_replication_slots = 32
max_worker_processes = 32
shared_memory_type = mmap
shared_preload_libraries =
ssl_max_protocol_version = TLSv1.3
ssl_min_protocol_version = TLSv1.3
wal_keep_size = 512MB
wal_level = logical
wal_log_hints = on
wal_sender_timeout = 5s
wal_receiver_timeout = 5s
```

> **警告**
>
> PostgreSQL クラスターでの WAL セグメント保持を計画し、予想および観察されたワークロードに基づいて、サーバーバージョンに応じて `wal_keep_size` または `wal_keep_segments` のいずれかを適切に構成するのはあなたの義務です。

また、唯一のストリーミングレプリケーションクライアントが高可用性クラスターで実行されているレプリカインスタンスである場合、クラスターレベルでレプリケーションスロットのサポートを追加するレプリケーションスロット機能を利用できます。`replicationSlots.highAvailability` オプションを使用して機能を有効にできます。詳細については、レプリケーション設定 を参照してください。

レプリケーションスロットも継続的バックアップも設定されていない場合、`wal_keep_size` または `wal_keep_segments` の構成がスタンバイを同期外れから保護する唯一の方法です。スタンバイが同期から外れると、`"could not receive data from WAL stream: ERROR: requested WAL segment **** **** **** **** **** **** has already been removed"` のようなエラーメッセージが生成されます。これには、`PGDATA` の一部、または WAL ファイルの保存専用のボリュームを専用にして、ストリーミングレプリケーションの目的で古い WAL セグメントを保持する必要があります。

```
archive_command = /controller/manager wal-archive %p
hot_standby = true
listen_addresses = *
port = 5432
restart_after_crash = false
ssl = on
ssl_ca_file = /controller/certificates/client-ca.crt
ssl_cert_file = /controller/certificates/server.crt
ssl_key_file = /controller/certificates/server.key
unix_socket_directories = /controller/run
```

固定パラメーターは最後に追加されるため、ユーザーが YAML 構成を介してオーバーライドすることはできません。これらのパラメーターは、正しい WAL アーカイブとレプリケーションに必要です。

先行書き込みログレベル

wal_level

PostgreSQL のパラメーターは、先行書き込みログ WAL に書き込まれる情報の量を決定します。次の値を受け入れます。

- `minimal` クラッシュリカバリーに必要な情報のみを書き込みます。
- `replica` スタンバイインスタンスで読み取り専用クエリを実行する機能など、WAL アーカイブとストリーミングレプリケーションをサポートするための十分な情報を追加します。
- `logical` `replica` からのすべての情報に加えて、論理デコードとレプリケーションに必要な追加情報が含まれます。

デフォルトでは、上流の PostgreSQL は `wal_level` を `replica` に設定します。 CloudNativePG クラスターの EDB Postgres AI は、代わりに、デフォルトで `wal_level` を `logical` に設定して、すぐに論理レプリ

ケーションを有効にします。これにより、外部の PostgreSQL サーバーからの移行などのユースケースのサポートが簡単になります。

クラスターが論理レプリケーションを必要としない場合は、`wal_level` を `replica` に設定して、WAL ボリュームとオーバーヘッドを削減することをお勧めします。

最後に、CloudNativePG クラスターの EDB Postgres AI では、WAL アーカイブが無効になっているシングルインスタンスクラスターに限り、`wal_level` を `minimal` に設定できます。

レプリケーション設定

`primary_conninfo` 、`restore_command` 、および `recovery_target_timeline` パラメーターは、クラスター内のインスタンスのロールに基づいて、オペレーターによって自動的に管理されます。これらのパラメーターは、インスタンスがレプリカとして動作している場合にのみ有効に適用されます。

```
primary_conninfo = host=<PRIMARY> user=postgres dbname=postgres tcp_user_timeout=5000
recovery_target_timeline = latest
```

ログ制御設定

オペレーターは、CSV 形式でログを出力するために PostgreSQL を必要とし、インスタンスマネージャーは自動的にそれを解析し、JSON 形式で出力します。その結果、Fixed parameters にリストされている特定の PostgreSQL ログ設定は固定され、変更できません。

詳細については、 *JSON 形式の PostgreSQL エラーメッセージの標準出力ログ* を参照してください。

共有プリロードライブラリ

PostgreSQL の `shared_preload_libraries` オプションは、サーバー起動時にコンマ区切りリストの形式でプリロードされる 1 つ以上の共有ライブラリを指定するために存在します。通常、PostgreSQL で使用され、システム全体のほとんどのデータベースセッションで使用する必要がある拡張機能がロードされます `pg_stat_statements` など。

CloudNativePG クラスターの EDB Postgres AI では、`shared_preload_libraries` オプションはデフォルトで空です。`shared_preload_libraries` のコンテンツをオーバーライドすることはできますが、熟練した Postgres ユーザーのみがこのオプションを利用することをお勧めします。

CloudNativePG クラスターの EDB Postgres AI は、最も使用される一部の PostgreSQL 拡張機能の `shared_preload_libraries` オプションのコンテンツを自動的に管理できます。詳細については、以下の **マネージド拡張機能** セクションを参照してください。

具体的には、構成パラメーターに管理ライブラリのいずれかが必要であることにオペレーターが気づくと、必要なライブラリが自動的に追加されます。オペレーターは、実際のパラメーターが必要としないとすぐにライブラリを削除します。

文字列のリストとして、`.spec.postgresql.shared_preload_libraries` を介して追加の `shared_preload_libraries` を提供できます。オペレーターは、それらを自動的に管理するものとマージします。

マネージド拡張機能

前のセクションで予想されたように、CloudNativePG クラスターの EDB Postgres AI は、いくつかのよく知られサポートされている拡張機能の `shared_preload_libraries` のコンテンツを自動的に管理します。現在のリストには以下が含まれます。

- `auto_explain`
- `pg_stat_statements`
- `pgaudit`
- `pg_failover_slots`

これらのライブラリの一部は、使用する前にデータベース内の追加オブジェクトも必要とします。通常は、データベースで実行する `CREATE EXTENSION` コマンドを介して管理されるビューおよび/またはファンクションです。通常、`DROP EXTENSION` コマンドはこれらのオブジェクトを削除します。

このようなライブラリの場合、CloudNativePG クラスターの EDB Postgres AI は、次のクエリーで識別される、クラスター内の接続を受け入れるすべてのデータベースの拡張機能の作成と削除を自動的に処理します。

```
SELECT datname FROM pg_database WHERE datallowconn
```

> **注釈**
>
> 上記のクエリーには、*template1* のようなテンプレートデータベースも含まれています。

`Database` CRD では、拡張機能を直接管理できるようになりました。その結果、マネージド拡張機能は、CloudNativePG クラスターの EDB Postgres AI の将来のバージョンで重要な変更が発生する可能性があり、一部の機能は非推奨になる可能性があります。

auto_explain の有効化

auto_explain

拡張機能は、 `EXPLAIN` を手動で実行することなく、遅いステートメントの実行プランを自動的にログに記録する手段を提供します最適化されていないクエリーを追跡するのに役立ちます。

次の抜粋例のように `auto_explain.` で始まるパラメーターを構成に追加することにより、`auto_explain` を有効にできます。完了までに 10 秒以上かかるクエリの実行プランを自動的にログに記録します。

```
#  ...

postgresql:
  parameters:
    auto_explain.log_min_duration: "10s"

#  ...
```

> **注釈**
>
> auto_explain を有効にすると、パフォーマンスの問題が発生する可能性があります。 the auto explain documentation を参照してください。

pg_stat_statements の有効化

pg_stat_statements

拡張機能は、クエリのリアルタイムモニタリングのために PostgreSQL で使用可能な最も重要な機能の 1 つです。

次の例の抜粋のように、`pg_stat_statements.` で始まるパラメーターを構成に追加することにより、`pg_stat_statements` を有効にできます。

```
#  ...

postgresql:
  parameters:
    pg_stat_statements.max: "10000"
    pg_stat_statements.track: all

#  ...
```

前述のように、オペレーターは `pg_stat_statements` を `shared_preload_libraries` に自動的に追加し、各データベースで `CREATE EXTENSION IF NOT EXISTS pg_stat_statements` を実行します。これにより、`pg_stat_statements` ビューに対してクエリーを実行できます。

pgaudit の有効化

`pgaudit` 拡張機能は、標準の PostgreSQL ロギング機能を介して詳細なセッションおよび/またはオブジェクト監査ログを提供します。

CloudNativePG クラスターの EDB Postgres AI は、PostgreSQL クラスターで PGAudit の透過的かつネイティブサポートを備えています。詳細については、:ref:``pgaudit` の有効化 <*pgaudit* の有効化>` を参照してください。

次の例の抜粋のように、`pgaudit.` で始まるパラメーターを構成に追加することにより、`pgaudit` を有効にできます。

```
#
postgresql:

  parameters:
    pgaudit.log: "all, -misc"
    pgaudit.log_catalog: "off"
    pgaudit.log_parameter: "on"
    pgaudit.log_relation: "on"
```

(次のページに続く)

(前のページからの続き)

```
#


####  Enabling `pg_failover_slots`

The [`pg_failover_slots`](https://github.com/EnterpriseDB/pg_failover_slots)
extension by EDB ensures that logical replication slots can survive a
failover scenario. Failovers are normally implemented using physical
streaming replication, like in the case of EDB Postgres AI for CloudNativePG Cluster.

You can enable `pg_failover_slots` by adding to the configuration a parameter
that starts with `pg_failover_slots.`: as explained above, the operator will
transparently manage the `pg_failover_slots` entry in the
`shared_preload_libraries` option depending on this.

Please refer to [`the `pg_failover_slots` documentation`](/pg_extensions/pg_failover_
↪slots)
for details on this extension.

Additionally, for each database that you intend to you use with `pg_failover_slots`
you need to add an entry in the `pg_hba` section that enables each replica to
connect to the primary.
For example, suppose that you want to use the `app` database with `pg_failover_slots`,
you need to add this entry in the `pg_hba` section:

``` yaml postgresql: pg_hba: - hostssl appstreaming_replica all cert
```

4.1.2 The pg_hba section

pg_hba is a list of PostgreSQL Host Based Authentication rules used to create the pg_hba.conf used by the pods.

!!! Note Important See the PostgreSQL documentation for *more information on ``pg_hba.conf`* <https://www.postgresql.org/docs/current/auth-pg-hba-conf.html>`__.

Since the first matching rule is used for authentication, the pg_hba.conf file generated by the operator can be seen as composed of four sections:

1. Fixed rules
2. User-defined rules
3. Optional LDAP section
4. Default rules

Fixed rules:

``` テキストローカル すべて すべて ピア

hostssl postgresstreaming_replica all cert map=cnp_streaming_replica hostssl replication stable_replica all cert map=cnp_streaming_replica hostssl all cnp_pooler_pgbouncer all certmap=cnp_pooler_pgbouncer

```
Default rules:

``` text host all all all <default-authentication-method>
```

From PostgreSQL 14 the default value of the `password_encryption` database parameter is set to `scram-sha-256`. Because of that, the default authentication method is `scram-sha-256` from this PostgreSQL version.

PostgreSQL 13 and older will use `md5` as the default authentication method.

The resulting `pg_hba.conf` will look like this:

``` テキストローカル すべて すべて ピア

hostssl postgresstreaming_replica all cert map=cnp_streaming_replica hostssl replication stable_replica all cert map=cnp_streaming_replica hostssl all cnp_pooler_pgbouncer all certmap=cnp_pooler_pgbouncer

host all all all scram-sha-256 #または PostgreSQL バージョン<= 13 の場合は md5
```

```
Inside the cluster manifest, `pg_hba` lines are added as list items
in `.spec.postgresql.pg_hba`, as in the following excerpt:

``` yaml postgresql: pg_hba: - hostssl app app 10.244.0.0/16 md5
```

In the above example we are enabling access for the `app` user to the `app` database using MD5 password authentication (you can use `scram-sha-256` if you prefer) via a secure channel (`hostssl`).

Dynamic address resolution with `podSelectorRefs`

In Kubernetes, pod IPs are ephemeral. Manually updating `pg_hba` rules every time a client pod restarts is unsustainable. The `.spec.podSelectorRefs` option automates this by allowing you to define **named label selectors** that the operator resolves into up-to-date IP addresses.

How it Works

1. **Define Selectors:** Map a friendly name to a standard Kubernetes `labelSelector` for pods.
2. **Reference Selectors:** Use the `${podselector:NAME}` placeholder in your `pg_hba` rules.
3. **Automatic Expansion:** The operator resolves matching pod IPs and the instance manager expands each reference into individual CIDR entries (`/32` for IPv4, `/128` for IPv6) within `pg_hba.conf`.

Configuration Example

The following snippet defines selectors for application and monitoring pods, then applies them to the PostgreSQL access rules:

``` yaml podSelectorRefs: - name: app-pods selector: matchLabels: app: myapp - name: モニタリングセレクター: matchLabels: role: モニタリング postgresql: pg_hba: - “hostssl mydb myuser ${podselector:app-pods} scram-sha-256” - “hostssl postgres Monitor ${podselector:monitoring} scram-sha-256”

```
IP Expansion Mapping

If the operator detects pods with IPs `10.0.0.5`, `10.0.0.12` (App), and
`10.0.1.3` (Monitoring), the instance manager transforms the template as
follows:

```テキスト

#  展開元 hostssl mydb myuser ${podselector:app-pods} scram-sha-256

hostssl mydb myuser 10.0.0.5/32 scram-sha-256 hostssl mydb myuser 10.0.0.12/32 scram-
↪sha-256

#  展開元 hostssl postgres Monitor ${podselector:monitoring} scram-sha-256

hostssl postgres Monitor 10.0.1.3/32 scram-sha-256
```

Key Constraints & Behavior

- **Scope:** Selectors are restricted to the **same namespace** as the Cluster for security. Cross-namespace lookups are not supported.
- **Placement:** The `${podselector:NAME}` syntax is valid **only in the address field** of host-type entries (`host`, `hostssl`, `hostnossl`, `hostgssenc`, `hostnogssenc`).
- **Reactivity:** The operator watches for pod lifecycle events (creation, deletion, or IP updates). Updates trigger an automatic configuration regeneration and a PostgreSQL `reload`.
- **Validation:** Selector names must follow the pattern `^[a-z]([a-z0-9_-]*[a-z0-9])?$` and each `${podselector:NAME}` reference in a `pg_hba` rule must correspond to a defined entry in `podSelectorRefs`. The webhook validates both constraints.

!!! warning

When a selector matches zero pods, the corresponding `pg_hba` lines referencing it are omitted from `pg_hba.conf`. Ensure that your default rules provide appropriate fallback access. !!!

For a complete example, see `cluster-example-pod-selector-refs.yaml <../https://raw.githubusercontent.com/EnterpriseDB/docs/main/product_docs/docs/postgres_for_kubernetes/1/samples/cluster-example-pod-selector-refs.yaml>`__.

LDAP Configuration

Under the `postgres` section of the cluster spec there is an optional `ldap` section available to define an LDAP configuration to be converted into a rule added into the `pg_hba.conf` file.

This will support two modes: `simple bind` mode which requires specifying a `server`, `prefix` and `suffix` in the LDAP section and the `search+bind` mode which requires specifying `server`, `baseDN`, `binDN`, and a `bindPassword` which is a secret containing the ldap password. Additionally, in `search+bind` mode you have the option to specify a `searchFilter` or `searchAttribute`. If no `searchAttribute` is specified the default one of `uid` will be used.

Additionally, both modes allow the specification of a `scheme` for ldapscheme and a `port`. Neither scheme nor port are required, however.

This section filled out for search+bind could look as follows:

``` yaml postgresql: ldap: server: “openldap.default.svc.cluster.local” bindSearchAuth:baseDN: “ou=org,dc=example,dc=com” bindDN: “cn=admin,dc=example,dc=com” bindPassword: name: “ldap-BindPassword” key: “data” searchAttribute: “uid”

```
The `pg_ident` section

`pg_ident` is a list of PostgreSQL User Name Maps that EDB Postgres AI for␣
↪CloudNativePG Cluster uses to
generate and maintain the ident map file (known as `pg_ident.conf`) inside the
data directory.

!!! Note Important
    See the PostgreSQL documentation for
    [more information on `pg_ident.conf`](https://www.postgresql.org/docs/current/
↪auth-username-maps.html).

The `pg_ident.conf` file written by the operator is made up of the following
two sections:

1. Fixed rules

2. User-defined rules

Currently the only fixed rule, automatically generated by the operator, is:

``` text local <postgres system user> postgres
```

The instance manager detects the user running the PostgreSQL instance and automatically adds a rule to map it to the `postgres` user in the database.

If the `postgres` user is not properly configured inside the container, the instance manager will allow any local user to connect and then log a warning message like the following:

``` text 現在のユーザーを識別できません。安全でないマッピングへのフォールバック。

```
The resulting `pg_ident.conf` will look like this:

``` text local <postgres system user> postgres

<ユーザー定義行>
```

Inside the cluster manifest, `pg_ident` lines are added as list items in `.spec.postgresql.pg_ident`. For example:

``` yaml postgresql: pg_ident: - “mymap /^(.*)@mydomain\.com$ \1”

```
Changing configuration

You can apply configuration changes by editing the `postgresql` section of
the `Cluster` resource.

After the change, the cluster instances will immediately reload the
configuration to apply the changes.
If the change involves a parameter requiring a restart, the operator will
perform a rolling upgrade.

Enabling `ALTER SYSTEM`

EDB Postgres AI for CloudNativePG Cluster strongly advocates employing the Cluster␣
↪manifest as the
exclusive method for altering the configuration of a PostgreSQL cluster. This
approach guarantees coherence across the entire high-availability cluster and
aligns with best practices for Infrastructure-as-Code.

In EDB Postgres AI for CloudNativePG Cluster the default configuration disables the␣
↪use of `ALTER SYSTEM`
on new Postgres clusters. This decision is rooted in the recognition of
potential risks associated with this command. To enable the use of `ALTER SYSTEM`,
you can explicitly set `.spec.postgresql.enableAlterSystem` to `true`.

!!! warning
    Proceed with caution when utilizing `ALTER SYSTEM`. This command operates
    directly on the connected instance and does not undergo replication.
    EDB Postgres AI for CloudNativePG Cluster assumes responsibility for certain␣
↪fixed parameters and complete
    control over others, emphasizing the need for careful consideration.

Starting from PostgreSQL 17, the `.spec.postgresql.enableAlterSystem` setting
directly controls the [`allow_alter_system` GUC in PostgreSQL](https://www.postgresql.
↪org/docs/17/runtime-config-compatible.html#GUC-ALLOW-ALTER-SYSTEM)
```

(次のページに続く)
```

(前のページからの続き)

```
— a feature directly contributed by EDB Postgres AI for CloudNativePG Cluster to␣
↪PostgreSQL.

Prior to PostgreSQL 17, when `.spec.postgresql.enableAlterSystem` is set to
`false`, the `postgresql.auto.conf` file is made read-only. Consequently, any
attempt to execute the `ALTER SYSTEM` command will result in an error. The
error message might look like this:

```出力エラー ファイル「postgresql.auto.conf」を開くことができませんでした。アクセスが拒否されました
```

4.1.3 Dynamic Shared Memory settings

PostgreSQL supports a few implementations for dynamic shared memory management through the `dynamic_shared_memory_type <https://www.postgresql.org/docs/current/runtime-config-resource.html#GUC-DYNAMIC-SHARED-MEMORY-TYPE>`__ configuration option. In EDB Postgres AI for CloudNativePG Cluster we recommend to limit ourselves to any of the following two values:

- `posix`: which relies on POSIX shared memory allocated using `shm_open` (default setting)
- `sysv`: which is based on System V shared memory allocated via `shmget`

In PostgreSQL, this setting is particularly important for memory allocation in parallel queries. For details, please refer to this *thread from the ``pgsql-general`* mailing list <https://www.postgresql.org/message-id/CA%2BhUKGJOj7qzDLxeFPVvto8YEWop6FSQoTYPO9Z6Ee%3Di-nPS_Q%40mail.gmail.com>`__.

POSIX shared memory

The default setting of `posix` should be enough in most cases, considering that the operator automatically mounts a *memory-bound ``EmptyDir`` volume* called `shm` under `/dev/shm`. You can verify the size of such volume inside the running Postgres container with:

```
sh mount |
```

4.1.4 Dynamic Shared Memory settings

PostgreSQL supports a few implementations for dynamic shared memory management through the `dynamic_shared_memory_type <https://www.postgresql.org/docs/current/runtime-config-resource.html#GUC-DYNAMIC-SHARED-MEMORY-TYPE>`__ configuration option. In EDB Postgres AI for CloudNativePG Cluster we recommend to limit ourselves to any of the following two values:

- `posix`: which relies on POSIX shared memory allocated using `shm_open` (default setting)
- `sysv`: which is based on System V shared memory allocated via `shmget`

In PostgreSQL, this setting is particularly important for memory allocation in parallel queries. For details, please refer to this *thread from the ``pgsql-general`* mailing list <https://www.postgresql.org/message-id/CA%2BhUKGJOj7qzDLxeFPVvto8YEWop6FSQoTYPO9Z6Ee%3Di-nPS_Q%40mail.gmail.com>`__.
```

POSIX shared memory

The default setting of `posix` should be enough in most cases, considering that the operator automatically mounts a *memory-bound ``EmptyDir`` volume* called `shm` under `/dev/shm`. You can verify the size of such volume inside the running Postgres container with:

``` grep shm

```
You should get something similar to the following output:

``` console shm on /dev/shm type tmpfs (rw,nosuid,nodev,noexec,relatime,size= **** **)
```

If you would like to set a maximum size for the `shm` volume, you can do so by setting the `.spec.ephemeralVolumesSizeLimit.shm` field in the `Cluster` resource. For example:

``` yaml spec 一時的なボリュームサイズ制限 shm 1Gi

```
System V shared memory

In case your Kubernetes cluster has a high enough value for the `SHMMAX`
and `SHMALL` parameters, you can also set:

``` yaml Dynamic_shared_memory_type: "sysv"
```

You can check the `SHMMAX`/`SHMALL` from inside a PostgreSQL container, by running:

``` sh ipcs -lm

```
For example:

``` console ------ 共有メモリ制限 -------- 最大セグメント数 = 4096 最大セグメントサイズキロバイト = 18014398509465599 最大合計共有メモリキロバイト = 18014398509481980 最小セグメントサイズバイト = 1
```

As you can see, the very high number of `max total shared memory` recommends setting `dynamic_shared_memory_type` to `sysv`.

An alternate method is to run:

``` sh cat /proc/sys/kernel/shmall cat /proc/sys/kernel/shmmax

```
Fixed parameters

Some PostgreSQL configuration parameters should be managed exclusively by the
operator. The operator prevents the user from setting them using a webhook.

Users are not allowed to set the following configuration parameters in the
`postgresql` section:
```

(次のページに続く)
```

(前のページからの続き)

```
- `allow_alter_system`

- `allow_system_table_mods`

- `archive_cleanup_command`

- `archive_command`

- `archive_mode`

- `bonjour`

- `bonjour_name`

- `cluster_name`

- `config_file`

- `data_directory`

- `data_sync_retry`

- `edb_audit`

- `edb_audit_destination`

- `edb_audit_directory`

- `edb_audit_filename`

- `edb_audit_rotation_day`

- `edb_audit_rotation_seconds`

- `edb_audit_rotation_size`

- `edb_audit_tag`

- `edb_log_every_bulk_value`

- `event_source`

- `external_pid_file`

- `hba_file`
```

(次のページに続く)

(前のページからの続き)

```
- `hot_standby`
- `ident_file`
- `jit_provider`
- `listen_addresses`
- `log_destination`
- `log_directory`
- `log_file_mode`
- `log_filename`
- `log_rotation_age`
- `log_rotation_size`
- `log_truncate_on_rotation`
- `logging_collector`
- `port`
- `primary_conninfo`
- `primary_slot_name`
- `promote_trigger_file`
- `recovery_end_command`
- `recovery_min_apply_delay`
- `recovery_target`
- `recovery_target_action`
- `recovery_target_inclusive`
- `recovery_target_lsn`
```

(次のページに続く)

(前のページからの続き)

```
- `recovery_target_name`

- `recovery_target_time`

- `recovery_target_timeline`

- `recovery_target_xid`

- `restart_after_crash`

- `restore_command`

- `shared_preload_libraries`

- `ssl`

- `ssl_ca_file`

- `ssl_cert_file`

- `ssl_crl_file`

- `ssl_dh_params_file`

- `ssl_ecdh_curve`

- `ssl_key_file`

- `ssl_passphrase_command`

- `ssl_passphrase_command_supports_reload`

- `ssl_prefer_server_ciphers`

- `stats_temp_directory`

- `synchronous_standby_names`

- `syslog_facility`

- `syslog_ident`

- `syslog_sequence_numbers`

- `syslog_split_messages`
```

(次のページに続く)

(前のページからの続き)

```
- `unix_socket_directories`

- `unix_socket_group`

- `unix_socket_permissions`

###  cluster-example-custom.yaml

``` yaml apiVersion: postgresql.k8s.enterprisedb.io/v1 kind: クラスターメタデータ:␣
↪name: cluster-example-custom spec: インスタンス数: 3

#  パラメーターと pg_hba 構成は、クラスターが機能するようにデフォルトのものに追加されます␣
↪postgresql: parameters: max_worker_processes: "60" pg_hba: # TCP/IP を介してアクセスす
るには、シークレットクラスターからユーザー名とパスワードを取得する必要があります- example-
↪custom-app - host all all all md5

# ローリング更新戦略の例 # - unsupervised すべての場合のプライマリの自動更新 # レプリカが
アップグレードされたデフォルト # - supervised を実行するには手動の監視が必要です # プライマ
リのスイッチオーバー primaryUpdateStrategy: unsupervised

# デフォルトのストレージクラスを使用して、インスタンスごとに 1Gi の領域が必要です storage:␣
↪size: 1Gi
```

cluster-example-pod-selector-refs.yaml

``` yaml apiVersion postgresql.k8s.enterprisedb.io/v1 kind クラスターメタデータ name cluster-example-pod-selector-refs spec インスタンス数 3

## 4.2 動的 pg_hba アドレス解決のための名前付けポッドラベルセレクターを定義します。 # オペレーターは、一致するポッド IP を解決し、${podselector:NAME} # pg_hba ルールの参照を/32 IPv4 または/128 IPv6 マスクを使用して IP ごとに 1 行に展開します。 podSelectorRefs: - name: app-pods selector: matchLabels: app: myapp - name: モニタリングセレクター:matchLabels:role: モニタリング

postgresql: pg_hba: #これらのルールは、${podselector:NAME}構文を使用して podSelectorRef を参照します。 #各参照は、一致するポッド IP ごとに 1 行に展開されます。 -“hostssl mydb myuser ${podselector:app-pods} scram-sha-256”-“hostssl postgres Monitor ${podselector:monitoring} scram-sha-256”# 展開のない標準ルールは変更せずにパススルーされます。 - host all all 10.244.0.0/16 md5

primaryUpdateStrategy 教師なし

ストレージ サイズ 1Gi ```
```
```

4.3 Operator configuration

CloudNativePG クラスターの EDB Postgres AI のオペレーターは、標準の展開マニフェストからインストールされ、構成パラダイムに関する規則に従います。ほとんどの場合、これで問題ありませんが、次のようなデフォルトの動作を変更する必要があるシナリオもあります。

- オペレーターが管理するすべての展開で共有される企業ライセンスキーの設定
- オペレーターが作成し、クラスターリソースに設定されているすべてのリソースに継承されるアノテーションとラベルの定義
- PostgreSQL の別のデフォルトイメージまたは追加のプルシークレットを定義

デフォルトでは、オペレーターは `postgresql-operator-controller-manager` と呼ばれる Kubernetes `Deployment` として `postgresql-operator-system` 名前空間にインストールされます。

> **注釈**
>
> 以下の例では、オペレーター展開のデフォルトの名前と名前空間を想定しています。

オペレーターの動作は、オペレーター展開と同じ名前空間にあり、名前が `postgresql-operator-controller-manager-config` を持つ `ConfigMap` /`Secret` を介してカスタマイズできます。

4.3.1 使用可能なオプション

オペレーターは、 `ConfigMap` /`Secret` で定義される次の環境変数を探します。

Name	Description
CERTIFICATE_DURATION	生成された証明書の有効期間を日単位で示します。デフォルト 90 です。
CLUSTERS_ROLLOUT_DELAY	オペレーターのアップグレード中にさまざまなクラスターのロールアウトの間で待機する時間秒単位。この設定は、クラスター全体でアップグレードのタイミングを制御し、アップグレードを分散してシステムへの影響を軽減します。デフォルト値は `0`で、PostgreSQL クラスターのアップグレードの間に遅延がないことを意味します。
CREATE_ANY_SERVICE	*true* `に設定すると、クラスターの-any`サービスを作成します。デフォルトは `false`です
DRAIN_TAINTS	ノードドレインのインジケーターとして解釈されるテイントキーを指定します。デフォルトでは、:ref:``kubectl` の `cnp` プラグインを使用する<*kubectl* の `cnp` プラグインを使用する>` 、Cluster Autoscaler 、および Karpenter *node.kubernetes.io/unschedulable*、*ToBeDeletedByClusterAutoscaler*、*karpenter.sh/disrupted*、`karpenter.sh/disruption`によって一般的に適用されるテイントが含まれます。

次のページに続く

表 1 – 前のページからの続き

Name	Description
EDB_LICENSE_KEY	デフォルトライセンスキー クラスターが定義していない場合にのみ使用され、できれば`Secret`
ENABLE_INSTANCE_MANAGER_INPL	*true*`に設定すると、オペレーターの更新後にインスタンスマネージャーのインプレース更新が有効になり、クラスターのローリング更新が回避されますデフォルト` *false*
ENABLE_REDWOOD_BY_DEFAULT	EPAS を使用する場合、デフォルトで Redwood 互換性を有効にします。
ENABLE_WEBHOOK_NAMESPACE_SI	*true*`に設定されている場合、オペレーターは-<OPERATOR_NAMESPACE>`接尾辞の名前の下で `MutatingWebhookConfiguration`および `ValidatingWebhookConfiguration`を検索しますたとえば *postgresql-operator-mutating-webhook-configuration-postgresql-operator-team-a*`および` *postgresql-operator-validating-webhook-configuration-postgresql-operator-team-a* 。名前の衝突を回避するために、同じクラスターで複数の名前空間オペレーターインスタンスを実行する場合にこれを使用します。オペレーターは、サフィックス付き構成を作成しません。このフラグを有効にしてオペレーターを開始する前に、それらを作成する必要があります。このフラグが有効になっているときに`OPERATOR_NAMESPACE`が設定されていない場合、オペレーターはスタートを拒否します。デフォルトは`false`です。
EXPIRING_CHECK_THRESHOLD	証明書の有効期限が間もなく識別されるためのしきい値を日単位で決定します。デフォルト 7 です。
EXTER-NAL_BACKUP_ADDON_CONFIGURA	*external-backup-adapter*`アドオンの構成。` *:ref:`#10743 <Add-ons>* を参照してください
INCLUDE_PLUGINS	クラスターの調整に常に含まれるプラグインのコンマ区切りリスト。
INHERITED_ANNOTATIONS	`Cluster`メタデータで定義されている場合、ポッドを含む生成されたすべてのリソースに継承されるアノテーション名のリスト
INHERITED_LABELS	`Cluster`メタデータで定義されている場合、ポッドを含む生成されたすべてのリソースに継承されるラベル名のリスト
INSTANCES_ROLLOUT_DELAY	オペレーターのアップグレード中に、同じクラスター内の個々の PostgreSQL インスタンスのロールアウトの間で待機する時間秒単位。デフォルト値は`0`で、同じ PostgreSQL クラスター内のインスタンスのアップグレードの間に遅延がないことを意味します。
KUBERNETES_CLUSTER_DOMAIN	Kubernetes クラスター内のサービス FQDN のドメインサフィックスを定義します。設定しない場合、デフォルトの「cluster.local」になります。

次のページに続く

表 1 – 前のページからの続き

Name	Description
MANAGE_WEBHOOK_CONFIGURATI	`true`デフォルトに設定されている場合、オペレーターは CA バンドルを `MutatingWebhookConfiguration`および `Validating-WebhookConfiguration`に注入します。 CA バンドルが外部から注入される場合、cert-manager の CA インジェクターまたは GitOps などによって、 `false`に設定されます。これは、オペレーターが PKI を所有するときに常に管理する Webhook サービング証明書とは無関係です。
METRICS_CERT_DIR	オペレーターメトリックサーバーの TLS 証明書が保存されているディレクトリ。設定すると、ポート 8080 でメトリックエンドポイントの TLS が有効になります。ディレクトリには、標準の Kubernetes TLS シークレット規則に従って `tls.crt`および `tls.key`ファイルが含まれている必要があります。設定しない場合、メトリックサーバーは TLS なしで動作しますデフォルトの動作。
MONITOR-ING_QUERIES_CONFIGMAP	作成されるすべてのクラスターに適用されるデフォルトのクエリー `queries`の下で指定されるオペレーターの名前空間の ConfigMap の名前
MONITORING_QUERIES_SECRET	作成されるすべてのクラスターに適用されるデフォルトのクエリー `queries`の下で指定されるオペレーターの名前空間のシークレットの名前
OPERATOR_IMAGE_NAME	ポッドのブートストラップに使用されるオペレーターイメージの名前。インストール中に指定されたイメージがデフォルトになります。
PGBOUNCER_IMAGE_NAME	新しいプーラーにデフォルトで使用される PgBouncer イメージの名前。演算子で指定されたバージョンのデフォルト。
POSTGRES_IMAGE_NAME	新しいクラスターにデフォルトで使用される PostgreSQL イメージの名前。演算子で指定されたバージョンのデフォルト。
PULL_SECRET_NAME	オペレーターの名前空間で定義され、イメージのダウンロードに使用される追加のプルシークレットの名前
STANDBY_TCP_USER_TIMEOUT	スタンバイインスタンスからプライマリへのレプリケーション接続の ``TCP_USER_TIMEOUT` socket option <https://www.postgresql.org/docs/current/runtime-config-connection.html#GUC-TCP-USER-TIMEOUT>`__ をミリ秒単位で定義します。デフォルトは 50005 秒です。システムのデフォルトを使用するには、 `0`に設定します。
WATCH_NAMESPACE	オペレーターがリソースを監視する名前空間を指定します。コンマで区切って複数の名前空間を指定できます。設定されていない場合、オペレーターはすべての名前空間を監視しますクラスター全体モード。

警告

CloudNativePG クラスターの EDB Postgres AI は、現在、同じクラスターでの複数のオペレーターの実行をサポートしていません。これは、 *ENABLE_WEBHOOK_NAMESPACE_SUFFIX* および `WATCH_NAMESPACE` を使用した名前空間展開を使用して実現できますが、複数のオペレーターが同じ共有 CRD セットを使用します。複数の同時実行オペレーターにわたる下位互換性のアップグレードは保証できません。

PULL_SECRET_NAME パラメーターを使用して追加のプルシークレット名を指定すると、オペレーターはそのシークレットを使用して、作成されたすべての PostgreSQL クラスターのプルシークレットを作成します。そのシークレットの名前は<cluster-name>-pull になります。

オペレーターが PULL_SECRET_NAME シークレットを検索する名前空間は、オペレーターをインストールした場所です。オペレーターがそのシークレットを見つけることができない場合、構成パラメーターは無視されます。

4.3.2 オペレーター構成マップの定義

次の例では、デフォルトのライセンスキーつまりカンパニーキー、 *in-place updates for the instancemanager* を有効にすることにより、後でデプロイされる Cluster オブジェクトによって作成されたリソースが継承するラベル/アノテーション名を定義することにより、オペレーターの動作をカスタマイズします。、そしてアップグレードを広めることによって。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: postgresql-operator-controller-manager-config
  namespace: postgresql-operator-system
data:
  CLUSTERS_ROLLOUT_DELAY: "60"
  ENABLE_INSTANCE_MANAGER_INPLACE_UPDATES: "true"
  EDB_LICENSE_KEY: <YOUR_BASE64_ENCODED_EDB_LICENSE_KEY_HERE>
  INHERITED_ANNOTATIONS: categories
  INHERITED_LABELS: environment, workload, app
  INSTANCES_ROLLOUT_DELAY: "10"
```

4.3.3 オペレーターシークレットの定義

次の例では、デフォルトのライセンスキーつまりカンパニーキー、後でデプロイされる Cluster オブジェクトによって作成されたリソースによって継承されるラベル/アノテーション名を定義し、有効にすることにより、オペレーターの動作をカスタマイズします。*in-place updates for the instancemanager* 、および分散アップグレードによる。

```
apiVersion: v1
kind: Secret
metadata:
  name: postgresql-operator-controller-manager-config
```

(次のページに続く)

(前のページからの続き)

```
  namespace: postgresql-operator-system
type: Opaque
stringData:
  CLUSTERS_ROLLOUT_DELAY: "60"
  ENABLE_INSTANCE_MANAGER_INPLACE_UPDATES: "true"
  EDB_LICENSE_KEY: <YOUR_BASE64_ENCODED_EDB_LICENSE_KEY_HERE>
  INHERITED_ANNOTATIONS: categories
  INHERITED_LABELS: environment, workload, app
  INSTANCES_ROLLOUT_DELAY: "10"
```

4.3.4 オペレーターを再起動して構成をリロードする

変更を有効にするには、オペレーターポッドを再作成して構成マップをリロードする必要があります。マニフェストを使用して Kubernetes にオペレーターをインストールしている場合、次のコマンドを発行してそれを行うことができます。

```
kubectl rollout restart deployment \
    -n postgresql-operator-system \
    postgresql-operator-controller-manager
```

それ以外の場合、OLM を使用してオペレーターをインストールしている場合、または Openshift で実行している場合、オペレーターがインストールされている名前空間を指定して次のコマンドを実行します。

```
kubectl delete pods -n [NAMESPACE_NAME_HERE] \
  -l app.kubernetes.io/name=cloud-native-postgresql
```

> **警告**
>
> カスタマイズは、オペレーター展開のリロード後に作成される `Cluster` リソースにのみ適用されます。

上記の例に従って、 `Cluster` 定義に `categories` アノテーションと `environment` 、 `workload` 、または `app` ラベルのいずれかが含まれている場合、これらはデプロイメントによって生成されたすべてのリソースに継承されます。

4.3.5 受付 Webhook を使用しないデフォルトと検証

CloudNativePG クラスターの EDB Postgres AI は、通常、デフォルトで、アドミッション Webhook を介してリソースを検証します。Webhook がインストールされていない場合、または一時的に応答できない場合、オペレーターは調整中にフォールバックとして同じデフォルト設定と検証を実行します。

リソースが検証に失敗すると、リコンシリエーションが停止し、その理由がリソースステータスに表示されます。`Cluster` および `Backup` は、それぞれ `Invalid cluster definition` および `invalid backup definition` フェーズに移動しますが、`Pooler` および `ScheduledBackup` は `.status.error` でそれを報告します。ステータスには、問題のフィールドパスのみがリストされます。フィールド値を含む場合がある

完全な検証エラーは、オペレーターログにのみ書き込まれます。仕様を修正するとエラーがクリアされ、リコンシリエーションが再開されます。

調整中に実行される検証では、受付 Webhook が行うのとまったく同様に、`k8s.enterprisedb.io/validation: disabled` アノテーションが尊重されます。

4.3.6 プロファイリングツール

オペレーターは、`localhost:6060` で pprof HTTP サーバーを公開できます。これを有効にするには、オペレーターの展開を編集し、フラグ`--pprof-server=true` をコンテナ引数に追加します。

```
kubectl edit deployment -n postgresql-operator-system postgresql-operator-controller-
↪manager
```

`--pprof-server=true` を引数リストに追加します。例

```
containers:
  - args:
      - controller
      - --leader-elect
      - --config-map-name=postgresql-operator-controller-manager-config
      - --secret-name=postgresql-operator-controller-manager-config
      - --log-level=info
      - --pprof-server=true # relevant line
    command:
      - /manager
```

保存後、展開がロールアウトされ、新しいポッドで pprof サーバーが有効になります。

ローカルマシンから pprof エンドポイントにアクセスするには、ポート転送を使用します。

```
kubectl port-forward -n postgresql-operator-system deploy/postgresql-operator-
↪controller-manager 6060
curl -sS http://localhost:6060/debug/pprof/
go tool pprof http://localhost:6060/debug/pprof/profile?seconds=30
```

ブラウザーを使用して pprof にアクセスすることもできます。http://localhost:6060/debug/pprof/。

> **警告**
>
> 上記の例では、ローカルテストにのみ`kubectl port-forward` を使用します。これは、運用環境で機能を公開するための意図された方法ではありません**。 pprof を機密デバッグインターフェイスとして扱い、決して公開しないでください。リモートでアクセスする必要がある場合は、適切なネットワークポリシーとアクセス制御で保護してください。

4.4 Instance Pod configuration

4.4.1 予測される量

CloudNativePG クラスターの EDB Postgres AI は、`.spec.projectedVolumeTemplate` を介して Postgres ポッド内のカスタムファイルのマウントをサポートしています。この機能は、追加のデータファイルを必要とするいくつかの Postgres 機能と拡張機能に役立ちます。CloudNativePG クラスターの EDB Postgres AI では、`.spec.projectedVolumeTemplate` フィールドは projected volume

Postgres ポッドの`/projected` フォルダーの下に任意のデータをマウントできる Kubernetes のテンプレート。

この単純な例は、既存の TLS シークレット `sample-secret` 名前付けをファイルとして Postgres ポッドにマウントする方法を示しています。`sample-secret` の秘密キー `tls.crt` および `tls.key` の値は、Postgres ポッドのパス`/projected/certificate/tls.crt` および`/projected/certificate/tls.key` にファイルとしてマウントされます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-projected-volumes
spec:
  instances: 3
  projectedVolumeTemplate:
    sources:
      - secret:
          name: sample-secret
          items:
            - key: tls.crt
              path: certificate/tls.crt
            - key: tls.key
              path: certificate/tls.key
  storage:
    size: 1Gi
```

投影されたボリュームテンプレートを使用してシークレットと ConfigMap をマウントする完全な例は、cluster-example-projected-volume.yaml にあります。

展開マニフェスト。

4.4.2 エフェメラルボリューム

CloudNativePG クラスターの EDB Postgres AI は ephemeral volumes に依存しています

社内活動の一部。一時ボリュームは、ポッドの存続期間が唯一の期間存在し、ポッドが再起動されても持続しません。

一時ストレージのボリューム要求テンプレート

オペレーターはデフォルトで emptyDir ボリュームを使用しますが、.spec.ephemeralVolumesSizeLimit field を使用してカスタマイズできます。これは、.spec.ephemeralVolumeSource フィールドでボリューム要求テンプレートを指定することにより、オーバーライドできます。

次の例では、1Gi エフェメラルボリュームが設定されます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-ephemeral-volume-source
spec:
  instances: 3
  ephemeralVolumeSource:
    volumeClaimTemplate:
      spec:
        accessModes: ["ReadWriteOnce"]
        # example storageClassName, replace with one existing in your Kubernetes␣
↪cluster
        storageClassName: "scratch-storage-class"
        resources:
          requests:
            storage: 1Gi
```

.spec.emphemeralVolumeSource と .spec.ephemeralVolumesSizeLimit.temporaryData は同時に指定できません。

共有メモリのボリューム

このボリュームは、Postgres の共有メモリ領域として、およびエフェメラルタイプとして使用されますが、メモリに保存されます。クラスター仕様の.spec.ephemeralVolumesSizeLimit.shm フィールドを使用して、サイズの上限を構成できます。このフィールドは、PostgreSQL running with `posix shared memory dynamic allocation <PostgreSQL Configuration>` の場合にのみ使用します。

4.4.3 環境変数

重要

オペレーター使用のために予約されている環境変数`PG` または`CNP_`、および`POD_NAME`、*NAMESPACE*、および`CLUSTER_NAME` で始まる名前は、*env* および`envFrom` フィールドを介して設定できず、受付時に拒否されます。

CloudNativePG クラスターの EDB Postgres AI では、クラスター仕様の env および envFrom スタンザを使用して、カスタム環境変数を設定できます。

この例では、デフォルトのクラスターレベルのタイムゾーンとして Australia/Sydney タイムゾーンを使用して PostgreSQL クラスターを定義します。

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3

  env:
    - name: TZ
      value: Australia/Sydney

  storage:
    size: 1Gi
```

envFrom スタンザは、ConfigMap またはシークレットを参照して、コンテンツを環境変数として使用できます。

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3

  envFrom:
    - configMapRef:
        name: config-map-name
    - secretRef:
        name: secret-name

  storage:
    size: 1Gi
```

オペレーターは、次の環境変数を設定することを許可していません。

- POD_NAME
- NAMESPACE
- 名前が PG で始まる環境変数。

env または envFrom セクションを変更すると、PostgreSQL ポッドのローリング更新がトリガーされます。

env または envFrom セクションがシークレットまたは ConfigMap を参照している場合、オペレーターはそれらの変更を検出せず、ロールアウトをトリガーしません。kubelet はポッドと同じ動作を使用するため、ポッドのロールアウトを手動でトリガーする必要があります。

cluster-example-projected-volume.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-projected-volume
spec:
  instances: 3
  projectedVolumeTemplate:
    sources:
      - secret:
          name: sample-secret
          items:
            - key: tls.crt
              path: certificate/tls.crt
            - key: tls.key
              path: certificate/tls.key
      - configMap:
          name: sample-configmap
          items:
            - key: key1
              path: config/key1
            - key: key2
              path: config/key2
  storage:
    size: 1Gi

- --
apiVersion: v1
data:
  tls.crt:␣
↪LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNQekNDQWVXZ0F3SUJBZ0lRSHVDY2lKcDVkZis0dHZBcDBrTk9rekFLQmdr
  tls.key:␣
↪LS0tLS1CRUdJTiBFQyBQUklWQVRFIEtFWS0tLS0tCk1IY0NBUUVFSUtucWxVQjFqU2sxWW5VZzAyb0tGbXlRdDJLUEZwaFc0K1lm
kind: Secret
metadata:
  name: sample-secret
type: kubernetes.io/tls
- --
apiVersion: v1
data:
  key1: value1
  key2: value2
  key3: value3
kind: ConfigMap
metadata:
```

(次のページに続く)

(前のページからの続き)

```
  name: sample-configmap
```

4.5 Examples

例は、PostgreSQL クラスターをセットアップするための構成ファイルを示しています。

4.5.1 基本

基本的なクラスター

cluster-example.yaml

クラスターの基本的な例。

EDB Postgres Advanced Server クラスター

cluster-example-epas.yaml

EPAS クラスターの基本的な例。

EDB Postgres ExtendedPGE クラスター

cluster-example-pge.yaml

PGE クラスターの基本的な例。

カスタムクラスター

cluster-example-custom.yaml

デフォルトのストレージクラスとカスタムパラメーターを使用する基本的なクラスター `postgresql.conf` および `pg_hba.conf` ファイル。

動的 pg_hba アドレス解決を使用したクラスター

cluster-example-pod-selector-refs.yaml

`podSelectorRefs` を使用して、`${podselector:NAME}` 構文を介して `pg_hba` ルールのポッド IP を動的に解決するクラスター。

カスタマイズされたストレージクラスを使用したクラスター

cluster-storage-class.yaml `standard` の指定されたストレージクラスを使用する基本クラスター。

永続的なボリューム要求 PVC テンプレートが構成されたクラスター

cluster-pvc-template.yaml 明示的な永続ボリューム要求テンプレートを使用した基本クラスター。

拡張構成例

cluster-example-full.yaml 使用可能なオプションのほとんどを設定するクラスター。

SQL ファイルを使用したブートストラップクラスター

cluster-example-initdb-sql-refs.yaml データベースが作成された直後に、シークレットと `ConfigMap` で定義されたクエリーセットを実行するクラスター例。

カスタマイズされた ``pg_hba`` 構成を使用したサンプルクラスター

cluster-example-pg-hba.yaml ユーザアプリが証明書を使用して認証できるようにする基本的なクラスター。

投影されたボリュームテンプレートを使用してマウントされたシークレットと ConfigMap を含むサンプルクラスター

cluster-example-projected-volume.yaml

投影されたボリュームマウントを使用して Postgres ポッドにマウントされた既存の `Secret` および `ConfigMap` を含む基本的なクラスター。

TDE が有効になったクラスター

cluster-example-tde.yaml

TDE を使用した EPAS 15 クラスター。使用するイメージをダウンロードするには、アクセス資格情報が必要であることに注意してください。

4.5.2 セキュリティ

カスタムセキュリティコンテキストを使用したサンプルクラスター

cluster-example-security-context.yaml

ポッドとコンテナの両方のセキュリティコンテキストをカスタマイズする方法を示すクラスター。これは、ポッドセキュリティ標準を使用する場合、または特定のセキュリティ要件を満たす場合に役立ちます。

4.5.3 バックアップ

カスタマイズされたストレージクラスとバックアップ

前提条件バケットストレージが利用できる必要があります。サンプル構成は AWS 用です。セットアップに合わせて変更します。

cluster-storage-class-with-backup.yaml

バックアップが構成されたクラスター。

バックアップ

Prerequisites cluster-storage-class-with-backup.yaml

展開され準備ができています。

backup-example.yaml 前のサンプルに対して実行されるバックアップの例。

minio 用にバックアップが構成された単純なクラスター

Prerequisites この構成は、minio が実行されて動作していることを前提としています。minio パラメーターまたはクラウドソリューションで `backup.barmanObjectStore` を更新します。

cluster-example-with-backup.yaml

バックアップが構成された基本的なクラスター。

Scaleway Object Storage 用に構成されたバックアップを含む単純なクラスター

Prerequisites この構成では、Scaleway Object Storage バケットが存在することを前提としています。`backup.barmanObjectStore` を Scaleway パラメーターで更新します。

cluster-example-with-backup-scaleway.yaml

Scaleway Object Storage で動作するように構成されたバックアップを含む基本的なクラスター。

4.5.4 レプリカクラスター

オブジェクトストアからのバックアップによるレプリカクラスター

Prerequisites cluster-storage-class-with-backup.yaml

導入して準備ができている、およびバックアップ cluster-example-trigger-backup.yaml

適用されて完了しました。

cluster-example-replica-from-backup-simple.yaml バックアップが構成されたクラスターに続くレプリカクラスター。

ボリュームスナップショットによるレプリカクラスター

Prerequisites cluster-example-with-volume-snapshot.yaml

展開され準備ができている、およびボリュームスナップショット backup-with-volume-snapshot.yaml

適用されて完了しました。

cluster-example-replica-from-volume-snapshot.yaml ボリュームスナップショットが構成されたクラスターに続くレプリカクラスター。

ストリーミング pg_basebackup を介したレプリカクラスター

Prerequisites cluster-example.yaml

展開され準備ができています。

cluster-example-replica-streaming.yaml ストリーミングレプリケーションを使用した `cluster-example` に続くレプリカクラスター。

4.5.5 PostGIS

画像ボリューム拡張機能を使用した PostGIS の例

postgis-example.yaml イメージボリューム拡張を使用した PostGIS クラスターの例。詳細は、*PostGIS* を参照してください。

4.5.6 管理対象ロール

宣言的ロール管理を使用したクラスター

cluster-example-with-roles.yaml `managed` スタンザを使用してロールを宣言します。Kubernetes シークレットを使用したパスワード管理が含まれます。

``DatabaseRole`` リソースを使用した宣言的ロール管理

role-examples.yaml 既存のクラスターをターゲットとするスタンドアロンの DatabaseRole リソース。Kubernetes シークレットによるパスワード管理を含む。

4.5.7 管理対象サービス

管理対象サービスを使用したクラスター

cluster-example-managed-services.yaml managed スタンザを使用してサービスを宣言します。デフォルトサービスが無効になり、定義された LoadBalancer タイプの新しい rw サービステンプレートが含まれます。

4.5.8 宣言的テーブルスペース

宣言的テーブルスペースを使用したクラスター

cluster-example-with-tablespaces.yaml

宣言的テーブルスペースとバックアップを使用したクラスター

Prerequisites この構成は、minio が実行されて動作していることを前提としています。minio パラメーターまたはクラウドソリューションで backup.barmanObjectStore を更新します。

cluster-example-with-tablespaces-backup.yaml

オブジェクトストアからテーブルスペースを使用して復元されたクラスター

Prerequisites 以前のクラスターが適用され、ベースバックアップが完了しました。bootstrap.recovery.backup.name をバックアップ名で更新することを忘れないでください。

cluster-restore-with-tablespaces.yaml

使用可能なオプションのリストについては、*API Reference - v1.30.0* を参照してください。

4.5.9 プーラー構成

カスタムサービス構成を使用したプーラー

pooler-external.yaml

4.5.10 宣言的なパブリケーションおよびサブスクリプションオブジェクトを介した論理レプリケーション

2 つのテストマニフェストには、論理レプリケーションのセットアップに必要なすべてが含まれています。

パブリケーションを含むソースクラスター

cluster-example-logical-source.yaml

app データベースで作成されたいくつかのテーブルを使用してクラスター cluster-example をセットアップし、重要なことに_app user_にレプリケーションを追加します。app データベースにクラスターのパブリケーションが作成されます。パブリケーションは、クラスターのプライマリが起動して実行された後にのみリコンサイルされることに注意してください。

サブスクリプションを使用した宛先クラスター

Prerequisites 上記のように定義された、パブリケーションを含むソースクラスター。

cluster-example-logical-destination.yaml

次を使用してクラスター `cluster-example-dest` を設定します。

- `externalClusters` スタンザで定義されたソースクラスター。 `app` ロールを使用して接続することに注意してください。これは、ソースクラスターが `replication` 特権を付与していることを前提としています。
- `schemaOnly` が有効になったマイクロサービスタイプのブートストラップインポート

サブスクリプションは宛先クラスターで作成されます。サブスクリプションは、宛先クラスターのプライマリが起動して実行された後にのみ調整されることに注意してください。

両方のクラスターがリコンサイルされた後、パブリケーションオブジェクトとサブスクリプションオブジェクトとともに、ソースクラスターのテーブルとそのデータが宛先クラスターでレプリケートされたことを確認できます。

さらに、いくつかのスタンドアロンのマニフェストの例があります。

すべてのテーブルをターゲットとしたプレーンなパブリケーション

Prerequisites 既存のクラスター `cluster-example` 。

publication-example.yaml

制約されたパブリケーションターゲットを含むパブリケーション

Prerequisites 既存のクラスター `cluster-example` 。

publication-example-objects.yaml

プレーンなサブスクリプション

前提条件 パブリケーション `pub-all` を使用して、ソースとして設定された既存のクラスター `cluster-example` 。宛先クラスターとして設定されたクラスター `cluster-example-dest` 。レプリケーション特権を持つロールを含む、ソースクラスターへの接続パラメーターを含む `externalClusters` スタンザを含みます。

subscription-example.yaml

上記のマニフェストはすべて、`app` データベースにパブリケーションまたはサブスクリプションを作成します。Database CRD は、データベースを宣言的に作成する便利な方法を提供します。これを使用すると、任意のデータベースに論理レプリケーションを設定できます。次のセクションに進みます。

4.5.11 Postgres データベースの宣言的管理

プレーンなデータベース

Prerequisites 既存のクラスター `cluster-example` 。

database-example.yaml

ICU ローカル仕様のデータベース

Prerequisites Postgres 16 以降を実行している既存のクラスター `cluster-example` 。

database-example-icu.yaml

cluster-example.yaml

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3

  storage:
    size: 1Gi
```

cluster-example-epas.yaml

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: postgresql-advanced-cluster
spec:
  instances: 3
  imageName: docker.enterprisedb.com/k8s/edb-postgres-advanced:18-standard-ubi9

  storage:
    size: 1Gi
```

cluster-example-pge.yaml

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: postgresql-extended-cluster
spec:
  instances: 3
  imageName: docker.enterprisedb.com/k8s/edb-postgres-extended:18-standard-ubi9

  storage:
    size: 1Gi
```

cluster-example-custom.yaml

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-custom
spec:
  instances: 3

  # Parameters and pg_hba configuration will be append
  # to the default ones to make the cluster work
  postgresql:
    parameters:
      max_worker_processes: "60"
    pg_hba:
      # To access through TCP/IP you will need to get username
      # and password from the secret cluster-example-custom-app
      - host all all all md5

  # Example of rolling update strategy:
  # - unsupervised: automated update of the primary once all
  #                 replicas have been upgraded (default)
  # - supervised: requires manual supervision to perform
  #               the switchover of the primary
  primaryUpdateStrategy: unsupervised

  # Require 1Gi of space per instance using default storage class
  storage:
    size: 1Gi
```

cluster-example-pod-selector-refs.yaml

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-pod-selector-refs
spec:
  instances: 3

  # Define named pod label selectors for dynamic pg_hba address resolution.
  # The operator resolves matching pod IPs and expands ${podselector:NAME}
  # references in pg_hba rules into one line per IP with /32 (IPv4) or /128 (IPv6)␣
↪masks.
  podSelectorRefs:
    - name: app-pods
```

(次のページに続く)

(前のページからの続き)

```
    selector:
      matchLabels:
        app: myapp
  - name: monitoring
    selector:
      matchLabels:
        role: monitoring

postgresql:
  pg_hba:
    # These rules use ${podselector:NAME} syntax to reference podSelectorRefs.
    # Each reference is expanded to one line per matching pod IP.
    - "hostssl mydb myuser ${podselector:app-pods} scram-sha-256"
    - "hostssl postgres monitor ${podselector:monitoring} scram-sha-256"
    # Standard rules without expansion are passed through unchanged.
    - host all all 10.244.0.0/16 md5

primaryUpdateStrategy: unsupervised

storage:
  size: 1Gi
```

cluster-storage-class.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: postgresql-storage-class
spec:
  instances: 3

  # Example of rolling update strategy:
  # - unsupervised: automated update of the primary once all
  #                 replicas have been upgraded (default)
  # - supervised: requires manual supervision to perform
  #               the switchover of the primary
  primaryUpdateStrategy: unsupervised

  # Persistent storage configuration
  storage:
    storageClass: standard
    size: 1Gi
```

cluster-pvc-template.yaml

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: postgresql-pvc-template
spec:
  instances: 3

  # Example of rolling update strategy:
  # - unsupervised: automated update of the primary once all
  #                 replicas have been upgraded (default)
  # - supervised: requires manual supervision to perform
  #               the switchover of the primary
  primaryUpdateStrategy: unsupervised

  # Persistent storage configuration
  storage:
    size: 1Gi
    pvcTemplate:
      accessModes:
        - ReadWriteOnce
      resources:
        requests:
          storage: 1Gi
      storageClassName: standard
      volumeMode: Filesystem
```

cluster-example-full.yaml

```yaml
#  Example of definition of a test cluster using all the elements available

#  in the CRD. Please change values appropriately for your environment.

#  Remember that you can take advantage of convention over configuration

#  and normally you dont need to use all these definitions.

apiVersion: v1
data:
  password:␣
↪VHhWZVE0bk44MlNTaVlIb3N3cU9VUlp2UURhTDRLcE5FbHNDRUVlOWJ3RHhNZDczS2NrSWVYelM1Y1U2TGlDMg==
  username: YXBw
kind: Secret
metadata:
```

(次のページに続く)

(前のページからの続き)

```
  name: cluster-example-app-user
type: kubernetes.io/basic-auth
- --
apiVersion: v1
data:
  password:␣
↪dU4zaTFIaDBiWWJDYzRUeVZBYWNCaG1TemdxdHpxeG1PVmpBbjBRSUNoc0pyU211OVBZMmZ3MnE4RUtLTHBaOQ==
  username: cG9zdGdyZXM=
kind: Secret
metadata:
  name: cluster-example-superuser
type: kubernetes.io/basic-auth
- --
apiVersion: v1
kind: Secret
metadata:
  name: backup-creds
data:
  ACCESS_KEY_ID: a2V5X2lk
  ACCESS_SECRET_KEY: c2VjcmV0X2tleQ==
- --
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-full
spec:
  description: "Example of cluster"
  imageName: docker.enterprisedb.com/k8s/postgresql:18.4-standard-ubi9
  # imagePullSecret is only required if the images are located in a private registry
  # imagePullSecrets:
  #   - name: private_registry_access
  instances: 3
  startDelay: 300
  stopDelay: 300
  primaryUpdateStrategy: unsupervised

  postgresql:
    parameters:
      shared_buffers: 256MB
      pg_stat_statements.max: 10000
      pg_stat_statements.track: all
      auto_explain.log_min_duration: 10s
    pg_hba:
      - host all all 10.244.0.0/16 md5
```

(次のページに続く)

(前のページからの続き)

```yaml
bootstrap:
  initdb:
    database: app
    owner: app
    secret:
      name: cluster-example-app-user
  # Alternative bootstrap method: start from a backup
  #recovery:
  #  backup:
  #    name: backup-example

enableSuperuserAccess: true
superuserSecret:
  name: cluster-example-superuser

storage:
  storageClass: standard
  size: 1Gi

backup:
  barmanObjectStore:
    destinationPath: s3://cluster-example-full-backup/
    endpointURL: http://custom-endpoint:1234
    s3Credentials:
      accessKeyId:
        name: backup-creds
        key: ACCESS_KEY_ID
      secretAccessKey:
        name: backup-creds
        key: ACCESS_SECRET_KEY
    wal:
      compression: gzip
      encryption: AES256
    data:
      compression: gzip
      encryption: AES256
      immediateCheckpoint: false
      jobs: 2
  retentionPolicy: "30d"

resources:
  requests:
    memory: "512Mi"
    cpu: "1"
  limits:
```

(次のページに続く)

(前のページからの続き)

```
      memory: "1Gi"
      cpu: "2"

  affinity:
    enablePodAntiAffinity: true
    topologyKey: failure-domain.beta.kubernetes.io/zone

  nodeMaintenanceWindow:
    inProgress: false
    reusePVC: false
```

cluster-example-initdb-sql-refs.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: post-init-sql-configmap
data:
  configmap.sql: |
    create table configmaps (i integer);
    insert into configmaps (select generate_series(1,10000));
- --
apiVersion: v1
kind: Secret
metadata:
  name: post-init-sql-secret
stringData:
  secret.sql: |
    create table secrets (i integer);
    insert into secrets (select generate_series(1,10000));
- --
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-initdb
spec:
  instances: 3

  bootstrap:
    initdb:
      database: appdb
      owner: appuser
      postInitSQL:
        - create table numbers (i integer)
```

(次のページに続く)

(前のページからの続き)

```
      - insert into numbers (select generate_series(1,10000))
    postInitTemplateSQL:
      - create extension intarray
    postInitApplicationSQL:
      - create table application_numbers (i integer)
      - insert into application_numbers (select generate_series(1,10000))
    postInitApplicationSQLRefs:
      configMapRefs:
      - name: post-init-sql-configmap
        key: configmap.sql
      secretRefs:
      - name: post-init-sql-secret
        key: secret.sql

  storage:
    size: 1Gi
```

cluster-example-pg-hba.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3
  postgresql:
    pg_hba:
      - hostssl app all all cert

  storage:
    size: 1Gi
```

cluster-example-projected-volume.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-projected-volume
spec:
  instances: 3
  projectedVolumeTemplate:
    sources:
      - secret:
```

(次のページに続く)

(前のページからの続き)

```
          name: sample-secret
          items:
            - key: tls.crt
              path: certificate/tls.crt
            - key: tls.key
              path: certificate/tls.key
      - configMap:
          name: sample-configmap
          items:
            - key: key1
              path: config/key1
            - key: key2
              path: config/key2
  storage:
    size: 1Gi

- --
apiVersion: v1
data:
  tls.crt:␣
↪LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNqekNDQWVXZ0F3SUJBZ0lRSHVDY2lKcDVkZis0dHZBcDBrTk9rekFLQmdr
  tls.key:␣
↪LS0tLS1CRUdJTiBFQyBQUklWQVRFIEtFWS0tLS0tCk1IY0NBUUVFSUtucWxVQjFqU2sxWW5VZzAyb0tGbXlRdDJLUEZwaFc0K1lm
kind: Secret
metadata:
  name: sample-secret
type: kubernetes.io/tls
- --
apiVersion: v1
data:
  key1: value1
  key2: value2
  key3: value3
kind: ConfigMap
metadata:
  name: sample-configmap
```

cluster-example-tde.yaml

```
- --
apiVersion: v1
kind: Secret
metadata:
  name: tde-key
```

(次のページに続く)

(前のページからの続き)

```
data:
  key:␣
↪bG9zcG9sbGl0b3NkaWNlbnBpb3Bpb3Bpb2N1YW5kb3RpZW5lbmhhbWJyZWN1YW5kb3RpZW5lbmZyaW8=

- --
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3
  imageName: docker.enterprisedb.com/k8s/edb-postgres-advanced:18-standard-ubi9
  postgresql:
    epas:
      tde:
        enabled: true
        secretKeyRef:
          name: tde-key
          key: key

  storage:
    size: 1Gi
```

cluster-example-security-context.yaml

```
#  Example of PostgreSQL cluster with custom security contexts

#

#  This example demonstrates how to customize both PodSecurityContext and

#  Container SecurityContext for a PostgreSQL cluster. This is particularly

#  useful when working with Pod Security Standards (PSS) or when you need

#  to meet specific security requirements.

#
apiVersion: postgresql.k8s.enterprisedb.io/v1

kind: Cluster
metadata:
  name: cluster-security-context
spec:
```

(次のページに続く)

(前のページからの続き)

```
  instances: 3

  # Storage configuration
  storage:
    size: 1Gi

  # Custom PodSecurityContext
  # This will be applied to all pods in the cluster and merged with operator defaults.
  # Only RunAsUser, RunAsGroup, and SeccompProfile are merged from defaults if not␣
↪specified.
  podSecurityContext:
    runAsUser: 26
    runAsGroup: 26
    fsGroup: 26
    runAsNonRoot: true
    supplementalGroups: [1000, 2000]
    fsGroupChangePolicy: "OnRootMismatch"

  # Custom Container SecurityContext
  # This will be applied to all containers in the cluster pods and merged with␣
↪operator defaults.
  # The operator provides secure defaults for all fields, which will be used if not␣
↪explicitly set.
  securityContext:
    allowPrivilegeEscalation: false
    # Note: capabilities are not merged with operator defaults.
    # If specified, they fully replace any defaults.
    capabilities:
      drop:
      - ALL
      add:
      - NET_BIND_SERVICE
    privileged: false
    readOnlyRootFilesystem: true
    runAsNonRoot: true
```

cluster-storage-class-with-backup.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: pg-backup
spec:
  instances: 3
```

(次のページに続く)

(前のページからの続き)

```
  # Example of rolling update strategy:
  # - unsupervised: automated update of the primary once all
  #                 replicas have been upgraded (default)
  # - supervised: requires manual supervision to perform
  #               the switchover of the primary
  primaryUpdateStrategy: unsupervised

  # Persistent storage configuration
  storage:
    storageClass: standard
    size: 1Gi

  # Backup properties
  backup:
    barmanObjectStore:
      destinationPath: s3://BUCKET_NAME/path/to/folder
      s3Credentials:
        accessKeyId:
          name: aws-creds
          key: ACCESS_KEY_ID
        secretAccessKey:
          name: aws-creds
          key: ACCESS_SECRET_KEY
      wal:
        compression: gzip
```

backup-example.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Backup
metadata:
  name: pg-backup-example
spec:
  cluster:
    name: pg-backup
```

cluster-example-with-backup.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-with-backup
```

(次のページに続く)

(前のページからの続き)

```
spec:
  instances: 3
  primaryUpdateStrategy: unsupervised

  # Persistent storage configuration
  storage:
    storageClass: csi-hostpath-sc
    size: 1Gi

  # Backup properties
  # This assumes a local minio setup
  backup:
    barmanObjectStore:
      destinationPath: s3://backups/
      endpointURL: http://minio:9000
      s3Credentials:
        accessKeyId:
          name: minio
          key: ACCESS_KEY_ID
        secretAccessKey:
          name: minio
          key: ACCESS_SECRET_KEY
      wal:
        compression: gzip
      data:
        additionalCommandArgs:
          - "--min-chunk-size=5MB"
          - "--read-timeout=60"
          - "-vv"
```

cluster-example-with-backup-scaleway.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: pg-backup-scaleway
spec:
  instances: 3
  storage:
    storageClass: standard
    size: 1Gi
  backup:
    barmanObjectStore:
      destinationPath: "s3://<bucket>/backups/"     # change <bucket> with your␣
```

(次のページに続く)

(前のページからの続き)

```
↪buckets name.
      endpointURL: "https://s3.<region>.scw.cloud"  # change <region> with your␣
↪buckets location/region.
      s3Credentials:
        accessKeyId:
          name: scaleway
          key: ACCESS_KEY_ID
        secretAccessKey:
          name: scaleway
          key: ACCESS_SECRET_KEY
        region:
          name: scaleway
          key: ACCESS_REGION
```

cluster-example-trigger-backup.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Backup
metadata:
  name: cluster-example-trigger-backup
spec:
  cluster:
    name: cluster-example-with-backup
```

cluster-example-replica-from-backup-simple.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-replica-from-backup-simple
spec:
  instances: 1

  bootstrap:
    recovery:
      source: cluster-example-backup

  replica:
    enabled: true
    source: cluster-example-backup

  storage:
    size: 1Gi
```

(次のページに続く)

(前のページからの続き)

```
  externalClusters:
  - name: cluster-example-backup
    barmanObjectStore:
      destinationPath: s3://backups/
      endpointURL: http://minio:9000
      s3Credentials:
        accessKeyId:
          name: minio
          key: ACCESS_KEY_ID
        secretAccessKey:
          name: minio
          key: ACCESS_SECRET_KEY
```

cluster-example-with-volume-snapshot.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-with-volume-snapshot
spec:
  instances: 3
  primaryUpdateStrategy: unsupervised

  # Persistent storage configuration
  storage:
    storageClass: csi-hostpath-sc
    size: 1Gi
  walStorage:
    storageClass: csi-hostpath-sc
    size: 1Gi

  # Backup properties
  backup:
    volumeSnapshot:
       className: csi-hostpath-snapclass
    barmanObjectStore:
      destinationPath: s3://backups/
      endpointURL: http://minio:9000
      s3Credentials:
        accessKeyId:
          name: minio
          key: ACCESS_KEY_ID
        secretAccessKey:
```

(次のページに続く)

(前のページからの続き)

```
        name: minio
        key: ACCESS_SECRET_KEY
    wal:
      compression: gzip
```

backup-with-volume-snapshot.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Backup
metadata:
  name: backup-with-volume-snapshot
spec:
  method: volumeSnapshot
  cluster:
    name: cluster-example-with-volume-snapshot
```

cluster-example-replica-from-volume-snapshot.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-replica-from-snapshot
spec:
  instances: 1

  storage:
    storageClass: csi-hostpath-sc
    size: 1Gi
  walStorage:
    storageClass: csi-hostpath-sc
    size: 1Gi

  bootstrap:
    recovery:
      source: cluster-example-with-volume-snapshot
      volumeSnapshots:
        storage:
          name: cluster-example-with-volume-snapshot-2-1692618163
          kind: VolumeSnapshot
          apiGroup: snapshot.storage.k8s.io
        walStorage:
          name: cluster-example-with-volume-snapshot-2-wal-1692618163
          kind: VolumeSnapshot
```

(次のページに続く)

(前のページからの続き)

```
        apiGroup: snapshot.storage.k8s.io

  replica:
    enabled: true
    source: cluster-example-with-volume-snapshot

  externalClusters:
    - name: cluster-example-with-volume-snapshot

      connectionParameters:
        host: cluster-example-with-volume-snapshot-rw.default.svc
        user: postgres
        dbname: postgres
      password:
        name: cluster-example-with-volume-snapshot-superuser
        key: password

      barmanObjectStore:
        destinationPath: s3://backups/
        endpointURL: http://minio:9000
        s3Credentials:
          accessKeyId:
            name: minio
            key: ACCESS_KEY_ID
          secretAccessKey:
            name: minio
            key: ACCESS_SECRET_KEY
        wal:
          maxParallel: 8
```

cluster-example-replica-streaming.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-replica-example
spec:
  instances: 1

  bootstrap:
    pg_basebackup:
      source: cluster-example

  replica:
```

(次のページに続く)

(前のページからの続き)

```
    enabled: true
    source: cluster-example

  storage:
    size: 1Gi
  # note the namespace default in the host name `cluster-example-rw.default.svc`
  # remember to change accordingly with the namespace of the main cluster
  externalClusters:
  - name: cluster-example
    connectionParameters:
      host: cluster-example-rw.default.svc
      user: streaming_replica
      sslmode: verify-full
      dbname: postgres
    # NOTE: if this cluster is created in a different namespace than the main cluster
    # remember to create the `-replication` and `-ca` secrets in the follower namespace
    # before creating the follower cluster
    sslKey:
      name: cluster-example-replication
      key: tls.key
    sslCert:
      name: cluster-example-replication
      key: tls.crt
    sslRootCert:
      name: cluster-example-ca
      key: ca.crt
```

postgis-example.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: postgis-example
spec:
  imageName: docker.enterprisedb.com/k8s/postgresql:18.4-minimal-ubi9
  instances: 1

  storage:
    size: 1Gi

  postgresql:
    extensions:
    - name: postgis
      image:
```

(次のページに続く)

(前のページからの続き)

```
        reference: ghcr.io/cloudnative-pg/postgis-extension:3.6.1-18-trixie
      ld_library_path:
      - system
- --
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Database
metadata:
  name: postgis-example-app
spec:
  name: app
  owner: app
  cluster:
    name: postgis-example
  extensions:
  - name: postgis
    version: 3.6.1
  - name: postgis_raster
  - name: postgis_sfcgal
  - name: fuzzystrmatch
  - name: address_standardizer
  - name: address_standardizer_data_us
  - name: postgis_tiger_geocoder
  - name: postgis_topology
```

cluster-example-with-roles.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-with-roles
spec:
  instances: 3
  storage:
    size: 1Gi

  managed:
    roles:
    - name: app
      createdb: true
      login: true
    - name: dante
      ensure: present
      comment: my database-side comment
      login: true
```

(次のページに続く)

(前のページからの続き)

```
      superuser: false
      createdb: true
      createrole: false
      inherit: false
      replication: false
      bypassrls: false
      connectionLimit: 4
      validUntil: "2053-04-12T15:04:05Z"
      inRoles:
        - pg_monitor
        - pg_signal_backend
      passwordSecret:
        name: cluster-example-dante
- --
apiVersion: v1
data:
  username: ZGFudGU=
  password: ZGFudGU=
kind: Secret
metadata:
  name: cluster-example-dante
type: kubernetes.io/basic-auth
```

role-examples.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: DatabaseRole
metadata:
  name: cluster-example-reporting
spec:
  cluster:
    name: cluster-example
  name: reporting
  login: true
- --
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: DatabaseRole
metadata:
  name: role-dante
spec:
  cluster:
    name: cluster-example
  name: dante
  comment: my database-side comment
```

(次のページに続く)

(前のページからの続き)

```
  login: true
  superuser: false
  createdb: true
  createrole: false
  inherit: false
  clientCertificate:
    enabled: true
  replication: false
  bypassrls: false
  connectionLimit: 4
  validUntil: "2053-04-12T15:04:05Z"
  inRoles:
    - pg_monitor
    - pg_signal_backend
  passwordSecret:
    name: cluster-example-dante
- --
apiVersion: v1
data:
  username: ZGFudGU=
  password: ZGFudGU=
kind: Secret
metadata:
  name: cluster-example-dante
  labels:
    k8s.enterprisedb.io/reload: "true"
type: kubernetes.io/basic-auth
```

cluster-example-managed-services.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-managed-services
spec:
  instances: 1
  storage:
    size: 1Gi

  managed:
    services:
      ## disable the default services
      disabledDefaultServices: ["ro", "r"]
      additional:
```

(次のページに続く)

(前のページからの続き)

```
      - selectorType: rw
        serviceTemplate:
          metadata:
            name: "test-rw"
            labels:
              test-label: "true"
            annotations:
              test-annotation: "true"
          spec:
            type: LoadBalancer
```

cluster-example-with-tablespaces.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-with-tablespaces
spec:
  instances: 3

  storage:
    size: 1Gi

  tablespaces:
    - name: atablespace
      storage:
        size: 1Gi
        storageClass: standard
      temporary: true
    - name: another_tablespace
      storage:
        size: 2Gi
        storageClass: standard
      temporary: true
    - name: tablespacea1
      storage:
        size: 2Gi
        storageClass: standard
```

cluster-example-with-tablespaces-backup.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
```

(次のページに続く)

(前のページからの続き)

```
metadata:
  name: cluster-example-with-tablespaces
spec:
  instances: 3

  storage:
    size: 1Gi

  backup:
    barmanObjectStore:
      destinationPath: s3://backups/
      endpointURL: http://minio:9000
      s3Credentials:
        accessKeyId:
          name: minio
          key: ACCESS_KEY_ID
        secretAccessKey:
          name: minio
          key: ACCESS_SECRET_KEY
      wal:
        compression: gzip

  tablespaces:
    - name: atablespace
      storage:
        size: 1Gi
        storageClass: standard
    - name: another_tablespace
      storage:
        size: 2Gi
        storageClass: standard
    - name: tablespacea1
      storage:
        size: 2Gi
        storageClass: standard
```

cluster-restore-with-tablespaces.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-restore-with-tablespaces
spec:
  instances: 3
```

(次のページに続く)

(前のページからの続き)

```
  storage:
    size: 1Gi

  bootstrap:
    recovery:
      backup:
        name: cluster-example-with-tablespaces-20231128093940

  tablespaces:
    atablespace:
      storage:
        size: 1Gi
        storageClass: standard
    another_tablespace:
      storage:
        size: 2Gi
        storageClass: standard
    tablespacea1:
      storage:
        size: 2Gi
        storageClass: standard
```

pooler-external.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Pooler
metadata:
  name: pooler-example-rw
spec:
  cluster:
    name: cluster-example
  instances: 3
  type: rw
  serviceTemplate:
    metadata:
      labels:
        app: pooler
    spec:
      type: LoadBalancer
  pgbouncer:
    poolMode: session
    parameters:
      max_client_conn: "1000"
```

(次のページに続く)

(前のページからの続き)

```
    default_pool_size: "10"
```

cluster-example-logical-source.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: DatabaseRole
metadata:
  name: cluster-example-app
spec:
  cluster:
    name: cluster-example
  name: app
  login: true
  replication: true
- --
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3

  imageName: docker.enterprisedb.com/k8s/postgresql:18-standard-ubi9

  storage:
    size: 1Gi

  bootstrap:
    initdb:
      postInitApplicationSQL:
        - CREATE TABLE numbers (i SERIAL PRIMARY KEY, m INTEGER)
        - INSERT INTO numbers (m) (SELECT generate_series(1,10000))
        - ALTER TABLE numbers OWNER TO app
        - CREATE TABLE numbers_two (i SERIAL PRIMARY KEY, m INTEGER)
        - INSERT INTO numbers_two (m) (SELECT generate_series(1,10000))
        - ALTER TABLE numbers_two OWNER TO app
        - CREATE SCHEMA another_schema
        - ALTER SCHEMA another_schema OWNER TO app
        - CREATE TABLE another_schema.numbers_three (i SERIAL PRIMARY KEY, m INTEGER)
        - INSERT INTO another_schema.numbers_three (m) (SELECT generate_series(1,
↪10000))
        - ALTER TABLE another_schema.numbers_three OWNER TO app

  replicationSlots:
```

(次のページに続く)

(前のページからの続き)

```
    highAvailability:
      synchronizeLogicalDecoding: true

  postgresql:
    parameters:
      hot_standby_feedback: on
      sync_replication_slots: on

- --
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Publication
metadata:
  name: cluster-example-pub
spec:
  name: pub
  dbname: app
  cluster:
    name: cluster-example
  target:
    allTables: true
```

cluster-example-logical-destination.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-dest
spec:
  instances: 1

  storage:
    size: 1Gi

  bootstrap:
    initdb:
      import:
        type: microservice
        schemaOnly: true
        databases:
          - app
        source:
          externalCluster: cluster-example

  externalClusters:
```

(次のページに続く)

(前のページからの続き)

```
  - name: cluster-example
    connectionParameters:
      host: cluster-example-rw.default.svc
      user: app
      dbname: app
    password:
      name: cluster-example-app
      key: password
- --
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Subscription
metadata:
  name: cluster-example-dest-sub
spec:
  cluster:
    name: cluster-example-dest
  name: sub
  dbname: app
  publicationName: pub
  externalClusterName: cluster-example
  parameters:
    failover: true
```

Publication-example.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Publication
metadata:
  name: publication-example
spec:
  cluster:
    name: cluster-example
  name: pub-all
  dbname: app
  target:
    allTables: true
```

Publication-example-objects.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Publication
metadata:
  name: publication-example-objects
```

(次のページに続く)

(前のページからの続き)

```
spec:
  cluster:
    name: cluster-example
  name: pub-objects
  dbname: app
  target:
    objects:
      - tablesInSchema: public
      - table:
          schema: another_schema
          name: numbers_three
          only: true
```

subscription-example.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Subscription
metadata:
  name: subscription-sample
spec:
  name: sub
  dbname: app
  publicationName: pub-all
  cluster:
    name: cluster-example-dest
  externalClusterName: cluster-example
```

Database-example.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Database
metadata:
  name: db-one
spec:
  name: one
  owner: app
  cluster:
    name: cluster-example
```

Database-example-icu.yaml

```yaml
#  NOTE: this manifest will only work properly if the Postgres version supports

#  ICU locales and rules (version 16 and newer)

apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Database
metadata:
  name: db-icu
spec:
  name: declarative-icu
  owner: app
  encoding: UTF8
  localeProvider: icu
  icuLocale: en
  icuRules: fr
  template: template0
  cluster:
    name: cluster-example
```

第5章　USING

5.1 Bootstrap

> **注釈**
>
> このセクションで「PostgreSQL クラスター」を参照する場合、特に断りのない限り、同じ概念が PostgreSQL と EDB Postgres Advanced Server の両方に適用されます。

このセクションでは、新しい PostgreSQL クラスターを作成するために使用可能なオプションと、その背後にある設計理論的根拠について説明します。新しいクラスターをブートストラップするには、主に 2 つの方法があります。

- スクラッチから `initdb`
- 既存の PostgreSQL クラスターから直接 `pg_basebackup` または物理ベースバックアップ `recovery` を介して間接的に

`initdb` ブートストラップは、Kubernetes の外部にある場合、または PostgreSQL の別のメジャーバージョンを実行している場合でも、既存の PostgreSQL クラスターから 1 つ以上のデータベースをインポートするオプションも提供します。この機能の詳細については、*Importing Postgres databases* セクションを参照してください。

> **警告**
>
> CloudNativePG クラスターの EDB Postgres AI では、 *postgres* ユーザーとデータベースの両方が常に存在する必要があります。クラスターで管理タスクを実行するには、ローカルの Unix ドメインソケットを使用して、 *postgres* ユーザーとして `peer` 認証を介して `postgres` データベースに接続する必要があります。 *postgres* ユーザーまたは `postgres` データベースは**削除しないでください**。

> **注釈**
>
> CloudNativePG クラスターの EDB Postgres AI は、*Kubernetes' native `VolumeSnapshot` API <https://github.com/cloudnative-pg/cloudnative-pg/issues/2081>`_ のサポートを段階的に導入しています

バックアップおよびリカバリ操作におけるインクリメンタルおよび差分コピーの場合- 基になるストレージクラスでサポートされている場合。*Recovery from Volume Snapshot objects* を参照してください。

詳細については。

5.1.1 bootstrap セクション

_bootstrap_メソッドは、クラスター仕様の bootstrap セクションで定義できます。 CloudNativePG クラスターの EDB Postgres AI は、現在次のブートストラップ方法をサポートしています。

- initdb 新しい PostgreSQL クラスターを初期化しますデフォルト
- recovery 既存のクラスターのベースバックアップから復元し、必要に応じて、使用可能なすべての WAL ファイルまたは指定された_ポイントインタイム_までを再生することにより、PostgreSQL クラスターを作成します
- pg_basebackup ストリーミングレプリケーションプロトコルを介して pg_basebackup を使用して、同じメジャーバージョンの既存のクローンを作成することにより、PostgreSQL クラスターを作成します。この方法は、CloudNativePG クラスターの EDB Postgres AI にデータベースを移行する場合に特に役立ちますが、すべての要件を満たすのは難しい場合があります。ライブクラスターからのブートストラップ`pg_basebackup <ライブクラスターからのブートストラップ *pg_basebackup*>` の警告を必ずご確認ください。

注意深く。

マニフェストでは 1 つのブートストラップメソッドのみを指定できます。複数のブートストラップメソッドを定義しようとすると、検証エラーが発生します。

initdb メソッドとは対照的に、 recovery と pg_basebackup は両方とも別のクラスターオフラインまたはオンラインに基づいて新しいクラスターを作成し、レプリカクラスターをスピンアップするために使用できます。どちらも外部クラスターの定義に依存します。詳細については、*replica cluster section* を参照してください。

CloudNativePG クラスターオペレーターの EDB Postgres AI が recovery に提供する考えられるバックアップ方法とバックアップストレージの組み合わせの量を考えると、各方法のガイダンスについては専用の **オブジェクトストアからのリカバリー** を参照してください。

詳細については、

5.1.2 externalClusters セクション

クラスターマニフェストの externalClusters セクションを使用して、1 つ以上の PostgreSQL クラスターへのアクセスを_sources_として構成できます。主な使用例には次のものが含まれます。

1. **データベースのインポート** initdb ブートストラップ方法の一部として、論理バックアップとリストアを介して *importation of databases* 中に利用する外部ソースを指定します。
2. **クロスリージョンレプリケーション**物理レプリケーションを採用するクロスリージョン PostgreSQL クラスターを定義し、個別の Kubernetes クラスターまたは従来の VM/ベアメタル環境に拡張できます。
3. **物理ベースバックアップからの復旧**物理ベースバックアップを参照することにより、PostgreSQL クラスターを完全にまたは特定のポイントインタイムに復旧します。

> **注釈**
>
> 進行中の開発は`externalClusters` の機能を拡張して、将来のリリースでの論理レプリケーションやフォーリンサーバーなどの追加のユースケースに対応します。

ブートストラップに関する限り、 `externalClusters` は、`pg_basebackup` メソッドまたは `recovery` メソッドのソース PostgreSQL クラスターを定義するために使用できます。外部クラスターには以下が必要です。

- 外部クラスターを識別する名前。 `source` オプションを介して参照として使用します
- 次の少なくとも 1 つ
- ストリーミング接続に関する情報 b_tran_3 **リカバリオブジェクトストア**に関する情報。これは、以下を含む Barman Cloud 互換オブジェクトストアです。
- WAL アーカイブポイントインタイムリカバリーに必要
- カタログ Postgres クラスターの物理ベースバックアップの

> **注釈**
>
> リカバリオブジェクトストアは、通常、Barman Cloud によって管理される AWS S3、Azure Blob Storage、または Google Cloud Storage ソースです。

ストリーミング接続のみが定義されている場合、ソースは `pg_basebackup` メソッドに使用できます。リカバリオブジェクトストアのみが定義されている場合、ソースは `recovery` メソッドに使用できます。両方が定義されている場合、2 つのブートストラップ方法のいずれかを選択できます。次の表は、オプションをまとめたものです。

Content of external-Clusters	pg_basebackup	recovery
Only streaming	✓	
Only object store		✓
Streaming and object store	✓	✓

さらに、 `pg_basebackup` または完全な `recovery` ポイントインタイムの場合、クラスターはレプリカクラスターモードの資格があります。これは、クラスターがストリーミングを介して、PostgreSQL の `restore_command` を介した WAL シッピングを介して、または 2 つのいずれかを介してソースから継続的にフィードされることを意味します。

詳細については、

パスワードファイル

`externalClusters` エントリー内でパスワードが指定されるたびに、CloudNativePG クラスターの EDB Postgres AI は PostgreSQL password file

その場合、各インスタンスの`/controller/external/NAME/pgpass` にあります。

このアプローチにより、CloudNativePG クラスターの EDB Postgres AI は、接続文字列でパスワードを公開せずに、外部サーバーとの接続を安全に確立できます。代わりに、接続は `passfile` 接続パラメーターを介して前述のファイルを安全に参照します。

5.1.3 空のクラスターをブートストラップする initdb

initdb ブートストラップ方法は、新しい PostgreSQL クラスターをスクラッチから作成するために使用されます。特に指定しない限り、デフォルトのものです。

次の例には、initdb 構成の完全な構造が含まれています。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-initdb
spec:
  instances: 3

  bootstrap:
    initdb:
      database: app
      owner: app
      secret:
        name: app-secret

  storage:
    size: 1Gi
```

上記のブートストラップの例では、次のことを行います。

1. PostgreSQL のネイティブ initdb コマンドを使用して、新しい PGDATA フォルダーを作成します
2. app という名前の_unprivileged_ユーザーを作成します
3. app-secret シークレットのパスワードを使用して、後者 app のパスワードを設定します username が owner の同じ名前と一致することを確認します
4. app ユーザーが所有する app という名前のデータベースを作成します。

_構成上の規則パラダイム_のおかげで、オペレーターにデフォルトのデータベース名 app とデフォルトのアプリケーションユーザー名データベース名と同じを選択させたり、スーパーユーザーとアプリケーションの両方の安全なパスワードをランダムに生成したりできます PostgreSQL のユーザー。

または、上記の例で説明したように、パスワードを生成し、シークレットとして保存し、PostgreSQL クラスターで使用できます。

提供されたシークレットは、kubernetes.io/basic-auth の仕様に準拠する必要があります。その結果、シークレットの username は、owner アプリケーションシークレットの場合およびスーパーユーザーの postgres のいずれかと一致する必要があります。

以下は、basic-auth シークレットの例です。

```
apiVersion: v1
data:
  username: YXBw
```

(次のページに続く)

(前のページからの続き)

```
  password: cGFzc3dvcmQ=
kind: Secret
metadata:
  name: app-secret
type: kubernetes.io/basic-auth
```

アプリケーションデータベースは、アプリケーションデータを保存するために使用するデータベースです。アプリケーションは、アプリケーションデータベースを所有するユーザーでクラスターに接続する必要があります。

データベース名を指定しない場合、オペレーターは慣例に従って app データベースを作成し、_defaulting Webhook_を使用してクラスター定義に追加します。データベースを所有するユーザーは、代わりにデータベース名をデフォルトにします。

アプリケーションユーザーは、オペレーターによって内部的に使用されず、代わりにスーパーユーザーに依存して、クラスターを目的のステータスに調整します。

initdb にオプションを渡す

PostgreSQL データディレクトリは、initdb を使用して初期化されます。

CloudNativePG クラスターの EDB Postgres AI を使用すると、`initdb` の動作をカスタマイズ、デフォルトのロケール構成やデータチェックサムなどの設定を変更できます。

> **⚠ 警告**
>
> PostgreSQL のロケールサポートの進行中の重要な機能強化により、 CloudNativePG クラスターの EDB Postgres AI は、ロケール関連オプションの `initdb` への直接プロキシとしてのみ機能します。 PostgreSQL ドキュメントに従って、正しいオプションが提供されていることを確認し、ブートストラッププロセスが正常に完了したことを確認するのはあなたの責任です。

`initdb` コマンドにカスタムオプションを含めるには、次のパラメーターを使用できます。

組み込みロケール

`builtinLocale` が値に設定されている場合、CloudNativePG クラスターの EDB Postgres AI は、それを `initdb` の`--builtin-locale` オプションに渡します。このオプションは、 Locale Support で定義された組み込みロケールを制御します

PostgreSQL ドキュメントからデフォルト空このオプションでは、`localeProvider` を `builtin` に設定する必要があることに注意してください。 PostgreSQL 17 から利用できます。

データチェックサム

`dataChecksums` が `true` に設定されている場合、CloudNativePG クラスターの EDB Postgres AI は、`initdb` の`-k` オプションを呼び出して、データページのチェックサムを有効にし、I/O システムによる破損の検出に役立ちます。それ以外の場合はサイレントデフォルト `false` 。

エンコーディング

encoding が値に設定されている場合、CloudNativePG クラスターの EDB Postgres AI は、それを initdb の--encoding オプションに渡します。これは、テンプレートデータベースデフォルト UTF8 のエンコードを選択します。

icu ロケール

icuLocale が値に設定されている場合、CloudNativePG クラスターの EDB Postgres AI は、それを initdb の--icu-locale オプションに渡します。このオプションは、Locale Support で定義された ICU ロケールを制御します

PostgreSQL ドキュメントからデフォルト空このオプションでは、localeProvider を icu に設定する必要があることに注意してください。 PostgreSQL 15 から利用できます。

icu ルール

icuRules が値に設定されている場合、CloudNativePG クラスターの EDB Postgres AI は、それを initdb の--icu-rules オプションに渡します。このオプションは、PostgreSQL ドキュメントデフォルト空の LocaleSupport で定義された ICU ロケールを制御します。このオプションでは、localeProvider を icu に設定する必要があることに注意してください。 PostgreSQL 16 から利用できます。

ロケール

locale が値に設定されている場合、CloudNativePG クラスターの EDB Postgres AI は、それを initdb の--locale オプションに渡します。このオプションは、PostgreSQL ドキュメントの Locale Support で定義されているロケールを制御します。デフォルトでは、locale パラメーターは空です。この場合、LANG などの環境変数はロケールを決定するために使用されます。これらの変数はコンテナイメージごとに異なる場合があり、一貫性のない動作が発生する可能性があることに注意してください。

ロケール Collate

localeCollate が値に設定されている場合、CloudNativePG クラスターの EDB Postgres AI は、それを initdb の--lc-collate オプションに渡します。このオプションは、Locale Support で定義されているように、照合順序 LC_COLLATE サブカテゴリを制御します

PostgreSQL ドキュメントデフォルト C より。

ロケール C タイプ

localeCType が値に設定されている場合、CloudNativePG クラスターの EDB Postgres AI は、それを initdb の--lc-ctype オプションに渡します。このオプションは、Locale Support で定義されているように、照合順序 LC_CTYPE サブカテゴリを制御します

PostgreSQL ドキュメントデフォルト C より。

ロケールプロバイダー

localeProvider が値に設定されている場合、CloudNativePG クラスターの EDB Postgres AI は、それを initdb の--locale-provider オプションに渡します。このオプションは、PostgreSQL ドキュメントの Locale Support デフォルト空、PostgreSQL の libc を意味しますの Locale Support で定義されたロケールプロバイダーを制御します。PostgreSQL 15 から利用できます。

walSegmentSize

walSegmentSize が値に設定されている場合、CloudNativePG クラスターの EDB Postgres AI は、それを initdb の--wal-segsize オプションに渡しますデフォルト未設定-16 メガバイトとして PostgreSQL で定

義されています。

> **注釈**
>
> *initdb* ブートストラップ中に CloudNativePG クラスターの EDB Postgres AI が実装する 2 つの唯一のロケールオプションは、 *LC_COLLATE* および `LC_TYPE` サブカテゴリーを参照します。残りのロケールサブカテゴリは、 *lc_messages* 、 *lc_monetary* 、 *lc_numeric* 、および `lc_time` パラメーターを使用して、PostgreSQL 構成で直接構成できます。

次の例は、データのチェックサムを有効にし、デフォルトのエンコードを `LATIN1` に設定します。

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-initdb
spec:
  instances: 3

  bootstrap:
    initdb:
      database: app
      owner: app
      dataChecksums: true
      encoding: "LATIN1"
  storage:
    size: 1Gi
```

> **警告**
>
> CloudNativePG クラスターの EDB Postgres AI は、 *options* サブセクションを使用して、*initdb* 呼び出しの動作をカスタマイズする別の方法をサポートしています。ただし、オペレーターの動作を壊す可能性のあるオプション `--auth` や `-d` などがあることを考えると、このテクニックは非推奨であり、API の将来のバージョンから削除されます。

初期化後のクエリの実行

クラスターが作成および構成された直後に、1 回実行されるクエリのカスタムリストを指定できます。これらのクエリーは、3 つの異なるデータベースに対して_superuser_ `postgres` として、次の特定の順序で実行されます。

1. `postgres` データベース `postInit` セクション
2. `template1` データベース `postInitTemplate` セクション
3. アプリケーションデータベース `postInitApplication` セクション

これらのセクションごとに、CloudNativePG クラスターの EDB Postgres AI は、次の順序で実行されるカスタムクエリを指定する 2 つの方法を提供します。

- クラスターの定義 `postInitSQL` 、`postInitTemplateSQL` 、および `postInitApplicationSQL` スタンザ内の SQL クエリーのリストとして
- シークレットおよび/または ConfigMap のリストとして、それぞれに実行される SQL スクリプト `postInitSQLRefs` 、`postInitTemplateSQLRefs` 、および `postInitApplicationSQLRefs` スタンザが含まれます。シークレットは ConfigMap の前に処理されます。

各リストのオブジェクトはシークエンシャルに処理されます。

警告

クエリーはスーパーユーザーとして実行され、クラスター全体を中断する可能性があるため、 *postInit* 、*postInitTemplate* 、および `postInitApplication` オプションを使用する際には細心の注意を払ってください。これらのクエリのエラーはブートストラップフェーズを中断し、クラスターが不完全なままとなり、手動介入が必要になります。

注釈

これらのクエリーは標準の `"$user", public` *search_path* で実行されますが、オペレーター発行の接続はセキュリティ上の理由から固定 `search_path` を固定します。 スキーマの解決と `search_path の強化 <スキーマの解決と `search_path` の強化>` を参照

詳細については。 `search_path` から独立する必要がある場合、オブジェクト参照をスキーマ修飾します。

次の例では、`postInitSQL` スタンザの一部として単一の SQL クエリーを実行します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-initdb
spec:
  instances: 3

  bootstrap:
    initdb:
      database: app
      owner: app
      dataChecksums: true
      localeCollate: "en_US"
      localeCType: "en_US"
      postInitSQL:
        - CREATE DATABASE angus
  storage:
    size: 1Gi
```

次の例では、postInitApplicationSQLRefs を使用して、シークレットと、アプリケーションデータベースの初期化後に実行するクエリーを含む ConfigMap を指定します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-initdb
spec:
  instances: 3

  bootstrap:
    initdb:
      database: app
      owner: app
      postInitApplicationSQLRefs:
        secretRefs:
          - name: my-secret
            key: secret.sql
        configMapRefs:
          - name: my-configmap
            key: configmap.sql
  storage:
    size: 1Gi
```

> **注釈**
>
> SQL スクリプト内では、各 SQL ステートメントは PostgreSQL semantics に従ってサーバーで 1 回の exec で実行されます。コメントを含めることはできますが、 *psql* のような内部コマンドは含めることができません。

互換性機能

EDB Postgres Advanced Server は、プレーンコミュニティ PostgreSQL に多くの互換性機能を追加します。これに関する詳細については、EDB Postgres Advanced Server を参照してください。

これらの機能は、EPAS でのクラスターの作成中に既に有効になっており、コミュニティ PostgreSQL イメージではサポートされていません。それらを無効にするには、次の例のように initdb セクションで redwood フラグを使用できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-initdb
spec:
  instances: 3
  imageName: <EPAS-based image>
```

(次のページに続く)

(前のページからの続き)

```
  bootstrap:
    initdb:
      database: app
      owner: app
      redwood: false
  storage:
    size: 1Gi
```

5.1.4 別のクラスターからのブートストラップ

CloudNativePG クラスターの EDB Postgres AI は、同じメジャーバージョンの別のクラスターから開始するクラスターのブートストラップを有効にします。この操作は、ストリーミングレプリケーション pg_basebackup を介してソースクラスターに直接接続するか、既存の物理的_basebackup_recovery を介して間接的に実行できます。

ソースクラスターは、 name で識別される externalClusters セクションで定義する必要があります。オリジンクラスターと同じ name を使用することをお勧めします。

デフォルトでは、外部クラスター定義の name の値に割り当てられます。

バックアップからのブートストラップ recovery

recovery の CloudNativePG クラスターオペレーターの EDB Postgres AI が提供するさまざまなバックアップ方法とバックアップストレージオプションの組み合わせを考慮して、各方法の詳細なガイダンスについては、専用の オブジェクトストアからのリカバリー を参照してください。

ライブクラスターからのブートストラップ pg_basebackup

pg_basebackup ブートストラップモードでは、有効な_streaming replication_接続を使用して、CloudNativePG クラスターの EDB Postgres AI によって管理される既存の **バイナリ互換** PostgreSQL インスタンス_source_ の正確な物理コピーとして、新しいクラスター_target_を作成できます。。ソースインスタンスは、プライマリまたはスタンバイの PostgreSQL サーバーのいずれかです。PostgreSQL の物理レプリケーションの長所と短所が完全に適用されるため、以下の要件セクションを完全に確認することが重要です。

この方法の主な使用例には次のものが含まれます。

- 定期的に (毎日、毎週) 再生成する必要があるレポートおよびビジネスインテリジェンスクラスター
- 定期的な再生毎日、毎週、毎月、匿名化が必要なライブデータを含むテストデータベース
- スタンドアロンレプリカクラスターの迅速なスピンアップ
- CloudNativePG Cluster クラスターの EDB Postgres AI の別の名前空間または Kubernetes クラスターへの物理的移行

> **警告**
>
> 現在の実装では、このメソッドはソース PostgreSQL インスタンスのクローンを作成し、それにより _snapshot_を作成します。クローン作成プロセスが完了すると、新しいクラスターがすぐに起動されます。詳細については、 **現在の制限** を参照してください。

`recovery` ブートストラップ方法と同様、クローン操作が完了すると、オペレーターは最初のインスタンスからターゲットクラスターの完全な所有権を取得します。これには、CloudNativePG クラスターの EDB Postgres AI の必要に応じて特定の構成パラメーターのオーバーライド、スーパーユーザーパスワードのリセット、`streaming_replica` ユーザーの作成、レプリカの管理などが含まれます。結果のクラスターは、ソースインスタンスから独立して動作します。

`pg_basebackup` によって透過的に管理されるターゲットインスタンス上のストリーミングレプリケーションクライアントは、次のいずれかの方法を使用してソースインスタンスで認証できます。

1. **ユーザー名/パスワードの認証**
2. *TLS* **証明書認証**

両方の認証方法について以下に詳しく説明します。

要件

`pg_basebackup` ブートストラップ方法には、次の要件が適用されます。

- ターゲットとソースは同じハードウェアアーキテクチャである必要があります
- ターゲットとソースは同じメジャー PostgreSQL バージョンである必要があります
- ターゲットとソースは同じテーブルスペースを持つ必要があります
- バックアップ用に少なくとも 1 つの_walsender_と WAL ストリーミング用に 1 つを提供することにより、この一回限りのオペレーションのターゲットからのアクセスを許可するために、十分な `max_wal_senders` でソースを構成する必要があります
- ターゲットインスタンスがソースインスタンスの PostgreSQL ポートに接続できるように、ソースとターゲット間のネットワークを構成する必要があります
- ソースには `REPLICATION LOGIN` 特権を持つロールが必要で、`pg_hba.conf` でこのロールのターゲットインスタンスからの接続を受け入れる必要があります、できれば TLS を介して下記の **レプリケーションユーザーについて** を参照してください
- ターゲットは、`REPLICATION LOGIN` 特権を持つロールを使用してソース PostgreSQL インスタンスに正常に接続できる必要があります

High Availability, Load Balancing, and Replication

PostgreSQL ドキュメント。

レプリケーションユーザーについて

要件セクションで説明したように、ソースインスタンスに SUPERUSER または、できれば REPLICATION 特権のみを持つユーザーが必要です。

ソースデータベースが CloudNativePG クラスターの EDB Postgres AI で作成されている場合、streaming_replica ユーザーを再利用し、クライアント TLS 証明書認証デフォルトで、streaming_replica に許可される唯一の接続方法を利用できます。

Kubernetes の外部を含む他のすべての場合、 REPLICATION 特権を持つユーザーが既に存在することを確認するか、以下の手順に従って新しいユーザーを作成してください。

ソースシステムの postgres ユーザーとして、次を実行してください。

```
createuser -P --replication streaming_replica
```

ターゲットインスタンスのシークレットに追加する必要があるため、プロンプトにパスワードを入力し、後で保存します。

> **注釈**
>
> 名前は重要ではありませんが、簡単にするために`streaming_replica` を使用します。以下のセクションの指示を調整する場合、必要に応じて変更してください。

ユーザー名/パスワードの認証

pg_basebackup ブートストラップを使用して CloudNativePG クラスターの EDB Postgres AI でサポートされている最初の認証方法は、ユーザー名とパスワードの一致に基づいています。

手順を開始する前に、次の情報があることを確認してください。

- ホスト名または IP アドレスと TCP ポートで識別されるソースインスタンスの場所
- レプリケーションユーザー名簡単にするために streaming_replica
- パスワード

次のような行をソース PostgreSQL インスタンスの pg_hba.conf ファイルに追加する必要がある場合があります。

```
#  A more restrictive rule for TLS and IP of origin is recommended

host replication streaming_replica all md5
```

次のマニフェストは、 pg_basebackup ブートストラップ方法を使用して、source-db externalClusters 配列内で定義された外部 PostgreSQL クラスターのクローンを作成します。ご覧のとおり、source-db 定義は source-db.foo.com ホストをポイントし、streaming_replica ユーザーとして接続します。そのパスワードは source-db-replica-user シークレットの password キーに保存されています。

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: target-db
spec:
  instances: 3
  imageName: docker.enterprisedb.com/k8s/postgresql:18.4-standard-ubi9

  bootstrap:
    pg_basebackup:
      source: source-db

  storage:
    size: 1Gi

  externalClusters:
    - name: source-db
      connectionParameters:
        host: source-db.foo.com
        user: streaming_replica
      password:
        name: source-db-replica-user
        key: password
```

クローン操作が機能するには、同じ PostgreSQL バージョンこの場合は 18.4 を含むすべての要件を満たす必要があります。

TLS 証明書認証

pg_basebackup ブートストラップを使用して CloudNativePG クラスターの EDB Postgres AI でサポートされている 2 番目の認証方法は、TLS クライアント証明書に基づいています。これは、セキュリティの観点から推奨されるアプローチです。

次の例では、同じ Kubernetes クラスター内の既存の PostgreSQL クラスター `cluster-example` のクローンを作成します。

> **注釈**
>
> この例は、Kubernetes クラスターの外部にあるインスタンスをカバーするように簡単に適合できます。

マニフェストは、 `cluster-clone-tls` と呼ばれる新しい PostgreSQL 18.4 クラスターを定義します。これは、 `cluster-example` 外部クラスターから pg_basebackup メソッドを使用してブートストラップされます。ホストは同じクラスター内の読み取り/書き込みサービスによって識別され、`streaming_replica` ユーザーは、提供されたキー、証明書、および認証局情報それぞれ `cluster-example-replication` および `cluster-example-ca` シークレットのおかげで認証されます。

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-clone-tls
spec:
  instances: 3
  imageName: docker.enterprisedb.com/k8s/postgresql:18.4-standard-ubi9

  bootstrap:
    pg_basebackup:
      source: cluster-example

  storage:
    size: 1Gi

  externalClusters:
    - name: cluster-example
      connectionParameters:
        host: cluster-example-rw.default.svc
        user: streaming_replica
        sslmode: verify-full
      sslKey:
        name: cluster-example-replication
        key: tls.key
      sslCert:
        name: cluster-example-replication
        key: tls.crt
      sslRootCert:
        name: cluster-example-ca
        key: ca.crt
```

アプリケーションデータベースの構成

また、initdb および recovery ブートストラップ方法の場合と同様に、ライブクラスターからブートストラップするクラスターのアプリケーションデータベースの構成もサポートしています。新しいクラスターがレプリカクラスターとしてレプリカモードが有効化されて作成される場合、アプリケーションデータベースの構成はスキップされます。

次の例では、ライブクラスターからブートストラップに従って、所有者 app と提供されたシークレット app-secret に保存されたパスワードを使用して app データベースを構成します。

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
[...]
spec:
  bootstrap:
```

(次のページに続く)

(前のページからの続き)

```
  pg_basebackup:
    database: app
    owner: app
    secret:
      name: app-secret
    source: cluster-example
```

上記の構成では、次は **復旧が完了した後** にのみ発生します。

1. app データベースが存在しない場合、作成されます。
2. app ユーザーが存在しない場合、作成されます。
3. app ユーザーが app データベースの所有者でない場合、所有権は app ユーザーに付与されます。
4. username 値がシークレットの owner 値と一致する場合、アプリケーションユーザーこの場合は app ユーザーのパスワードはシークレットの password 値に更新されます。

現在の制限

スナップショットコピー

pg_basebackup メソッドは、PostgreSQL ベースバックアップの形式でソースインスタンスのスナップショットを取得します。バックアップの開始からバックアップの適切な終了までに書き込まれるすべてのトランザクションは、2 番目の接続を使用してターゲットインスタンスにストリーミングされます pg_basebackup の--wal-method=stream オプションを参照してください。

バックアップが完了すると、新しいインスタンスが新しいタイムラインで起動され、ソースから分岐します。このため、ターゲットデータベースに移行する前に、ソースデータベースへのすべての書き込み操作を停止することをお勧めします。

この制限は、ターゲットクラスターがレプリカクラスターとして定義されていない場合にのみ適用されることに注意してください。

5.2 Image Catalog

ImageCatalog および ClusterImageCatalog は、PostgreSQL イメージのライフサイクルを Cluster 定義から分離できるカスタムリソース定義 CRD です。カタログを使用することにより、イメージの更新を一元管理できます。カタログエントリが更新されると、関連するすべてのクラスターが自動的に *roll out the new image* 。

カスタムカタログをビルドできますが、 CloudNativePG クラスターの EDB Postgres AI は、ClusterImageCatalog リソースとして *EDB CloudNativePG* **クラスターカタログ** を提供し、すべての公式コミュニティ PostgreSQL コンテナイメージをカバーします。

5.2.1 カタログのスコーピング

2 つのリソースの主な違いは、そのスコープです。

Resource	Scope	Best use case
ImageCatalog	Namespaced	Application-specific versions or team-level restrictions.
ClusterImageCatalog	Cluster-wide	Global standards across all namespaces for an organization.

5.2.2 カタログ構造

両方のリソースは、共通のスキーマを共有します。

- **メジャーバージョン** `major` PostgreSQL バージョンによってキー付けられたイメージのリスト。
- **一意性** `major` フィールドは、単一のカタログ内で一意である必要があります。
- **拡張機能**認定された拡張機能コンテナイメージのサポート `extension_control_path` 経由で PostgreSQL 18+で利用可能。
- **コンポーネントイメージ** PgBouncer などの非 PostgreSQL コンポーネントの名前付けイメージのオプションのリスト Component images を参照してください。

> **警告**
>
> オペレーターは画像検出を実行せずにユーザー定義の`major` バージョンを信頼しますが、CloudNativePG クラスターカタログの公式 EDB Postgres AI はコミュニティによって事前検証され、すべての拡張機能とオペランドイメージエントリが宣言されたメジャーバージョンと正しく一致することを確認します。カスタムカタログを作成する場合は、宣言されたメジャーバージョンが実際の PostgreSQL イメージと一致することを確認して、互換性を維持する必要があります。

PostgreSQL イメージに加えて、カタログは、 `componentImages` フィールドを介して他のコンポーネントのイメージを保存できます。各エントリは文字列 **key** によって識別されます。カタログ内で一意である必要がある小文字の英数字識別子ハイフン使用可、1〜63 文字。

現在、コンポーネントイメージの唯一のコンシューマは `Pooler` リソースであり、コンポーネントイメージエントリを参照して PgBouncer コンテナイメージを一元管理できます**イメージカタログを使用する** を参照してください。

次の例では、PgBouncer コンポーネントイメージを使用して名前空間 `ImageCatalog` を定義します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: ImageCatalog
metadata:
  name: my-catalog
  namespace: default
spec:
```

(次のページに続く)

(前のページからの続き)

```
  images:
    - major: 18
      image: docker.enterprisedb.com/k8s/postgresql:18.4-standard-ubi9
  componentImages:
    - key: pgbouncer
      image: docker.enterprisedb.com/k8s/pgbouncer:1.25.1-ubi9
```

クラスター全体のカタログの場合、kind: ClusterImageCatalog を使用し、metadata.namespace フィールドをドロップします spec はそれ以外の場合は同じです。

> **注釈**
>
> カタログには、最大 32 のコンポーネント画像エントリを含めることができます。キーは、小文字の英数字またはハイフンであり、英数字で始まり 63 文字までです。

カタログの定義

単一のカタログ内で複数のメジャーバージョンを定義できます。

次の例では、名前空間 ImageCatalog を定義します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: ImageCatalog
metadata:
  name: postgresql
  namespace: default
spec:
  images:
    - major: 15
      image: docker.enterprisedb.com/k8s/postgresql:15.14-standard-ubi9
    - major: 16
      image: docker.enterprisedb.com/k8s/postgresql:16.10-standard-ubi9
    - major: 17
      image: docker.enterprisedb.com/k8s/postgresql:17.6-standard-ubi9
    - major: 18
      image: docker.enterprisedb.com/k8s/postgresql:18.4-standard-ubi9
```

次の例は、クラスター全体の ClusterImageCatalog を定義します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: ClusterImageCatalog
metadata:
  name: postgresql-global
spec:
  images:
```

(次のページに続く)

(前のページからの続き)

```
    - major: 15
      image: docker.enterprisedb.com/k8s/postgresql:15.14-standard-ubi9
    - major: 16
      image: docker.enterprisedb.com/k8s/postgresql:16.10-standard-ubi9
    - major: 17
      image: docker.enterprisedb.com/k8s/postgresql:17.6-standard-ubi9
    - major: 18
      image: docker.enterprisedb.com/k8s/postgresql:18.4-standard-ubi9
```

クラスター内のカタログの参照

Cluster リソースは、 imageCatalogRef を使用して画像を選択します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3
  imageCatalogRef:
    apiGroup: postgresql.k8s.enterprisedb.io
    kind: ClusterImageCatalog # Or ImageCatalog
    name: postgresql-global
    major: 18
  storage:
    size: 1Gi
```

5.2.3 イメージボリューム拡張を使用したイメージカタログ

イメージボリューム拡張を使用したイメージカタログ では、カタログエントリー内で直接拡張機能のコンテナをバンドルできます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: ImageCatalog
metadata:
  name: postgresql
spec:
  images:
    - major: 18
      image: docker.enterprisedb.com/k8s/postgresql:18.4-minimal-ubi9
      extensions:
        - name: foo
          image:
            reference: # registry path for your `foo` extension image
```

extensions セクションは *ExtensionConfiguration* の後です

API スキーマと構造。イメージカタログを参照するクラスターは、関連する拡張機能を名前でロードできます。

> **注釈**
>
> *documentation of image volume extensions* を参照してください。

内部イメージ構造、構成オプション、およびクラスター内のカタログ拡張機能を選択またはオーバーライドする方法の詳細については、

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: ImageCatalog
metadata:
  name: postgresql
spec:
  images:
    - major: 18
      image: docker.enterprisedb.com/k8s_enterprise/postgresql:18.3-minimal-ubi9
      extensions:
        - name: foo
          image:
            reference: # registry path for your `foo` extension image
```

extensions セクションは *ExtensionConfiguration* の後にあります

API スキーマと構造。イメージカタログを参照するクラスターは、関連する拡張機能を名前でロードできます。

> **注釈**
>
> *documentation of image volume extensions* を参照してください

内部イメージ構造、構成オプション、およびクラスター内のカタログ拡張機能を選択またはオーバーライドする方法の詳細については、

5.2.4 EDB CloudNativePG クラスターカタログ

CloudNativePG Cluster プロジェクトの EDB Postgres AI は、サポートされているすべてのイメージの ClusterImageCatalog マニフェストを維持します。

これらのカタログは定期的に更新され、artifacts repository 内の 2 つの異なる場所で公開されます。

- ** image-catalogs :** 基本イメージタイプのコアカタログ定義。
- ** image-catalogs-extensions :** 上記のカタログと同じですが、 minimal イメージタイプに拡張定義が含まれるという主な違い。

各カタログは、イメージタイプと Debian リリースの特定の組み合わせに対応します `trixie` など。サポートされているすべての PostgreSQL メジャーバージョンの最新のコンテナイメージをリストしています。

> **重要**
>
> 最大限のセキュリティと不変性を保証するために、CloudNativePG クラスターカタログの公式 EDB Postgres AI カタログ内のすべての画像は、タグではなく**SHA256 ダイジェスト**によって識別されます。

コアカタログは古いバージョンのオペレーターで動作しますが、**``extensions`` セクションを含むカタログは、CloudNativePG クラスター 1.29 以降の EDB Postgres AI とのみ互換性があります**。古いオペレーターの拡張定義を含むカタログを使用すると、これらの定義が拒否されます。

インストールと使用

これらのカタログをインストールすることにより、クラスター管理者は、PostgreSQL クラスターが、選択した Debian ディストリビューションとイメージタイプの特定の PostgreSQL メジャーバージョン内の最新のパッチリリースに自動的に更新されることを確認できます。

たとえば、`minimal` PostgreSQL コンテナイメージの最新のカタログを Debian `trixie` にインストールするには、次のコマンドを実行します。

```
kubectl apply -f \
  https://raw.githubusercontent.com/cloudnative-pg/artifacts/refs/heads/main/image-
↪catalogs/catalog-minimal-trixie.yaml
```

`image-catalogs` ディレクトリにある `kustomization` ファイルを使用して、使用可能なすべてのカタログをインストールできます。

```
kubectl apply -k https://github.com/cloudnative-pg/artifacts//image-catalogs?ref=main
```

次に、次の方法で展開されたすべてのカタログを表示できます。

```
kubectl get clusterimagecatalogs.postgresql.k8s.enterprisedb.io
```

例 クラスターでのカタログの使用

`trixie` 上の PostgreSQL 18 の最新の `minimal` イメージを常に追跡するクラスターを作成するには、次のように `Cluster` を定義します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: angus
spec:
  instances: 3
```

(次のページに続く)

(前のページからの続き)

```
  imageCatalogRef:
    apiGroup: postgresql.k8s.enterprisedb.io
    kind: ClusterImageCatalog
    name: postgresql-minimal-trixie
    major: 18
  storage:
    size: 1Gi
```

5.3 Importing Postgres databases

このセクションでは、CloudNativePG Cluster クラスターの新しい EDB Postgres AI に 1 つ以上の既存の PostgreSQL データベースをインポートする方法について説明します。

インポート操作は、PostgreSQL のオンライン論理バックアップの概念に基づいており、オリジンホストへのネットワーク接続を介して `pg_dump` 、および `pg_restore` に依存します。ネイティブのマルチバージョン同時実行制御 MVCC とスナップショットのおかげで、PostgreSQL では、書き込みアクティビティを停止せずに、ネットワークを介して同時実行の方法で、一貫したバックアップを取得できます。

論理バックアップは、PostgreSQL バージョンのメジャーアップグレードを実行するための最も一般的な柔軟で信頼性の高い技術でもあります。

その結果、このセクションの手順は両方の場合に適しています。

- Kubernetes の外部でも、既存の PostgreSQL インスタンスから 1 つ以上のデータベースをインポート
- PostgreSQL バージョンから同じまたは新しいバージョンにデータベースをインポートし、PostgreSQL の_メジャーアップグレード_を有効にしますバージョン 14.x からバージョン 18.x など

> **警告**
>
> PostgreSQL のメジャーアップグレードを実行するときは、アプリケーションが新しいバージョンと互換性があり、データベース拡張機能を含むオブジェクトのアップグレードパスが実行可能であることを確認する責任があります。

どちらの場合も、操作は元のデータベースの一貫した **スナップショット**で実行されます。

5.3.1 その仕組み

概念的には、インポートには、*cluster-example-initdb-sql-refs.yaml* を使用して新しいクラスター_destination cluster_を作成し、`initdb.import` サブセクションを完了して、既存の Postgres クラスター_source cluster_からオブジェクトをインポートする必要があります。PostgreSQL の推奨に従って、_destination cluster_の PostgreSQL メジャーバージョンが_source cluster_のメジャーバージョン以上であることをお勧めします。

CloudNativePG クラスターの EDB Postgres AI は、ソースクラスターから宛先クラスターにオブジェクトをインポートする 2 つの主な方法を提供します。

- **マイクロサービスアプローチ**宛先クラスターは、CloudNativePG クラスタープロジェクトの EDB Postgres AI が推奨しているように、指定されたアプリケーションユーザーが所有する単一のアプリケーションデータベースをホストするように設計されています
- **モノリスアプローチ**宛先クラスターは、ソースクラスターからインポートされた複数のデータベースとさまざまなユーザーをホストするように設計されています。

最初のインポート方法は `microservice` タイプを介して使用でき、2 番目は `monolith` タイプを介して使用できます。

> **警告**
>
> 宛先クラスターが、スーパーユーザーまたは`pg_dump` で論理バックアップを取るための十分な特権を持つユーザーでソースクラスターにアクセスできることを確認するのはあなたの責任です。 *PostgreSQL documentation on `pg_dump* <https://www.postgresql.org/docs/current/app-pgdump.html>`_ を参照してください。

詳細については、

5.3.2 `microservice` タイプ

マイクロサービスアプローチでは、ソースクラスターから宛先クラスターにインポートする単一のデータベースを指定できます。操作は 4 つのステップで実行されます。

- 新しいクラスターの `initdb` ブートストラップ
- `pg_dump -Fd` を使用して、選択したデータベース initdb.import.databases`内をエクスポートする

`

- `initdb.owner` 使用者が所有する `initdb.database` アプリケーションデータベースへの `pg_restore --no-acl --no-owner` を使用したデータベースのインポート

r

- データベースダンプファイルのクリーンアップ
- `postImportApplicationSQL` パラメーターを介してアプリケーションデータベースでのユーザー定義の SQL クエリーのオプションの実行
- インポートされたデータベースでの `ANALYZE VERBOSE` の実行

次の図では、_N_データベースを含む単一の PostgreSQL クラスターが、CloudNativePG クラスタークラスターの個別の EDB Postgres AI にインポートされ、各クラスターは_N_ソースデータベースのいずれかに対するマイクロサービスインポートを使用します。

たとえば、以下の YAML は、 `postgres` に接続することにより、`cluster-pg96` クラスターサポートされていない PostgreSQL 9.6 から `angus` データベースをインポートする `cluster-microservice` と呼ばれる新しい 3 インスタンスの PostgreSQL クラスターオペレーターがリリースされた時点での利用可能な最新のメジャーバージョンを作成します `cluster-pg96-superuser` シークレットに保存されたパスワードを介して、 `postgres` ユーザーを使用してデータベースにアクセスします。

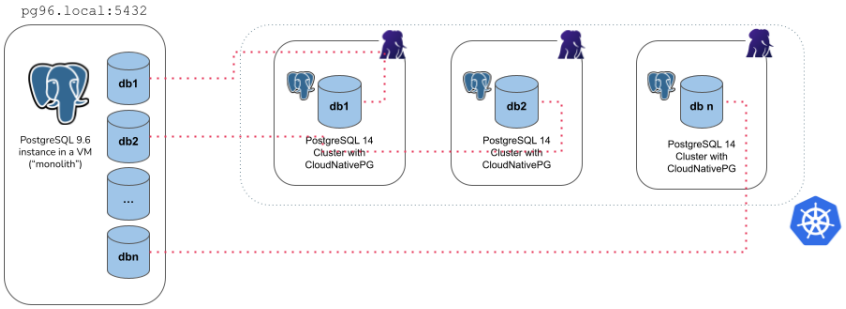


図 1: Example of microservice import type

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-microservice
spec:
  instances: 3

  bootstrap:
    initdb:
      import:
        type: microservice
        databases:
          - angus
        source:
          externalCluster: cluster-pg96
        #postImportApplicationSQL:
        #- |
        #  INSERT YOUR SQL QUERIES HERE
  storage:
    size: 1Gi
  externalClusters:
    - name: cluster-pg96
      connectionParameters:
        # Use the correct IP or host name for the source database
        host: pg96.local
        user: postgres
        dbname: postgres
      password:
        name: cluster-pg96-superuser
        key: password
```

警告

上記の例では、コミュニティでサポートされていない PostgreSQL のバージョンを実行しているソース

データベースを意図的に使用しています。したがって、CloudNativePG クラスターの EDB Postgres AI ではサポートされていません。ソースインスタンスからのデータのエクスポートは、宛先クラスターの `pg_dump` のバージョンを使用して実行されます。これは、サポートされているものであり、ソースのものと同等以上である必要があります。私たちの経験に基づいて、データをエクスポートするこの方法は Postgres の古いバージョンのサポートされていないバージョンでも動作するため、古いデータを Kubernetes 内のより良いシステムに移動する機会が得られます。これが、このセクションの例で 9.6 を使用した主な理由です。この領域で問題が発生した場合は、ご連絡をお待ちしております。

`microservice` タイプを使用するときは、いくつか注意する必要があります。

- インポートするデータを含む既存の PostgreSQL インスタンスをポイントする `externalCluster` が必要です。詳細については、 :ref:``externalClusters` セクション <*externalClusters* セクション>` を参照してください。
- オペレーション中に Kubernetes クラスターと `externalCluster` の間でトラフィックを許可する必要がある
- ソースデータベースへの接続は、`pg_dump` を実行し、ロール情報を読み取る必要がある指定されたユーザーで付与される必要があります_superuser_は OK です
- 現在、`pg_dump -Fd` 結果は `PGDATA` ボリュームの `dumps` フォルダー内に一時的に保存されます。そのため、割り当てられたノードにダンプ結果、復元されたデータとインデックスを一時的に含むための十分な利用可能な領域が必要です。インポート操作が完了すると、このフォルダーはオペレーターによって自動的に削除されます。
- `initdb.import.databases` 配列内で指定できるデータベースは 1 つだけです
- ロールはインポートされません - そのため、`initdb.import.roles` 内で指定できません

5.3.3 `monolith` タイプ

モノリスアプローチでは、ソースクラスターから宛先クラスターにインポートするロールとデータベースのセットを指定できます。操作は次の手順で実行されます。

- 新しいクラスターの `initdb` ブートストラップ
- 選択したロールのエクスポートとインポート

s

- `pg_dump -Fd` を使用して、選択したデータベース initdb.import.databases`内を 1 つずつエクスポート

`

- 選択した各データベースを作成し、 `pg_restore` を使用してデータをインポートします
- インポートされた各データベースで `ANALYZE` を実行します
- データベースダンプファイルのクリーンアップ

たとえば、以下の YAML は、 `cluster-monolith` と呼ばれる新しい 3 インスタンスの PostgreSQL クラスターオペレーターがリリースされた時点での利用可能な最新のメジャーバージョンを作成します。`postgres`

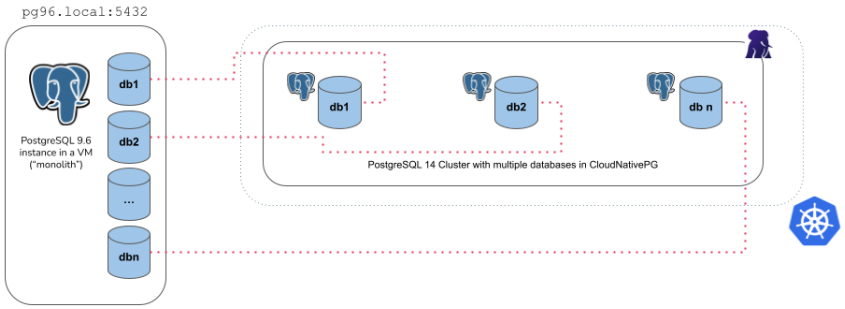


図 2: Example of monolith import type

ユーザーを使用して postgres データベースに接続し、cluster-pg96-superuser シークレットに保存されているパスワードを介して postgres データベースに接続します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-monolith
spec:
  instances: 3
  bootstrap:
    initdb:
      import:
        type: monolith
        databases:
          - accounting
          - banking
          - resort
        roles:
          - accountant
          - bank_user
        source:
          externalCluster: cluster-pg96
  storage:
    size: 1Gi
  externalClusters:
    - name: cluster-pg96
      connectionParameters:
        # Use the correct IP or host name for the source database
        host: pg96.local
        user: postgres
        dbname: postgres
        sslmode: require
      password:
        name: cluster-pg96-superuser
        key: password
```

monolith タイプを使用するときは、いくつか注意する必要があります。

- インポートするデータを含む既存の PostgreSQL インスタンスをポイントする externalCluster が必要です。詳細については、 :ref:``externalClusters` セクション *<externalClusters セクション>*` を参照してください。
- オペレーション中に Kubernetes クラスターと externalCluster の間でトラフィックを許可する必要がある
- ソースデータベースへの接続は、pg_dump を実行し、ロール情報を取得する必要がある指定されたユーザーで付与される必要があります_superuser_は OK です
- 現在、pg_dump -Fd 結果は、宛先クラスターのインスタンスの PGDATA ボリュームの dumps フォルダー内に一時的に保存されます。そのため、割り当てられたノードにダンプ結果、復元されたデータとインデックスを一時的に含むための十分な利用可能な領域が必要です。インポート操作が完了すると、このフォルダーはオペレーターによって自動的に削除されます。
- initdb.import.databases 配列で指定される少なくとも 1 つのデータベース
- インポートされたデータベースに必要なロールは、 initdb.import.roles 内で指定する必要があります。以下の制限があります。
- 次のロールが存在する場合、インポートされません。postgres 、streaming_replica 、cnp_pooler_pgbouncer
- SUPERUSER オプションは、インポートされたロールから削除されます
- ワイルドカード"*" は、databases および/または roles 配列の唯一の要素として使用して、その種類のすべてのオブジェクトをインポートできます。データベースを照合する場合、ワイルドカードは postgres データベース、テンプレートデータベース、および接続を許可しないデータベースを無視します
- クローンプロシージャが完了した後、すべてのデータベースに対して ANALYZE VERBOSE が実行されます。
- postImportApplicationSQL フィールドはサポートされていません

5.3.4 実際の例

monolith アプローチを使用して単一のデータベースをインポートすることを妨げるものはありません。そうした場合の結果が microservice アプローチを使用した場合とどのように異なるかを見るのは興味深いです。

ロール me が所有する mydb という名前のデータベースを使用するソースクラスター、たとえば、

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 1
```

(次のページに続く)

(前のページからの続き)

```
  postgresql:
    pg_hba:
      - host all all all trust

  storage:
    size: 1Gi

  bootstrap:
    initdb:
      database: mydb
      owner: me
```

microservice を介してインポートできます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-microservice
spec:
  instances: 1

  storage:
    size: 1Gi

  bootstrap:
    initdb:
      import:
        type: microservice
        databases:
          - mydb
        source:
          externalCluster: cluster-example

  externalClusters:
    - name: cluster-example
      connectionParameters:
        host: cluster-example-rw
        dbname: postgres
```

モノリス経由と同様に

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-monolith
```

(次のページに続く)

(前のページからの続き)

```
spec:
  instances: 1

  storage:
    size: 1Gi

  bootstrap:
    initdb:
      import:
        type: monolith
        databases:
          - mydb
        roles:
          - me
        source:
          externalCluster: cluster-example

  externalClusters:
    - name: cluster-example
      connectionParameters:
        host: cluster-example-rw
        dbname: postgres
```

どちらの場合も、データベースの内容はインポートされますが、

- マイクロサービスの場合、インポートされたデータベースの名前と所有者は両方とも app 、またはフィールドの構成のいずれかになります database および owner が bootstrap.initdb スタンザで設定されています。
- モノリスの場合、データベースと所有者はソースクラスター、つまりそれぞれ mydb および me とまったく同じように保持されます。 app データベースもユーザーも作成されません。インポートするソースデータベース/所有者と一致しない bootstrap.initdb スタンザの database および owner のカスタム設定がある場合、インスタンスマネージャーは新しい空のアプリケーションデータベースと所有者ロールを作成しますが、インポートされたデータベース/所有者はそのまま残します。

5.3.5 インポートの最適化

データベースの論理インポート中に、CloudNativePG クラスターの EDB Postgres AI は、以下を強制することにより、速度とデータの耐久性を優先するために PostgreSQL の構成を最適化します。

- archive_mode 〜off
- fsync 〜off
- full_page_writes 〜off
- max_wal_senders 〜0
- wal_level 〜minimal

インポートジョブを完了する前に、CloudNativePG クラスターの EDB Postgres AI は、予想される構成を復元し、`initdb --sync-only` を実行して、データがディスクに永続的に書き込まれるようにします。

インポートフェーズ中に実行できる最適化は他にもあります。このトピックは CloudNativePG クラスターの EDB Postgres AI の範囲を超えていますが、 `Cluster` 構成で直接 `shared_buffers` 、`max_wal_size` 、`checkpoint_timeout` などの Postgres GUC を調整することにより、チェックポイント領域への不要な書き込みを削減することをお勧めします。

5.3.6 pg_dump および pg_restore 動作のカスタマイズ

`pgDumpExtraOptions` および `pgRestoreExtraOptions` パラメーターを使用して追加オプションを指定することにより、`pg_dump` および `pg_restore` の動作をカスタマイズできます。これは、パフォーマンスの向上またはインポート/エクスポートの複雑さの管理に特に役立ちます。

たとえば、並列ジョブを有効にすると、データ転送が大幅に高速化されます。

```
bootstrap:
  initdb:
    import:
      type: microservice
      databases:
        - app
      source:
        externalCluster: cluster-example
      pgDumpExtraOptions:
        - "--jobs=2"
      pgRestoreExtraOptions:
        - "--jobs=2"
```

ステージ固有の pg_restore オプション

インポートプロセスをより詳細に制御するために、CloudNativePG クラスターの EDB Postgres AI は、次のフェーズでステージ固有の `pg_restore` オプションをサポートしています。

- `pre-data` – 例、スキーマ定義
- `data` – 例、テーブルの内容
- `post-data` – 例、インデックス、コンストレイン、トリガー

フェーズごとにオプションを指定することにより、並列処理を最適化し、復元するオブジェクトの性質に合わせて調整したフラグを適用できます。

```
bootstrap:
  initdb:
    import:
      type: microservice
      schemaOnly: false
      databases:
```

(次のページに続く)

(前のページからの続き)

```
    - mynewdb
  source:
    externalCluster: sourcedb-external
  pgRestorePredataOptions:
    - "--jobs=1"
  pgRestoreDataOptions:
    - "--jobs=4"
  pgRestorePostdataOptions:
    - "--jobs=2"
```

上記の例では

- `--jobs=1` は `pre-data` ステージに適用されて、スキーマ作成の順序を保持します。
- `--jobs=4` は、`data` ステージでの並列処理を増加させ、大規模なデータのインポートを高速化します。
- `--jobs=2` は、 `post-data` ステージでのパフォーマンスと依存関係の処理のバランスをとります。

これらのステージ固有の設定は、同時実行の調整によってパフォーマンスが大幅に向上する可能性がある大規模なデータベースまたはリソース依存度の環境で特に役立ちます。

注釈

提供される場合、ステージ固有のオプションは一般的な`pgRestoreExtraOptions` より優先されます。

警告

pgDumpExtraOptions 、*pgRestoreExtraOptions* 、およびすべてのステージ固有のリストアオプション `pgRestorePredataOptions` 、*pgRestoreDataOptions* 、*pgRestorePostdataOptions* は、オペレーターによる検証なしで基になる PostgreSQL ツールに直接渡されます。特定のフラグは、オペレーターの意図した機能またはデザインと競合する場合があります。これらのオプションは注意して使用し、運用に適用する前に、安全で制御された環境で常に徹底的にテストします。

5.3.7 オンラインインポートとアップグレード

論理レプリケーションは、次のアプローチを使用して、ネットワーク経由でアクセス可能な PostgreSQL データベースをインポートする強力な方法を提供します。

- **スキーマのみのオプションを使用したブートストラップのインポート**レプリケーションを開始する前に、ターゲットデータベースのスキーマを初期化します。
- **``Subscription`` リソース**連続レプリケーションを設定してデータ変更を同期します。

この手法は、最小限のダウンタイムでメジャー PostgreSQL のアップグレードを実行するために活用できるため、シームレスな移行とシステムのアップグレードに最適です。

制限やベストプラクティスを含む詳細については、ドキュメントの **宣言的なパブリケーションおよびサブスクリプションオブジェクトを介した論理レプリケーション** セクションを参照してください。

5.4 Security

CloudNativePG クラスターの EDB Postgres AI は、スタックのすべてのレイヤーにわたって「最小特権」の原則に従って、デフォルトで安全になるように設計されています。このドキュメントでは、セキュリティガバナンス、サプライチェーンの完全性、およびコード、コンテナ、クラスターの 3 つの異なるレイヤーで分析された技術的制御の概要を説明します。

> **警告**
>
> このページに含まれる情報は、Kubernetes クラスターで通常の InfoSec 業務を実行することから免責されてはなりません。

Overview of Cloud Native Security についてよく理解してください

Kubernetes ドキュメントのページ。

The 4C’ s Security Model in Kubernetes を参照してください。

EDB が CloudNativePG クラスターの EDB Postgres AI のセキュリティで採用したアプローチのより良い理解とコンテキストを取得するブログ記事。

5.4.1 コード

CloudNativePG Cluster のソースコードの EDB Postgres AI は、CI / CD パイプラインに直接統合されている Go の人気のオープンソースリンター GolangCI-Lint を使用して、セキュリティ脆弱性のチェックを含む体系的な静的分析を受けます。GolangCI-Lint は、同じソースコードで複数のリンターを実行できます。

次のツールは、セキュリティ問題を特定するために使用されます。

- ** Golang Security Checker `gosec`)** ハードコードされた資格情報、整数オーバーフロー、SQL などの既知の脆弱性、脅威、弱点を検出するように設計された一連のルールに対してソースコードの抽象構文ツリーをスキャンするリンター注射。GolangCI-Lint は、スイートの一部として `gosec` を実行します。
- ** govulncheck :** このツールは CI / CD パイプラインで実行され、Go コードまたはコンパイラーに影響を与える既知の脆弱性を報告します。既知の脆弱性を含む Go コンパイラーのバージョンでオペレーターがビルドされている場合、`govulncheck` はそれを検出します。
- ** CodeQL :** GitHub が提供するこのツールは、セキュリティ問題をスキャンし、検出された脆弱性を含むプルリクエストをブロックします。CodeQL は、Python や Bash などのリポジトリ内の他の言語を除き、Go コードのみをレビューするように構成されています。
- ** Snyk :** スケジュールされたジョブで毎晩コードスキャンを実行し、コードのセキュリティとライセンスの問題に関連する新しい結果を強調する週次レポートを生成します。

CloudNativePG クラスターリポジトリの EDB Postgres AI は、

Security section で_“Private vulnerability Reporting”_オプションが有効になっています。この機能を使用すると、ユーザーは一般に公開される前に、注意深い取り扱いが必要なセキュリティ問題を安全に報告できます。セキュリティバグを発見した場合は、この媒体を使用して報告してください。

重要

CI / CD パイプラインの静的コード分析フェーズの障害は、CloudNativePG クラスターの EDB Postgres AI の配信プロセス全体をブロックします。すべてのコミットは、GolangCI-Lint で定義されたすべてのリンターを渡す必要があります。

5.4.2 コンテナ

CloudNativePG クラスターの EDB Postgres AI のオペレーターイメージは、リリースプロセスの一部としてビルドおよび *published* されます。使用可能な最新のオペレーターバージョンについては、*EDB CloudNativePG Cluster 1.26.0 release notes* を参照してください。

オペランドイメージは、サポートされている PostgreSQL バージョンごとに EDB Postgres Extended および EDB Postgres Advanced を含むで毎月ビルドおよび公開されます。

画像は次のツールを使用してスキャンされます。

- ** Dockle :** コンテナビルドプロセスのベストプラクティスを保証します。
- ** Black Duck :** オープンソースコンポーネントの脆弱性をチェックし、ライセンスの一貫性を検証します。

重要

すべてのオペランドイメージは、パイプラインによって毎月自動的に再構築され、ベースイメージとパッケージレベルの両方で最新のセキュリティ更新が組み込まれ、EDB ダウンロードサイトに配布されるコンテナイメージの**パッチレベルの更新**を提供します。

警告

古いイメージを実行すると、環境がセキュリティリスクとパフォーマンスの問題にさらされる可能性があります。最新のイメージバージョンに更新し、イメージを最新の状態に保つことを強くお勧めします。これにより、利用可能な最新の更新とパッチを確実に利用できます。

コンテナセキュリティのガイドラインとフレームワーク

コンテナレベルのセキュリティを確保するために、次のガイドラインとフレームワークが検討されています。

- ** Container Image Creation and Deployment Guide :** 米国国防総省 DoD の防衛情報システム局 DISA によって開発されました。
- **

CIS Benchmark for Docker :
: インターネットセキュリティセンター CIS によって開発されました。

> **参考**
>
> CloudNativePG クラスターの EDB Postgres AI のコンテナレベルのセキュリティに関して EDB が採用したアプローチの詳細については、ブログ記事

Security and Containers in EDB Postgres AI for CloudNativePG Cluster を参照してください。

5.4.3 クラスター

クラスターレベルのセキュリティでは、コントロールプレーンとノードの両方を形成するすべての Kubernetes コンポーネント、およびクラスターで実行されるアプリケーション PostgreSQL を含むが考慮されます。最初に、このセクションの残りの部分を構成する信頼モデルとセキュリティ境界を定義します。

信頼モデルとセキュリティ境界

CloudNativePG クラスターの EDB Postgres AI は、セキュリティポリシーを適用するためのオペレーターと Kubernetes プラットフォーム間の明確な責任の分離に従います。以下のサブセクションでは、PostgreSQL ポッドに影響を与えると信頼されている人、適用境界が実際にある場所、オペレーターが管理するリソースを制限する方法、およびこれらすべてへのアクセスが RBAC によって制御される方法について説明します。

信頼できるクラスターリソースライター

`Cluster` カスタムリソースの作成または更新権限を持つユーザーは、オペレーターが管理する PostgreSQL ポッドの完全な仕様を定義すると信頼されます。これには、`spec.securityContext` 、`spec.podSecurityContext` 、`spec.serviceAccountName` などのセキュリティ重要フィールドが含まれます。オペレーターは、指示に従ってこれらの設定を適用します。クラスターライタがそれらをオーバーライドしない場合、オペレーターは強化されたデフォルトのポッドとコンテナセキュリティコンテキストを適用しますポッドとコンテナのセキュリティコンテキスト を参照してください。

これは、すべての Kubernetes オペレーターの動作と一貫しています。オペレーターは、高レベルのインテントを低レベルの Kubernetes リソースに変換する特権エージェントとして機能します。クラスターリソースへの書き込みアクセスを誰かに付与することは、生成されるポッド、サービス、PVC、およびその他の管理対象リソースに対する制御を付与することと同じです。

同じ原則が *CNPG-I plugins* に適用されます。プラグインは Pod のライフサイクルにフックし、Pod が作成される前に Pod 仕様のフィールドを変更できます。したがって、プラグインのインストールは、クラスターリソースへの書き込みアクセスを付与することと同様に、クラスター管理者によって行われる明示的な信頼の決定です。

同じ権限がデータベース自分自身にも拡張されます。オペレーターはスーパーユーザーとして PostgreSQL をリコンサイルするため、クラスターライタは、次のような文書化された目的のフィールドを介して、結果のクラスターに対してスーパーユーザーレベルの影響力を取得します。

- ブートストラップ SQL フック `postInitSQL` 、`postInitApplicationSQL` 、および `postInitTemplateSQL` 、初期化時に `postgres` スーパーユーザーとして実行されます初期化後のクエリの実行 を参照してください。
- インライン管理対象ロール

`spec.managed.roles superuser: true` を設定するか、既存のロールのメンバーシップを付与でき、`COPY ... FROM PROGRAM` などの PostgreSQL 機能を使用してオペランドポッド内でコマンドを実行できます。

- カスタム PostgreSQL 構成

これは意図された信頼モデルであり、特権のエスカレーションではありません。

オペランドポッドで実行されているコードは、マウントされた `ServiceAccount` の下でも実行され、独自のクラスターに関連付けられたシークレットを読み取ることができます**インスタンスマネージャーによる *API* サーバーへの呼び出し** を参照してください

および *External Secrets*)。したがって、クラスターライタは、**名前空間の分離** によってクラスター自分自身の名前空間に制限されているこれらのシークレットを読み取ることができます。

> **注釈**
>
> これらの信頼境界は、プロジェクトの self security assessment でもカバーされています。

Kubernetes Admission Control

オペレーターは、標準の Kubernetes API を介してポッドを作成します。つまり、すべてのポッドがクラスターに構成されている完全なアドミッション制御チェーンを通過します。これには

Pod Security Admission が含まれます

PSA、 Pod Security Standards を適用するビルトイン Kubernetes メカニズム

名前空間レベル、および使用中のサードパーティポリシーエンジンで。名前空間が PSA `restricted` または `baseline` プロファイルを適用する場合、Kubernetes API サーバーは、オペレーターが作成したポッドを含む、ポリシーに違反するポッドを拒否します。

上記の信頼できるクラスターライターは任意のポッドフィールドを設定できるため、このアドミッション制御チェーンオペレーターではなく、環境にポッドセキュリティポリシーを適用する境界となります。

名前空間の分離

オペレーターは、すべての管理対象リソースを親クラスターの名前空間に制限します。インスタンスポッドは常にクラスターと同じ名前空間に作成され、他の関係から推測されるのではなく明示的かつ決定論的に設定されます。Kubernetes の所有者参照は単一の名前空間に制限されているため、オペレーターは他の名前空間にリソースの所有権を確立することもできません。

カスタムリソースの#### RBAC

CloudNativePG クラスターカスタムリソースの EDB Postgres AI へのアクセスは、標準の Kubernetes RBAC によって管理されます。クラスターリソースへの書き込みアクセスは、結果のポッドの制御を意味するため、`k8s.enterprisedb.io/podPatch` ユーザー指定パッチを管理対象ポッドに適用する `k8s.enterprisedb.io/validation` 、`k8s.enterprisedb.io/reconciliationLoop` のようなアノテーションは、クラスターリソース自分自身と同じ RBAC ルールの対象となります。

特に、`k8s.enterprisedb.io/validation` を `disabled` に設定すると、`k8s.enterprisedb.io/podPatch` で実行される構文と適用性チェックを含む、そのクラスターのオペレーター独自の検証用アドミッション

Webhook がバイパスされます。したがって、CloudNativePG クラスターの EDB Postgres AI の Webhook 検証はアドバイザリーです。クラスターを書き込むことができる同じアクターがオフにできるため、セキュリティ境界ではありません。Kubernetes アドミッションコントロールは、このアノテーションに関係なく、結果のポッドに適用されます。

同じ理由が `DatabaseRole` リソースに適用されます。その仕様では、`superuser: true` と、`postgres` ロール自分自身を含む `inRoles` を介した既存のロールのメンバーシップが許可されています。したがって、名前空間内の `DatabaseRole` オブジェクトへの書き込みアクセスは、その名前空間の PostgreSQL クラスターへのスーパーユーザーアクセスと同等であり、`Cluster` リソースへの書き込みアクセスと同じ注意が必要です。

これらのリソースへの書き込みアクセスを高特権付与として扱います。マルチテナント展開の場合、テナントを別の名前空間に分離し、Kubernetes RBAC で名前空間ごとにスコープを設定し、**名前空間の分離** に依存して影響を制限します。

ここで説明する RBAC は、CloudNativePG クラスター独自のカスタムリソースの EDB Postgres AI へのアクセスを制御します。次のセクションでは、補完的な側、つまり、オペレーター自分自身が依存して、お客様に代わって Kubernetes リソースを管理する RBAC について説明します。

ロールベースのアクセス制御 RBAC

このセクションでは、上記の RBAC on Custom Resources で説明したカスタムリソースへのユーザーアクセスとは対照的に、オペレーターの_own_ RBAC に代わって Kubernetes リソースを管理するために使用するサービスアカウントとロールについて説明します。

オペレーターは、`postgresql-operator-manager` という名前の専用サービスアカウントを使用して Kubernetes API サーバーと対話します。このサービスアカウントは、通常、オペレーター名前空間、一般的に `postgresql-operator-system` にインストールされます。ただし、名前空間は、展開方法によって異なる場合があります以下のサブセクションを参照してください。

同じ名前空間に、`postgresql-operator-manager` サービスアカウントとロールの間にバインディングがあります。このロールの具体的な名前とタイプ `Role` または `ClusterRole` も、展開方法によって異なります。このロールは、オペレーターが正常に機能するために必要な権限を定義します。これらのロールの詳細については、展開方法に応じて、`kubectl describe clusterrole` または `kubectl describe role` コマンドを使用できます。この問題に関する OpenShift の詳細については、*Red Hat OpenShift* 、特に *Pre-defined RBAC objects* を参照してください。

> **重要**
>
> 上記の権限は、Kubernetes API サーバーと対話するためにオペレーターのサービスアカウント専用に予約されています。これらは、 *Cluster* 、*Pooler* 、*Backup* 、*ScheduledBackup* 、*Database* 、*Publication* 、*Subscription* 、*ImageCatalog* 、および`ClusterImageCatalog` リソースとのみ対話するオペレーターのユーザーは直接アクセスできません。

以下に、いくつかの例と、最も重要 EDB Postgres 理由を提供します。

`configmaps`

オペレーターは、Prometheus エクスポーターモニタリングメトリックのデフォルトの構成マップを作成および管理する必要があります。

deployments

オペレーターは、標準の Kubernetes Deployment リソースを使用して PgBouncer 接続プーラーを管理する必要があります。

jobs

オペレーターは、ジョブを処理して、さまざまな Cluster のフェーズを管理する必要があります。

persistentvolumeclaims

PGDATA が存在するボリュームは、PostgreSQL Cluster リソースの中心的な要素です。オペレーターは、選択したストレージクラスと対話して、定義されたスケジューリングポリシーに基づいて、要求されたボリュームを動的にプロビジョニングする必要があります。

pods

オペレーターは、Cluster のインスタンスを管理する必要があります。

secrets

Cluster オブジェクトに証明書とパスワードを提供しない限り、オペレーターは、ランダムに生成されたパスワードと TLS 証明書をセルフプロビジョニングし、シークレットに保存することにより、「構成より規約」パラダイムを採用します。

serviceaccounts

オペレーターは、インスタンスマネージャー PostgreSQL サーバーを制御するコンテナの_PID 1_プロセスが Kubernetes API サーバーと安全に通信してアクションを調整し、Cluster の信頼できるステータスを継続的に提供できるサービスアカウントを作成する必要があります。

services

オペレーターは、アプリケーションから PostgreSQL クラスターまたは接続プーラーへのネットワークアクセスを制御し、自動化された方法でフェールオーバー/スイッチオーバー操作を適切に管理する必要がありますたとえば、サービスの正しいエンドポイントを適切なプライマリに割り当てる PostgreSQL インスタンス)。

validatingwebhookconfigurations および mutatingwebhookconfigurations

オペレーターは、自己署名 Webhook CA を両方の Webhook 構成に注入します。これらは、管理するすべてのリソースを検証および変更するために必要です。詳細については、

Kubernetes documentation を参照してください。

volumesnapshots

オペレーターは、PostgreSQL サーバーのバックアップを取得するために、VolumeSnapshots オブジェクトを生成する必要があります。VolumeSnapshot は、復元プロセスを開始する前に検証するために読み取られます。

nodes

オペレーターは、アフィニティとアンチアフィニティのラベルを取得して、ポッドをスケジュールできるノードを決定できるようにする必要があります。これは、たとえば、レプリカが同じノードでスケジュールされるのを防ぐ場合に役立ちます。ノードが異なるアベイラビリティーゾーンにある場合に特に重要です。この権限は、ノードがスケジュールされているかどうかを判断し、スケジュールされていないノード

でのポッドの作成を防止、またはプライマリがスケジュールされていないノードに存在する場合にスイッチオーバーをトリガーするためにも使用されます。

デプロイメントと ClusterRole リソース

前述のように、各展開方法には、サービスアカウントの名前空間の場所、ロールバインディング、および各ロールの名前とタイプが異なる場合があります。

Kubernetes マニフェスト経由

Kubernetes マニフェストを使用して CloudNativePG クラスターの EDB Postgres AI をインストールすると、権限はデフォルトで ClusterRoleBinding に設定されます。次を実行して、オペレーターに必要な権限を検査できます。

```
kubectl describe clusterrole postgresql-operator-manager
```

OLM 経由

セキュリティの観点から、Operator Lifecycle ManagerOLM は、より柔軟な展開方法を提供します。すべての名前空間または特定の名前空間を監視するようにオペレーターを構成できるため、より詳細な権限管理が可能になります。

> **注釈**
>
> OLM では、オペレーターを独自の名前空間に展開し、CloudNativePG Cluster クラスターの EDB Postgres AI に使用される特定の名前空間を監視するように構成できます。この設定は、権限を含め、より効果的にアクセスを制限するのに役立ちます。

クラスターロールのアクセス許可が必要なのはなぜですか?

現在、オペレーターが nodes および ClusterImageCatalog オブジェクトを読み取るには、ClusterRole 権限が必要です。他のすべての権限は、名前空間スコープつまり Role またはクラスター全体つまり ClusterRole にすることができます。

ClusterRoles 権限を削除すると、オペレーターの Deployment 定義を編集して実験的な--no-cluster-roles=true コマンドライン引数を明示的に設定しない限り、オペレーターはクラッシュします。このフラグを有効にすると、オペレーターの ClusterImageCatalog オブジェクトを操作し、Node ステータスの変更に応答する機能が無効になることに注意してください。これはプーラーリソースにも影響します。PgBouncer イメージの ClusterImageCatalog を参照するプーラーは、カタログの更新に反応しません。

これらの権限でも、誰かが ServiceAccount へのアクセスを取得した場合、リソースの表示に制限されている get 、list 、および watch 権限のみがあります。ただし、無許可のユーザーが ServiceAccount へのアクセスを取得した場合、それはより重大なセキュリティ問題を示しています。

したがって、ユーザーがオペレーターの ServiceAccount および昇格した権限で他の ServiceAccount にアクセスしないようにすることが重要です。

インスタンスマネージャーによる API サーバーへの呼び出し

オペランドコンテナのエントリポイントであるインスタンスマネージャーは、Kubernetes API サーバーへのいくつかの呼び出しを行って、一部のリソースのステータスが正しく更新されることを確認し、そのPostgres クラスターに関連付けられている構成マップとシークレットにアクセスする必要があります。。このような呼び出しは、同じ PostgreSQL `Cluster` リソース名を共有するオペレーターによって作成された専用の `ServiceAccount` を介して実行されます。

> **重要**
>
> このオペランドは、API サーバーを介してリソースの特定の限定されたサブセットにのみアクセスできます。サービスアカウントは recommended way to access the API server from within a Pod です。

共有サービスアカウントを使用する

デフォルトでは、CloudNativePG クラスターの EDB Postgres AI は、クラスター自分自身にちなんで名前を付けて、クラスターごとに専用の `ServiceAccount` を作成します。ただし、IAM ロール AWS IRSA、GCP Workload Identity、Azure Workload Identity などを使用するクラウド環境では、独自の `ServiceAccount` を作成する各クラスターは、新しいクラスターごとに個別の IAM 構成を必要とし、複数のクラスターを管理する際のスケーリングが困難になります。

CloudNativePG クラスターの EDB Postgres AI を使用すると、クラスター仕様で `serviceAccountName` フィールドを指定することにより、複数のクラスターが単一の `ServiceAccount` を共有できます。これにより、その `ServiceAccount` を使用するすべてのクラスターで機能するワンタイム IAM 構成が有効になります。

> **重要**
>
> 共有 `ServiceAccount` を使用する場合、自分自身の責任で `ServiceAccount` を作成および管理します。オペレーターは、指定された `ServiceAccount` が存在することを検証しますが、作成または変更しません。

次に、共有 `ServiceAccount` を使用する例を示します。

```
#  Create the shared ServiceAccount once

apiVersion: v1
kind: ServiceAccount
metadata:
  name: postgres-cloud-sa
  annotations:
    # AWS IRSA annotation
    eks.amazonaws.com/role-arn: arn:aws:iam:us-east-1:123456789012:role/PostgresRole
- --

#  Reference it from multiple clusters
```

(次のページに続く)

(前のページからの続き)

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-prod
spec:
  serviceAccountName: postgres-cloud-sa
  instances: 3
  storage:
    size: 100Gi
```

注釈

serviceAccountName フィールドは `serviceAccountTemplate` と相互排他的です。オペレーターに `ServiceAccount` を管理させるか、オプションで `serviceAccountTemplate` でカスタマイズするか、*serviceAccountName* で既存のを参照できますが、両方はできません。

警告

serviceAccountName フィールドは、一度設定されると変更できません。 *ServiceAccount* を変更する必要がある場合は、クラスターを再作成する必要があります。

注釈

複数のクラスターが `ServiceAccount` を共有する場合、オペレーターはクラスターごとに個別の `Role` および `RoleBinding` を作成します。共有 `ServiceAccount` は、それを使用するすべてのクラスターの Kubernetes RBAC 権限を蓄積します。各 `Role` のスコープは特定のクラスターのリソースに狭いため、これは特権エスカレーションを構成しません。

プーラーでの共有サービスアカウントの使用

同じ共有 `ServiceAccount` 機能は `Pooler` リソースでも使用できます。IAM 統合を使用するクラウド環境に PgBouncer プーラーを展開する場合、プーラー仕様の `serviceAccountName` フィールドを指定して、オペレーターにプーラーごとに 1 つ作成させる代わりに、既存の `ServiceAccount` を参照できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Pooler
metadata:
  name: pooler-prod
spec:
  cluster:
    name: cluster-prod
  serviceAccountName: pgbouncer-cloud-sa
```

(次のページに続く)

(前のページからの続き)

```
  instances: 3
  pgbouncer:
    poolMode: session
```

> **警告**
>
> クラスターと同様に、プーラーの`serviceAccountName`フィールドは、一度設定されると変更できません。 *ServiceAccount* を変更する必要がある場合は、プーラーを再作成する必要があります。

透明性を維持するために、サービスアカウントに関連付けられている権限は roles.go

ファイル。たとえば、 myns 名前空間の汎用 mypg クラスターの権限を取得するには、次のコマンドを入力します。

```
kubectl get role -n myns mypg -o yaml
```

次に、ロールがサービスアカウントにバインドされていることを確認します。

```
kubectl get rolebinding -n myns mypg -o yaml
```

> **重要**
>
> **ロールは特定の名前空間**に制限されていることに注意してください。

以下に、汎用の Kubernetes リソースのサービスアカウントに関連付けられた権限の簡単な概要を提供します。

`configmaps`

インスタンスマネージャーは、カスタムモニタリングクエリなど、同じクラスターに関連する構成マップのみを読み取ることができます

`secrets`

インスタンスマネージャーは、同じクラスターに関連するシークレットのみを読み取ることができます。ストリーミングレプリケーションユーザー、アプリケーションユーザー、スーパーユーザー、LDAP 認証ユーザー、クライアント CA、サーバー CA、サーバー証明書、バックアップ資格情報、カスタムモニタリングクエリー

`events`

インスタンスマネージャーは、クラスターのイベントを作成し、PostgreSQL インスタンスのライフサイクルの特定の側面について API サーバーに通知できます

ここでは、代わりに、CloudNativePG クラスターの EDB Postgres AI に固有のリソースに関する同じ概要を提供します。

`clusters`

インスタンスマネージャーは、独自の Cluster リソースに対してのみ、読み取り専用権限、つまり get 、list 、および watch が必要です

clusters/status

インスタンスマネージャーは、自分自身の Cluster リソースのみのステータスを update および patch する必要があります

backups

インスタンスマネージャーが名前空間内の Backup リソースを読み取るには、 get および list 権限が必要です。さらに、オブジェクトストアに対応するものがない Backup オブジェクトを削除することにより、Kubernetes クラスターをクリーンアップするには、delete 権限が必要です。通常はリテンションポリシーのため

backups/status

インスタンスマネージャーは、名前空間内の Backup リソースのステータスを update および patch する必要があります。

ポッドとコンテナのセキュリティコンテキスト

Security Context

ポッドまたはコンテナの特権とアクセス制御設定を定義します。

CloudNativePG クラスターの EDB Postgres AI は、コンテナの実行に_privileged_モードを必要としません。PostgreSQL コンテナは、 postgres システムユーザーとして実行されます。root として実行する必要があるコンポーネントはありません。

同様に、ボリュームアクセスには、_privileged_モードも root 特権も必要ありません。Kubernetes プラットフォームおよび/または管理者が適切な権限を割り当てる必要があります。PostgreSQL コンテナは、読み取り専用のルートファイルシステムつまり書き込み可能なレイヤーなしで実行されます。

オペレーターは、PostgreSQL クラスターのすべてのポッドとコンテナのセキュリティコンテキストの設定を管理します。

Seccomp Profile

PostgreSQL コンテナに使用するサービスは、 Cluster リソースの spec.seccompProfile セクションで構成できます。このセクションが空白のままの場合、コンテナは seccomp プロファイル Type または RuntimeDefault 、つまりコンテナランタイムのデフォルトを使用します。

デフォルトの seccompProfile を使用する PostgreSQL コンテナのセキュリティコンテキストは次のようになります。

```
securityContext:
  allowPrivilegeEscalation: false
  capabilities:
    drop:
    - ALL
  privileged: false
  readOnlyRootFilesystem: true
```

(次のページに続く)

(前のページからの続き)

```
  runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
```

セキュリティコンテキストのカスタマイズ

CloudNativePG クラスターの EDB Postgres AI は、それぞれ spec.podSecurityContext および spec.securityContext フィールドを介してポッドとコンテナの両方のセキュリティコンテキストのきめの細かい制御を提供します。

ポッドセキュリティコンテキスト spec.podSecurityContext これにより、すべての PostgreSQL クラスターポッドに適用されるデフォルトの PodSecurityContext をオーバーライドできます。指定すると、演算子のデフォルト設定とマージされ、明示的に設定されたフィールドの値が優先されます。

例

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3
  podSecurityContext:
    runAsUser: 26
    runAsGroup: 26
    fsGroup: 26
    supplementalGroups: [2000, 3000]
    fsGroupChangePolicy: "OnRootMismatch"
```

コンテナセキュリティコンテキスト spec.securityContext これにより、PostgreSQL クラスターポッド内のすべてのコンテナに適用されるデフォルトの SecurityContext をオーバーライドできます。podSecurityContext と同様に、演算子のデフォルトとマージします。

例

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3
  securityContext:
    allowPrivilegeEscalation: false
    # Note: capabilities are not merged with operator defaults.
    # If specified, they fully replace any defaults.
    capabilities:
```

(次のページに続く)

(前のページからの続き)

```
    drop:
      - ALL
    add:
      - NET_BIND_SERVICE
  readOnlyRootFilesystem: true
  runAsNonRoot: true
```

注釈

これらのフィールドは、 Pod Security Standards を使用する場合に特に役立ちます。

restricted プロファイル、ポッドとコンテナのセキュリティコンテキストの厳密な要件があります。*Kubernetes Admission Control* で説明しているように、結果の Pod にこのようなプロファイルを適用するのは、オペレーターではなくこのアドミッションチェーンです。

セキュリティコンテキストの制約

Security Context Constraints (SCC) を利用している環境で実行されている場合

オペレーターは、PostgreSQL クラスターポッドのセキュリティコンテキストを明示的に設定せず、ポッドが既に定義されている制限されたセキュリティコンテキストコンストレインを継承することを許可します。

Red Hat OpenShift では、クラウドネイティブの PostgreSQL は、最も制限の厳しい restricted セキュリティコンテキスト制約で実行されます。目標は、ポッドの実行を名前空間に割り当てられた UID と SELinux コンテキストに制限することです。

参考

OpenShift のセキュリティコンテキスト制約 SCC の詳細については、

Managing SCC in OpenShift を参照してください。

記事。

Openshift documentation の記載

SCC はデフォルトの名前空間 default 、kube-system 、kube-public 、openshift-node 、openshift-infra 、openshift には適用されません。これらはポッドの実行に使用しないでください。これらの名前空間に展開された CNP クラスターは、SCC が欠落しているため起動できません。

AppArmor を使用してポッドアクセスを制限する

container.apparmor.security.beta.kubernetes.io アノテーションを介して、すべての Cluster ポッド内の postgres、initdb、join、full-recovery、および bootstrap-controller コンテナに AppArmor プロファイルを割り当てることができます。例

```
kind: Cluster
metadata:
  name: cluster-apparmor
  annotations:
    container.apparmor.security.beta.kubernetes.io/postgres: runtime/default
    container.apparmor.security.beta.kubernetes.io/initdb: runtime/default
    container.apparmor.security.beta.kubernetes.io/join: runtime/default
```

警告

この種類のアノテーションを使用すると、クラスターが動作を停止する可能性があります。この場合、アノテーションは`Cluster` から安全に削除できます。

AppArmor 構成は Kubernetes ノードレベルである必要があります。つまり、基になるオペレーティングシステムでこのオプションを有効にし、適切に構成する必要があります。

これは状況ではなく、アノテーションが `Cluster` 作成時に追加された場合、ポッドは作成されません。一方、`Cluster` が作成された後にアノテーションを追加すると、クラスター内のポッドは起動できず、次のようなエラーが発生します。

```
metadata.annotations[container.apparmor.security.beta.kubernetes.io/postgres]:␣
↪Forbidden: may not add AppArmor annotations]
```

このような場合、Kubernetes 管理者に連絡し、使用する適切な AppArmor プロファイルを聞いてください。

ネットワークポリシー

`Cluster` リソースによって作成されたポッドは、Kubernetes によって制御できます network policies

IP および TCP レベルでインバウンドおよびアウトバウンドのネットワークアクセスを有効/無効にします。詳細については、*networking document* をご覧ください。

重要

オペレーターは、TCP ポート 8000 で各インスタンスと通信して、PostgreSQL サーバーのステータスに関する情報を取得する必要があります。ネットワークポリシーを追加する場合にこれを念頭に置いて、より詳細な制御のために CloudNativePG クラスターの EDB Postgres AI が使用するポートのリストについては、以下の「公開ポート」セクションを参照してください。

ネットワークポリシーは、このドキュメントの範囲外です。Network policies を参照してください。

詳細については、Kubernetes ドキュメントのセクションを参照してください。

公開ポート

CloudNativePG クラスターの EDB Postgres AI は、以下の表にリストされているように、オペレーター、インスタンスマネージャー、およびオペランドレベルでポートを公開します。

System	Port number	Exposing	Name	TLS	Authentication
operator	9443	webhook server	*webhook-server*	Yes	Yes
operator	8080	metrics	*metrics*	No	No
instance manager	9187	metrics	*metrics*	Optional	No
instance manager	8000	status	*status*	Yes	Partial (1)
operand	5432	PostgreSQL instance	*postgresql*	Optional	Yes

(1) ステータス、状態、およびプローブエンドポイントは認証されていません。**オペレーターからインスタンスへの認証** で説明しているように、機密エンドポイントバックアップ、`pg_controldata` 、部分的な WAL アーカイブ、およびインスタンスマネージャーのアップグレードは、オペレーターのクライアント証明書が必要です

以下。

オペレーターからインスタンスへの認証

オペレーターは、起動時にメモリに自己署名 ECDSA クライアント証明書を生成し、クラスターの`.status.operatorCertificateFingerprint` で SHA-256 公開キーフィンガープリントを公開します。インスタンスマネージャーは、そのフィンガープリントを固定し、一致する証明書を提示しない機密エンドポイント backup、`pg_controldata` 、部分的な WAL アーカイブ、およびインスタンスマネージャーのアップグレードへの呼び出しを拒否します。ステータス、状態、およびプローブエンドポイントは認証されないままです。

証明書はディスクに書き込まれることはなく、オペレーターが再起動するたびに再生成されるため、CA 検証ではなくフィンガープリントのピンニングから信頼が得られます。

この保護には、ステータスポート **が TLS で提供される必要があります**というハード要件があります。これは、v1.24 以降のデフォルトです。クライアント証明書は TLS 接続でのみ提示できるため、ステータスポートが TLS を使用する場合にのみ、オペレーターが保護されたエンドポイントに到達できます。

> **警告**
>
> ステータスポートが TLS で提供されていない場合、インスタンスマネージャーはオペレーターを認証できず、*401 Unauthorized* で保護されたエンドポイント backup、*pg_controldata* 、部分的な WAL アーカイブ、およびインスタンスマネージャーのアップグレードへのすべての呼び出しを**永久に**拒否します。これは一時的な状態ではなく、自然に解決されることはありません。インスタンスマネージャーがこの認証を強制するバージョンにアップグレードされると、v1.24 より古いオペレーターによって作成されたインスタンスに影響を与える可能性があります。このようなインスタンスは、TLS でポッドが再作成されるようにロールアウトする必要があります。有効なステータスポート。新しく作成されたインスタンスは常にステータスポートで TLS を有効にし、影響を受けません。

PostgreSQL

CloudNativePG クラスターの EDB Postgres AI の現在の実装は、データベース所有者と、`enableSuperuserAccess` を `true` に設定して要求された場合にのみ、`postgres` スーパーユーザーのパスワードと`.pgpass` ファイルを自動的に作成します。

> **警告**
>
> *enableSuperuserAccess* は、オペレーターのセキュリティバイデフォルト姿勢を向上させるためにデフォルトで`false` に設定され、PostgreSQL への変更が`Cluster` リソースの`spec` を介して宣言的な方法で実行されるマイクロサービスアプローチを促進しながら、開発者に内部のフルパワーを提供しますデータベース所有者ユーザーを介してデータベースにアクセスします。

パスワードの暗号化に関する限り、CloudNativePG クラスターの EDB Postgres AI は、PostgreSQL のデフォルト動作に従います。PostgreSQL 14 以降、`password_encryption` はデフォルトで `scram-sha-256` に設定されますが、以前のバージョンでは `md5` に設定されています。

> **重要**
>
> Password authentication を参照してください。

詳細については、PostgreSQL ドキュメントのセクションを参照してください。

> **注釈**
>
> オペレーターは、*enableSuperuserAccess* オプションのトグルをサポートしています。実行中のクラスターで無効にすると、オペレーターはシークレットの内容を無視し、削除し、オペレーターが以前に生成した場合、*postgres* ユーザーのパスワードを`NULL` に設定しますパスワード認証を介したリモートアクセスを事実上無効にします。

詳細については、 *External Secrets* を参照してください。

これらのファイルを使用して、データベースへのアプリケーションのアクセスを構成できます。

デフォルトでは、すべてのレプリカは、`streaming_replica` と呼ばれる特別なユーザーを使用して、現在のプライマリインスタンスに**物理非同期ストリーミングレプリケーション**で接続するように自動的に構成されます。ノード間の接続は **暗号化**され、認証は **TLS クライアント証明書**を介して行われます。詳細については、*Client TLS/SSL connections* ページを参照してください。デフォルトでは、オペレーターは TLS v1.3 接続を必要とします。

現在、オペレーターは、管理者が `postgresql` 構成の `pg_hba` セクションの一部としてマニフェストに `pg_hba.conf` 行を直接追加できます。マニフェストで定義された行は、デフォルトの `pg_hba.conf` に追加されます。

オペレーターによる `pg_hba.conf` の管理方法の詳細については、ドキュメントの *PostgreSQL Configuration* を参照してください。

管理者は、デフォルトでローカルの postgres ユーザーをデータベース内の postgres ユーザーにのみマップする `pg_ident.conf` ファイルの内容をカスタマイズすることもできます。

オペレーターによる `pg_ident.conf` の管理方法の詳細については、ドキュメントの *PostgreSQL Configuration* を参照してください。

重要

例では、Kubernetes クラスターがプライベートで安全なネットワークで実行されていることを前提としています。

スキーマの解決と search_path の強化

データベースに対する特権を持つユーザーは、`public` などの書き込み可能なスキーマにオブジェクトファンクション、演算子、テーブル、またはタイプを植え付け、データベースまたはロールレベルの `search_path` たとえば `ALTER DATABASE ... SET search_path` または `ALTER ROLE ... SET search_path` を変更できます。後でそのデータベースに接続する特権セッションは、テナント制御の `search_path` を継承するため、クエリのいずれかにおける非修飾参照は、目的のオブジェクトではなく植えられたオブジェクトに解決される場合があります。これは、CWE-426 (Untrusted Search Path) に似た特権エスカレーションベクターであり、よく知られた CVE-2018-1058 と同じクラスの問題です。

これを防ぐために、CloudNativePG クラスターの EDB Postgres AI は、データベースまたは接続ロールで構成されている `search_path` に関係なく、PostgreSQL への接続を開くすべての接続で `search_path` を `pg_catalog, public, pg_temp` の固定値に固定します。

- `pg_catalog` が最初に検索されるため、ビルトインオブジェクトは常に別のスキーマに植えられた同じ名前のオブジェクトより優先されます。
- `pg_temp` セッションプライベート一時スキーマは、最初に検索されるのではなく最後に検索されるため、リレーションまたはデータ型をシャドウできません。

さらに、PgBouncer 統合で使用される `SECURITY DEFINER` ルックアップファンクションは、独自の固定された `search_path` で作成され、モニタリングクエリーは、展開元 hostssl postgres Monitor ${podselector:monitoring} scram-sha-256 で説明しているように `search_path` が固定されているトランザクション内で実行されます。

注釈

ユーザー作成 SQL ブートストラップ中の `postInitSQL` */postInitApplicationSQL /postInitTemplateSQL* 、および論理インポートのインポート後クエリーは、標準の `"$user", public` 解像度で実行されるため、プレーンの PostgreSQL セッションと同様に動作し続けます。 *search_path* から独立する必要がある場合、これらのスクリプトのオブジェクト参照をスキーマ修飾します。

ストレージ

CloudNativePG クラスターの EDB Postgres AI は、保存時の暗号化を基になるストレージクラスに委任します。運用環境でのデータ保護のために、保存時の暗号化をサポートするストレージクラスを選択することを強くお勧めします。

5.5 Postgres Instance Manager

CloudNativePG クラスターの EDB Postgres AI は、フェールオーバー管理の外部ツールに依存しません。これは、Kubernetes API サーバーと、**Postgres インスタンスマネージャー**と呼ばれるネイティブキーコンポーネントに依存するだけです。

インスタンスマネージャーは、PostgreSQL サーバープロセス `postmaster` としても知られるライフサイクル全体を処理します。

新しいクラスターを作成すると、オペレーターはインスタンスごとにポッドを作成します。フィールド `.spec.instances` は、作成するインスタンスの数を指定します。

各ポッドは、メインコンテナの親プロセス PID 1 としてインスタンスマネージャーを起動し、PostgreSQL インスタンスを実行します。ポッドのライフタイム中、インスタンスマネージャーはバックエンドとして機能して startup, liveness and readiness probes を処理します。

5.5.1 スタートアッププローブ

起動プローブは、PostgreSQL インスタンスがプライマリかスタンバイかにかかわらず、完全に起動していることを確認します。

> **注釈**
>
> デフォルトでは、起動プローブは pg_isready を使用します。ただし、別の起動戦略を指定することにより、動作をカスタマイズできます。

startup プローブの実行中、liveness プローブと readness プローブは無効のままです。Kubernetes 標準に従って、startup プローブが失敗すると、kubelet はコンテナを終了し、再起動します。

`.spec.startDelay` パラメーターは、startup プローブが成功するまでに許可される最大時間を秒単位で指定します。

デフォルトでは、`startDelay` は `3600` 秒に設定されます。特定の環境で PostgreSQL が完全に初期化する必要がある時間に基づいて、この設定を調整することをお勧めします。

> **警告**
>
> *.spec.startDelay* の設定が低すぎると、liveness プローブが途中でアクティブ化され、PostgreSQL が完全に初期化されていない場合は不必要な Pod が再起動する可能性があります。

CloudNativePG クラスターの EDB Postgres AI は、次のデフォルトパラメーターで起動プローブを構成します。

```
failureThreshold: FAILURE_THRESHOLD
periodSeconds: 10
successThreshold: 1
timeoutSeconds: 5
```

failureThreshold 値は、 startDelay を periodSeconds で除算することにより自動的に計算されます。

構成の.spec.probes.startup セクションでプローブ設定をカスタマイズできます。

警告

カスタムプローブ設定がクラスターの操作要件に合わせて調整されていることを確認して、意図しない中断を回避します。

注釈

プローブ構成の詳細については、 *probe API documentation* を参照してください。

.spec.probes.startup.failureThreshold を手動で指定すると、デフォルトの動作がオーバーライドされ、startDelay の自動使用が無効になります。

たとえば、次の構成は、 startDelay をバイパスして、カスタムプローブパラメーターを明示的に設定します。

```
#  ... snip

spec:
  probes:
    startup:
      periodSeconds: 3
      timeoutSeconds: 3
      failureThreshold: 10
```

スタートアッププローブ戦略

特定のシナリオでは、PostgreSQL クラスターの起動戦略をカスタマイズする必要がある場合があります。たとえば、プライマリからストリーミングを開始するまでレプリカを開始済みとしてマーキングするのを遅らせるか、レプリカの準備ができていると見なされる前に満たす必要があるレプリケーションラグしきい値を定義できます。

これらの要件に対応するために、CloudNativePG クラスターの EDB Postgres AI は、2 つのオプションのパラメーターで.spec.probes.startup スタンザを拡張します。

- type プローブが成功したとみなすための基準を指定します。受け入れられる値、複雑さ/深さの順に、次のものが含まれます。
- pg_isready pg_isready コマンドが 0 で終了したときに、プローブを成功としてマークします。これは、プライマリインスタンスとレプリカのデフォルトです。
- query 基本的なクエリーが postgres データベースでローカルに実行されたときにプローブを成功としてマークします。

- `streaming` レプリカがソースからストリーミングを開始し、指定された条件を満たしたときにプローブを成功としてマークします。ラグ要件詳細以下。
- `maximumLag` バイト単位で測定される最大許容レプリケーションラグを定義します Kubernetes 量として表現されます。このパラメーターは、`type` が `streaming` に設定されている場合にのみ適用されます。`maximumLag` が指定されない場合、レプリカはストリーミングを開始するとすぐに正常に開始されたとみなされます。

> **⚠ 警告**
>
> *maximumLag* オプションの構成が正しくない場合、startup プローブの継続的な失敗が発生し、レプリカの再起動が繰り返される場合があります。このオプションの動作を理解し、*failureThreshold* および `periodSeconds` の適切な値を構成して、レプリカがソースに追いつくのに十分な時間を与えます。

次の例では、レプリカが開始されたとみなされるソースからの最大 16Mi の遅延が必要です。

```
#  <snip>

probes:
  startup:
    type: streaming
    maximumLag: 16Mi
```

5.5.2 Liveness Probe

liveness プローブは、startup プローブが正常に完了した後に開始されます。その主な役割は、PostgreSQL インスタンスマネージャーが正常に動作していることを確認することです。

Kubernetes 標準に従って、liveness プローブが失敗すると、kubelet はコンテナを終了し、再起動します。

ポッドが生きていないとして分類されるまでの時間は、`.spec.livenessProbeTimeout` パラメーターを介して構成できます。

CloudNativePG クラスターの EDB Postgres AI は、次のデフォルトパラメーターを使用して liveness プローブを構成します。

```
failureThreshold: FAILURE_THRESHOLD
periodSeconds: 10
successThreshold: 1
timeoutSeconds: 5
```

`failureThreshold` 値は、 `livenessProbeTimeout` を `periodSeconds` で除算することにより自動的に計算されます。

デフォルトでは、`.spec.livenessProbeTimeout` は `30` 秒に設定されます。これは、liveness プローブが、各チェックの間に 10 秒の間隔で、3 回連続したプローブ障害を検出した場合に障害を報告することを意味します。

構成の`.spec.probes.liveness` セクションでプローブ設定をカスタマイズできます。

警告

カスタムプローブ設定がクラスターの操作要件に合わせて調整されていることを確認して、意図しない中断を回避します。

注釈

プローブ構成の詳細については、 *probe API documentation* を参照してください。

`.spec.probes.liveness.failureThreshold` を手動で指定すると、デフォルトの動作がオーバーライドされ、`livenessProbeTimeout` の自動使用が無効になります。

たとえば、次の構成は、 `livenessProbeTimeout` をバイパスして、カスタムプローブパラメーターを明示的に設定します。

```
#  ... snip

spec:
  probes:
    liveness:
      periodSeconds: 3
      timeoutSeconds: 3
      failureThreshold: 10
```

プライマリ分離

CloudNativePG クラスター 1.27 の EDB Postgres AI は、PostgreSQL プライマリの活性プローブの追加動作を導入し、次の条件の**両方** が満たされる場合に障害を報告します。

1. インスタンスマネージャーは Kubernetes API サーバーに到達できません
2. インスタンスマネージャーは、インスタンスマネージャーの REST API を介して **他の** インスタンスに到達できません

この動作の効果は、分離されたプライマリが生きていないと考えられ、その後、liveness プローブが失敗したときに**シャットダウン** します。

デフォルトで有効になっています 。以下を追加して無効にできます。

```
spec:
  probes:
    liveness:
      isolationCheck:
        enabled: false
```

また、仕様は 2 つのオプションのネットワーク設定 `requestTimeout` および `connectionTimeout` も受け入れます。どちらもデフォルトで `1000` ミリ秒単位です。クラウド環境では、これらの値を増やす必要がある場合があります。例

```
spec:
  probes:
    liveness:
      isolationCheck:
        enabled: true
        requestTimeout: "2000"
        connectionTimeout: "2000"
```

> **注釈**
>
> プライマリ分離は 安全な予備選挙 とは異なります

メカニズム。分離チェックは、API サーバーと他のインスタンスの両方への接続を失ったプライマリを_フェンス_し、プライマリリースは_どのインスタンスをプロモートできるかを調整します。2 つのメカニズムは相補的です。

スタートアッププローブが正常に完了すると、準備状況プローブが開始されます。その主な目的は、PostgreSQL インスタンスがポッドのライフサイクル中の任意の時点でトラフィックを受け入れ、要求を処理する準備ができているかどうかを確認することです。

> **注釈**
>
> デフォルトでは、準備状況プローブは pg_isready を使用します。ただし、別の準備状況戦略を指定することにより、動作をカスタマイズできます。

Kubernetes 標準に従って、準備状況プローブが失敗した場合、ポッドは未準備としてマークされ、どのサービスからトラフィックを受信しません。準備ができていないポッドは、自動フェイルオーバーシナリオ中のプロモーションの対象外です。

CloudNativePG クラスターの EDB Postgres AI は、準備状況プローブの次のデフォルト構成を使用します。

```
failureThreshold: 3
periodSeconds: 10
successThreshold: 1
timeoutSeconds: 5
```

デフォルト設定が要件に合わない場合は、 `.spec.probes.readiness` スタンザでパラメーターを指定して、readiness プローブを完全にカスタマイズできます。例

```
#  ... snip

spec:
  probes:
    readiness:
      periodSeconds: 3
```

(次のページに続く)

(前のページからの続き)

```
    timeoutSeconds: 3
    failureThreshold: 10
```

警告

カスタムプローブ設定がクラスターの操作要件と一致していることを確認して、意図しない中断を防ぎます。

注釈

プローブの構成の詳細については、 *probe API* を参照してください。

Readness Probe 戦略

特定のシナリオでは、クラスターの準備状況戦略をカスタマイズする必要がある場合があります。たとえば、プライマリからストリーミングを開始するまでレプリカの準備ができていることのマーキングを遅らせるか、レプリカの準備ができていると判断する前に最大レプリケーションラグしきい値を定義できます。

これらの要件に対応するために、CloudNativePG クラスターの EDB Postgres AI は、2 つのオプショナルパラメーター `type` および `maximumLag` で `.spec.probes.readiness` スタンザを拡張します。**スタートアッププローブ戦略** を参照してください。

これらのオプションの詳細については、セクションを参照してください。

警告

maximumLag オプションの誤った構成は、readness プローブの失敗を繰り返して、次のような重大な結果を引き起こす可能性があります。

- フェールオーバー中のプロモーションまたは同期レプリケーションクォーラムへの参加など、キーオペレーター機能からのレプリカの除外。
- 読み取り/読み取り専用サービスの中断。
- **フェールオーバー時間が長いシナリオでは、レプリカの準備ができていないと宣言され、手動介入が必要なクラスターストールが発生する場合があります。 .. ::**

  streaming および `maximumLag` オプションは非常に注意して使用します。 PostgreSQL レプリケーションに慣れていない場合は、デフォルトの戦略を使用してください。不明な場合は、専門家のアドバイスを求めてください。

次の例では、レプリカが準備ができているとみなされるように、ソースからの最大 64Mi の遅延が必要です。また、起動プローブが成功するまでに約 300 秒 30 失敗× 10 秒もかかります。

```
#  <snip>
```

(次のページに続く)

(前のページからの続き)

```
probes:
  readiness:
    type: streaming
    maximumLag: 64Mi
    failureThreshold: 30
    periodSeconds: 10
```

5.5.3 シャットダウン制御

Postgres を実行しているポッドが手動で、またはノードドレイン操作に続いて Kubernetes によって削除されると、kubelet はインスタンスマネージャーに終了信号を送信し、インスタンスマネージャーは適切な方法で PostgreSQL のシャットダウンを担当します。`.spec.smartShutdownTimeout` および `.spec.stopDelay` オプションは秒単位であり、PostgreSQL がシャットダウンするまでの時間を制御します。値はデフォルトでそれぞれ 180 および 1800 秒です。

シャットダウン手順は 2 つの手順で構成されます。

1. インスタンスマネージャーは、最初に `CHECKPOINT` を発行し、次に**スマート**シャットダウンを開始し、PostgreSQL への新しい接続を禁止します。このステップは最大 `.spec.smartShutdownTimeout` 秒間続きます。
2. PostgreSQL がまだアップしている場合、インスタンスマネージャーは**高速**シャットダウンを要求し、既存の接続を終了してすぐに終了します。インスタンスが WAL ファイルのアーカイブおよび/またはストリーミングを行っている場合、プロセスは `.spec.stopDelay` で設定された残り時間まで待機して操作が完了し、強制的にシャットダウンします。このようなタイムアウトは少なくとも 15 秒である必要があります。

スイッチオーバー中のプライマリのシャットダウン

スイッチオーバー中のシャットダウン手順は、一般的な場合と少し異なります。元のプライマリのインスタンスマネージャーは、最初に `CHECKPOINT` を発行し、指定された新しいプライマリが昇格する前に PostgreSQL の **高速**シャットダウンを開始し、新しいプライマリですべてのデータが安全に利用できるようにします。

このため、秒単位で表される `.spec.switchoverDelay` は、元のプライマリが正常にシャットダウンし、すべての WAL ファイルをアーカイブする時間を制御します。デフォルトでは、`3600` 1 時間に設定されています。

> **警告**
>
> *.spec.switchoverDelay* オプションは *リリース候補の目的* に影響します

と PostgreSQL データベースの *RTO* 低い値に設定すると、RPO よりも RTO が優先される場合がありますが、クラスターレベルおよび/またはバックアップレベルでデータの損失が発生します。逆に、高い値に設定すると、スイッチオーバー中にアクティブなプライマリのないクラスターを長時間放置する間のデータ損失のリスクを取り除くことができます。

5.5.4 フェイルオーバー

プライマリポッドに障害が発生した場合、クラスターはフェールオーバーモードになります。詳細は **自己修復のための自動フェイルオーバー** を参照してください。

5.5.5 ディスクフル障害

ストレージの枯渇は、PostgreSQL クラスターのよく知られた問題です。PostgreSQL documentation

では、考えられる障害シナリオと、ディスクの使用状況を監視してディスクがいっぱいになるのを防ぐことの重要性を強調しています。

同じことは、CloudNativePG クラスターおよび Kubernetes の EDB Postgres AI にも適用されます。 展開元 hostssl postgres Monitor ${podselector:monitoring} scram-sha-256

WAL セグメントが使用するディスク領域と、Prometheus にエクスポートされたディスク使用量の標準メトリックの確認に関する詳細を提供します。

> **注釈**
>
> 使い果たしたストレージの検出は、ディスクのサイズと使用状況を正確に報告するストレージクラスに依存しています。これは、Kind のようなシミュレートされた Kubernetes 環境、または `csi-driver-host-path` などのテストストレージクラス実装には当てはまらない場合があります。

WAL が含まれるディスクがいっぱいになり、これ以上 WAL セグメントを保存できなくなると、PostgreSQL は動作を停止します。CloudNativePG クラスターの EDB Postgres AI は、次の WAL セグメントを保存するための十分な領域があることを確認することにより、この問題を正しく検出し、復旧が複雑になる可能性のあるフェールオーバーのトリガーを回避します。

これにより、人間の管理者は根本的な原因に対処できます。

このような場合、ストレージクラスでサポートされている場合、現在最も早いアクションは次のことです。

1. フル PVC のストレージサイズを拡張する
2. `Cluster` リソースのサイズを同じ値に増加します

問題が解決され、WAL セグメント用の十分な空き領域が確保されると、ポッドが再起動し、クラスターが正常な状態になります。

ドキュメントの **ボリュームの拡張** も参照してください。

5.6 Scheduling

Kubernetes のスケジューリングは、いくつかの基準に基づいて、可能な限り最適なノードに新しいポッドを配置するプロセスです。

Kubernetes documentation を参照してください。

使用可能なすべてのポリシーを含むスケジュールの詳細については、をご覧ください。このページでは、アフィニティ、アンチアフィニティ、ノードセレクターなどの概念に精通していることを前提としています。

アフィニティ を介して、CloudNativePG Cluster クラスターのインスタンスの EDB Postgres AI をスケジュールする方法を制御できます

クラスターの定義のセクション。以下をサポートします。

- ポッドアフィニティ/アンチアフィニティ
- ノードセレクター
- 許容範囲

5.6.1 ポッドアフィニティとアンチアフィニティ

Kubernetes は、_affinity_および_anti-affinity_ルールを使用してポッドをスケジュールする場所を制御するメカニズムを提供します。これらのルールを使用すると、そこで既に実行されているワークロードに基づいて、ポッドを特定のノードでスケジュールするかどうかを指定するか、特定のノードで回避するかどうかを指定できます。この機能は、技術的に**ポッド間アフィニティ/アンチアフィニティ** と呼ばれます。

デフォルトでは、 CloudNativePG クラスターの EDB Postgres AI は、できれば別のノードでスケジュールされるようにクラスターインスタンスを構成しますが、pgBouncer インスタンスは同じノードで実行される場合があります。

例、次の `Cluster` 仕様の場合

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3
  imageName: docker.enterprisedb.com/k8s/postgresql:18.4-standard-ubi9

  affinity:
    enablePodAntiAffinity: true # Default value
    topologyKey: kubernetes.io/hostname # Default value
    podAntiAffinityType: preferred # Default value

  storage:
    size: 1Gi
```

インスタンスポッドに適用される `affinity` 構成は次のとおりです。

```yaml
affinity:
  podAntiAffinity:
    preferredDuringSchedulingIgnoredDuringExecution:
      - podAffinityTerm:
          labelSelector:
            matchExpressions:
              - key: k8s.enterprisedb.io/cluster
```

(次のページに続く)

(前のページからの続き)

```
            operator: In
            values:
              - cluster-example
          - key: k8s.enterprisedb.io/podRole
            operator: In
            values:
              - instance
        topologyKey: kubernetes.io/hostname
      weight: 100
```

この設定では、十分なリソースが利用可能であると仮定して、Kubernetes は 3 つの異なるノードにまたがって 3 ノード PostgreSQL クラスターをスケジュールすることを_prefer_します。

ポッドのアンチアフィニティの必要性

上記の設定を調整することにより、デフォルトの動作を変更できます。

たとえば、`podAntiAffinityType` を `required` に設定すると、`preferredDuringSchedulingIgnoredDuringExecution` の代わりに `requiredDuringSchedulingIgnoredDuringExecution` が適用されます。

ただし、この厳密な要件により、リソースが不十分な場合、ポッドが保留されたままになる場合があることに注意してください。これは、Cluster Autoscaler を使用する場合に特に関連します

Kubernetes クラスターでの自動ホリゾンタル スケーリング用。

Kubernetes documentation を参照してください。

トポロジの考慮事項

クラウド環境では、ポッドがノードだけではなく異なるアベイラビリティーゾーンに分散されるように、`topology.kubernetes.io/zone` を `topologyKey` として使用することを検討できます。その他のオプションについては、Well-Known Labels, Annotations, and Taints を参照してください。

アンチアフィニティポリシーの無効化

必要に応じて、`enablePodAntiAffinity` を `false` に設定することにより、オペレーター生成のアンチアフィニティポリシーを無効にできます。

カスタムルールを使用したきめの細かい制御

より正確な制御が必要なシナリオの場合、`additionalPodAffinity` および `additionalPodAntiAffinity` 構成属性を使用して、カスタムポッドアフィニティまたはアンチアフィニティルールを指定できます。これらのカスタムルールは、オペレーターが有効になっている場合は生成されたルールに追加され、オペレーター生成ルールが無効になっている場合は直接使用されます。

> **注釈**
>
> *additionalPodAntiAffinity* または `additionalPodAffinity` を使用する場合、Pod 仕様で想定されている完全な `podAntiAffinity` または `podAffinity` 構造を提供する必要があります。次の YAML 例は、どの PostgreSQL クラスターに属しているかに関係なく、ワーカーノードごとに PostgreSQL のインスタンスを 1 つだけ構成する方法を示しています。

```
additionalPodAntiAffinity:
  requiredDuringSchedulingIgnoredDuringExecution:
    - labelSelector:
        matchExpressions:
          - key: postgresql
            operator: Exists
            values: []
      topologyKey: "kubernetes.io/hostname"
```

5.6.2 nodeSelector によるノード選択

Kubernetes では、 `nodeSelector` がラベルのリストキーと値のペアとして定義を提供して、ポッドを実行できるノードを選択できます。具体的には、ノードには、スケジュールおよび実行するポッドのラベルとして、指定された各キーと値のペアが必要です。

同様に、CloudNativePG クラスターの EDB Postgres AI は、 `affinity` セクションで `nodeSelector` を定義することに同意して、これらのラベルを持つノードでのみ実行する PostgreSQL クラスターを要求できます。

5.6.3 許容範囲

Kubernetes では、 `taints` を介して `tolerations` を介して明示的に許容しないすべてのポッドをノードが反発するかどうかを指定できます `taints` を介して 。

したがって、特定のノードの `taints` と一致するワークロードに `tolerations` の適切なセットを設定することにより、Kubernetes スケジューラーは、汚染されたノードを考慮に入れて、ワークロードをスケジュールするノードを決定するようになります。`.spec.affinity.tolerations` セクションを介して、クラスターのすべてのポッドの許容値を構成できます。このセクションは、許容値の通常の Kubernetes 構文を受け入れます。

Kubernetes documentation を参照してください。

5.6.4 PostgreSQL ワークロードの分離

さまざまな方法で Kubernetes に PostgreSQL を展開できますが、運用環境では次の重要な原則に従うことをお勧めします。

- **アベイラビリティーゾーンのエクスプロイト**可能であれば、異なる AZ に PostgreSQL インスタンスを分散することにより、同じ Kubernetes クラスター内のアベイラビリティーゾーン AZ を利用します。

- **専用ワーカーノード** *PostgreSQL ワークロードのノードの予約* で詳細に説明しているように、`node-role.kubernetes.io/postgres` ラベルとテイントを介して PostgreSQL ワークロードの特定のワーカーノードを割り当てます。

セクション。

- **ノードのオーバーラップの回避**同じ PostgreSQL クラスターのインスタンスが同じノードで実行されていないことを確認します。

前のセクションで詳しく説明したように、CloudNativePG クラスターの EDB Postgres AI は、ポッドのアンチアフィニティ、ノードセレクター、および許容範囲を構成する柔軟性を提供します。

以下は、PostgreSQL `Cluster` が `postgres` ノードにデプロイされ、そのインスタンスがさまざまなノードに分散されていることを確認するためのサンプル構成です。

```
#  <snip>

affinity:
  enablePodAntiAffinity: true
  topologyKey: kubernetes.io/hostname
  podAntiAffinityType: required
  nodeSelector:
    node-role.kubernetes.io/postgres: ""
  tolerations:
    - key: node-role.kubernetes.io/postgres
      operator: Exists
      effect: NoSchedule

#  <snip>
```

そのシンプルさにもかかわらず、このセットアップは PostgreSQL ワークロードの最適な分散と分離を保証し、運用環境のパフォーマンスと信頼性の向上につながります。

5.7 Job scheduling with EDB Postgres AI for CloudNativePG

PostgreSQL 環境で定期的なメンテナンスまたは自動タスクを実行する必要がある場合、2 つの主なアプローチがあります。

1. Kubernetes Jobs または

CronJobs を使用した Kubernetes ネイティブスケジューリング。

2. 次のようなツールを使用したデータベース内スケジューリング

    - pg_cron PostgreSQL および EDB Postgres ExtendedPGE サーバーの軽量の cron ベースのジョブスケジューラー
    - DBMS_SCHEDULER 、 EDB Postgres Advanced Server で使用可能な堅牢なビルトインジョブスケジューラー)。

したがって、成功したジョブレポートは、成功したデータの永続性を保証するものではありません。

5.7.1 スケジュールされたメンテナンスタスクの例

一般的なスケジュールされた操作には次のものが含まれます。

- オフピーク時間にチャーンの多いテーブルで手動 VACUUM を実行します。
- マテリアライズドビューを更新して、分析を最新の状態に保ちます。
- トランザクションテーブルからの古いレコードのアーカイブまたはパージ。

ただし、スケジュールされたジョブは、次のようなミッションクリティカルなビジネス目的にも役立つ場合があります。

- 1 日の終わり EOD 財務調整。
- 貿易決済と清算のプロセス。
- 定期的なリスク計算または規制報告。

5.7.2 ビジネスクリティカルなジョブのベストプラクティス

ビジネスクリティカルなジョブのカスタム再試行と冪等性ロジックを実装するのはユーザーの責任です。これには次のものが含まれる場合があります。

- 挿入前の既存のレコードのチェック。
- アドバイザリーロックまたは状態マーカーを使用して、重複を回避します。
- データを破損せずにジョブを安全に再実行します。

EPAS では、特定のシナリオに autonomous transactions を活用することもできます。これらにより、作業ブロックは親トランザクションとは無関係にコミットでき、操作のロギング、監査、または補償に役立ちます。

5.8 Resource management

一般的な Kubernetes クラスターでは、ポッドは無制限のリソースで実行されます。デフォルトでは、必要なだけ CPU と RAM を使用できます。

CloudNativePG クラスターの EDB Postgres AI を使用すると、管理者は、マニフェストの `resources` セクションを介して、クラスターのポッドによるリソース使用を制御および管理できます。2 つのノブ

- `requests` 初期要件
- `limits` 最大使用量、リソースニーズが動的に増加した場合

たとえば、次のように 32MiB 128MiB までスケーラブルな RAM の初期量と 50m の CPU 100m までスケーラブルを要求できます。

```
resources:
  requests:
    memory: "32Mi"
    cpu: "50m"
```

(次のページに続く)

(前のページからの続き)

```
  limits:
    memory: "128Mi"
    cpu: "100m"
```

メモリ要求と制限はコンテナに関連付けられますが、ポッドにメモリ要求と制限があると考えると便利です。ポッドのメモリ要求は、ポッド内のすべてのコンテナのメモリ要求の合計です。

ポッドのスケジューリングは、制限ではなく要求にもとづいています。ポッドのメモリ要求を満たすだけの十分な使用可能なメモリがノードにある場合にのみ、ポッドはノードで実行されるようにスケジュールされます。

リソースごとに、コンテナを優先度の高い順に 3 つのサービス品質 QoS クラスに分割します。

- **保証**
- *Burstable*
- **ベストエフォート**

詳細については、 Configure Quality of Service for Pods を参照してください。

Kubernetes ドキュメントのセクション。

PostgreSQL ワークロードの場合、「保証」QoS を設定することをお勧めします。

> **注釈**
>
> サービス品質が「保証」に設定されている場合、CloudNativePG クラスターの EDB Postgres AI は、PostgreSQL documentation に従って、 *postmaster* プロセスの `PG_OOM_ADJUST_VALUE` を `0` に設定します。これにより、 *postmaster* は `-997` の低いアウトオブメモリ OOM スコアを維持できますが、その子プロセスは `0` の OOM スコア調整で実行されます。その結果、OOM キラーがトリガーされると、*postmaster* より前に子プロセスを終了します。この動作は、PostgreSQL インスタンスをできるだけ長く生かし続けるのに役立ち、エビクションが発生した場合のクリーンシャットダウン手順を有効にします。

Kubernetes のリソース関連の問題を回避するには、クラスターの作成中に「リソース不足」処理のベストプラクティスを参照できます。

- マニフェストファイルのリソースセクションでメモリと CPU の必要な値を指定します。この方法により、`OOM Killed` および `CPU throttle` または実行中のインスタンスに関するその他のリソース関連の問題を回避できます。
- クラスターのポッドを「保証」QoS クラスに割り当てるには、メモリと CPU の両方の制限と要求を同じ値に設定する必要があります。
- ポッドリソースと一貫して必要な PostgreSQL メモリパラメーターを指定します VM または物理マシンのシナリオで行うように、以下を参照してください。
- nodeSelector を使用して、専用ノードにデータベースサーバーポッドをセットアップします。API リファレンスページの *“affinityconfiguration”* リソースの「nodeSelector」および「tolerations」フィールドを参照してください。

次のマニフェストの例を参照できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: postgresql-resources
spec:
  instances: 3

  postgresql:
    parameters:
      shared_buffers: "256MB"

  resources:
    requests:
      memory: "1024Mi"
      cpu: 1
    limits:
      memory: "1024Mi"
      cpu: 1

  storage:
    size: 1Gi
```

上記の例では、 256MB の値で shared_buffers パラメーターを指定しました。つまり、データをキャッシュするために PostgreSQL サーバーに専用のメモリの量です。このパラメーターのデフォルト値は定義されていない場合の 128MB です。

shared_buffers の適切な開始値は、システム内のメモリの 25%です。例えば shared_buffers が 256 MB の場合、コンテナメモリサイズの推奨値は 1 GB です。これは、ポッド内ですべてのコンテナが合計 1 GB のメモリを持つことを意味し、Kubernetes が常に保持し、コンテナができることを意味します予想通りに動作します。詳細については、Resource Consumption を参照してください。

PostgreSQL ドキュメントのセクション。

Kubernetes ドキュメントのページ。

5.8.1 垂直ポッドオートスケーラー VPA との統合

Cluster CRD は、scale サブリソースとそのインスタンスポッドのラベルセレクターを公開します。これにより、Cluster が Vertical Pod Autoscaler の有効な targetRef になります

VPA)、VPA はインスタンスポッドを観察し、それらの CPU とメモリの推奨事項を発行できます。

VPA は **推奨専用** モード updatePolicy.updateMode: Off で実行することをお勧めします。このモードでは、VPA はその推奨事項のみをレポートし、オペレーターは通常のマニフェストの更新を介して Cluster の spec.resources に適用できます。 CloudNativePG クラスターの EDB Postgres AI は、通常のスイッチオーバー対応手順を使用してインスタンスのローリング更新を実行します。VPA はコンテナごとに推奨

事項を生成します。 spec.resources が構成するものであるため、 postgres コンテナ用のものを使用します。

Cluster をターゲットとする例

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: cluster-example-vpa
spec:
  targetRef:
    apiVersion: postgresql.k8s.enterprisedb.io/v1
    kind: Cluster
    name: cluster-example
  updatePolicy:
    updateMode: "Off"
```

警告

CloudNativePG クラスター管理の `Cluster` の EDB Postgres AI に対して *updateMode: Auto* 、*Recreate* 、または `Initial` を使用しないでください。オペレーターはポッド仕様を所有し、*Cluster* の `spec.resources` を真実のソースとして扱います。実行中のポッドに VPA が書き込むリソースを採用しないため、ライブポッドと宣言された `spec.resources` はサイレントに分岐します。宣言したリソースたとえば、オペレーターがメモリ要求に対して検証した `shared_buffers` 値に対してサイズを設定した調整は、ポッドの実際の制限と一致しなくなりました。また、VPA は、Kubernetes エビクション API を介してポッドをエビクトし、オペレーターのスイッチオーバー認識シークエンシングをバイパスします。デフォルトのプライマリ `PodDisruptionBudget` は、プライマリのエビクションをブロックするため、VPA は完了する代わりにストールします。代わりに、VPA 推奨事項を `Cluster` マニフェストに手動で適用します。

scale サブリソースは spec.instances も公開するため、Horizontal Pod Autoscaler

HPA は、Cluster のインスタンスの数を技術的に変更できます。実際には、これはいくつかの理由から PostgreSQL クラスターには**お勧めできません** 。

Cluster のスケーリングは、スタンバイレプリカのみを追加または削除します。プライマリの書き込み負荷が軽減されることはありません。スケールサブリソースを介して公開されるセレクターは、反対のワークロードプロファイル書き込み多いプライマリ、読み取り専用レプリカを持つプライマリおよびすべてのレプリカと一致するため、HPA が CPU またはメモリメトリックから計算するポッドごとの平均は意味がありません。それらのいずれかの信号。レプリカを追加すると、ホットプライマリに対処せずに、平均をさらに薄めるだけです。

HPA は、CloudNativePG クラスターの EDB Postgres AI spec.instances に対する独自の制約も認識しません。同期レプリカを構成する場合、maxSyncReplicas + 1 インスタンスより下のスケールダウンは検証 Webhook によって拒否され、HPA は満足できない値を再試行し続けます。それでも HPA から spec.instances を駆動する場合は、実際にリードレプリカの負荷を反映するカスタムメトリックに基づいて、minReplicas を同期レプリカのフロアより上に設定します。最初に、レプリケーションとクォーラムへの影響を注意深く確認します。

5.9 Failure Modes

> **注釈**
>
> CloudNativePG クラスターの EDB Postgres AI の以前のバージョンでは、このページには特定の障害シナリオが含まれていました。これらは主に標準の Kubernetes の動作に従うため、コンテンツを合理化して、基になる Kubernetes スタックに属し、CloudNativePG クラスターの EDB Postgres AI に固有ではない情報の重複を回避しました。

CloudNativePG クラスターの EDB Postgres AI は、自己修復と高可用性のための標準の Kubernetes 原則に準拠しています。ストレージクラス、PVC、ノード、ポッドなどのコア Kubernetes 概念に精通していることを前提としています。CloudNativePG クラスター固有の EDB Postgres AI の詳細については、スタートアップ、ライブネス、および準備状況プローブをカバーする *Postgres Instance Manager* および以下の **自己修復のための自動フェイルオーバー** セクションを参照してください。

5.9.1 自己修復

一次障害

プライマリポッドに障害が発生した場合

- オペレーターは、レプリケーションラグが最も低い最新のスタンバイをプロモートします。
- `-rw` サービスが更新されて、新しいプライマリをポイントします。
- 障害が発生した Pod は、`-r` および`-rw` サービスから削除されます。
- スタンバイポッドは、新しいプライマリから複製を開始します。
- 前者のプライマリは `pg_rewind` を使用して、PVC が利用可能な場合を再同期します。それ以外の場合、新しいプライマリのバックアップから新しいスタンバイが作成されます。

スタンバイ障害

スタンバイポッドに障害が発生した場合

- `-r` および`-ro` サービスから削除されます。
- ポッドは、PVC がある場合はそれを使用して再起動されます。それ以外の場合、新しいポッドは、現在のプライマリのバックアップから作成されます。
- 準備が整うと、Pod は`-r` および`-ro` サービスに再追加されます。

5.9.2 手動介入

自動復旧の対象とならない障害シナリオの場合、手動介入が必要になる場合があります。

リコンシリエーションの無効化

k8s.enterprisedb.io/reconciliationLoop アノテーションを使用すると、CloudNativePG クラスターリソースの EDB Postgres AI の調整ループを一時的に無効にすることができます。"disabled" に設定されている場合、オペレーターはアノテーション付きリソースの更新の処理を停止し、自動変更または自己修復アクションを防ぎます。

この注釈は **細心の注意を払って** 、緊急操作中にのみ使用してください。

> **警告**
>
> このアノテーションは、問題が解決されたらすぐに削除する必要があります。そのままにすると、オペレーターはアノテーション付きリソースを管理できません。クラスターでは、これには自己修復アクションとフェイルオーバーが含まれます。

- **クラスター** PostgreSQL クラスターのリコンシリエーションを無効にします
- **Backup** バックアップ操作のリコンシリエーションを無効にします

使用例

```
metadata:
  name: cluster-example-no-reconcile
  annotations:
    k8s.enterprisedb.io/reconciliationLoop: "disabled"
spec:
  # ...
```

5.10 Rolling updates

オペレーターを使用すると、アプリケーションの実行を継続しながら、クラスターで使用される PostgreSQL バージョンを変更できます。

ローリングアップグレードは、次の場合にトリガーされます。

- クラスター仕様の imageName 属性を変更する場合。
- クラスター仕様の.spec.postgresql.extensions スタンザの拡張イメージのリストを変更する場合。
- イメージカタログを使用する は、クラスターが使用するメジャーバージョンの新しいイメージで更新されます。
- PostgreSQL 構成の変更を適用するには、再起動が必要です。
- Cluster .spec.resources 値を変更します。
- オペレーターが更新され、ポッドが最新のインスタンスマネージャーを実行します *in-place updates are enabled* の場合

ローリングアップグレード中に、オペレーターは、すべてのレプリカを一度に 1 つのポッドずつ、最も高いシリアルを持つものから始めてアップグレードします。

プライマリは常にアップグレードされる最後のノードです。

ローリング更新は構成可能で、完全に自動化する unsupervised か、人間の介入を必要とする supervised のいずれかです。

アップグレードでは、データの再クローンを作成せずに、CloudNativePG クラスターの EDB Postgres AI の ID を維持します。ポッドは削除され、必要に応じて同じ PVC と新しいイメージを使用して再度作成されます。

ローリング更新手順中に、各サービスのエンドポイントが移動してクラスターのステータスを反映するため、アプリケーションは更新中のノードを無視できます。

5.10.1 自動更新 unsupervised

`primaryUpdateStrategy` が `unsupervised` に設定されている場合、ローリング更新プロセスは Kubernetes によって管理され、完全に自動化されます。レプリカがアップグレードされると、選択した `primaryUpdateMethod` オペレーションがプライマリで実行されます。教師なし更新はデフォルトの動作です。

`primaryUpdateMethod` オプションは、次のいずれかの値を受け入れます。

- `restart` 可能であれば、プライマリインスタンスが実行されているポッドの自動再起動を実行します。それ以外の場合、再起動要求は無視され、スイッチオーバーが発行されます。これはデフォルトの動作です。
- `switchover` スイッチオーバー操作が自動的に実行され、最もアライメントされたレプリカを新しいターゲットプライマリとして設定し、以前のプライマリポッドをシャットダウンします。

> **警告**
>
> *primaryUpdateMethod* が `switchover` に設定されている場合、イメージ名と PostgreSQL 構成パラメーターを同時に変更することはできません。オペレーターは、このような更新を検証エラーで拒否します。イメージと構成の両方を更新する必要がある場合は、これらの変更を順番に実行する必要があります。最初にイメージを更新し、ローリング更新が完了するのを待って、構成を更新します。その逆も可能です。この制限は、新しいイメージバージョンに関連付けられた構成の変更が、スイッチオーバープロセス中に古いイメージを実行しているポッドに適用されると、PostgreSQL に障害を発生させる可能性があるために存在します。

データベースの実際のワークロード、**リリース候補の目的** および *RTO* に関する要件、PostgreSQL アーキテクチャが共有されているか、何も共有されていないなどのいくつかの要素に依存するため、update 方法に万能の構成はありません。で。

実際、PostgreSQL はプライマリ/スタンバイアーキテクチャのデータベース管理システムであるため、更新プロセスは必然的にアプリケーションのダウンタイムを生成します。コンテキストに合わせて考慮する必要がある重要な側面の 1 つは、ポッドが新しい PostgreSQL コンテナイメージをダウンロードするのにかかる時間です。これは、Kubernetes クラスターの設定と仕様によって異なります。`switchover` メソッドは、昇格したインスタンスがコンテナのターゲットイメージバージョンを既に実行していることを確認します。代わりに `restart` メソッドでは、プライマリポッドがシャットダウンされた後、元のレジストリからイメージをダウンロードする必要がある場合があります。データベースにとって、ローリング更新プロ

シージャの一部として restart または switchover を使用することが最善であるかどうかは、あなたが判断する必要があります。

5.10.2 手動更新 supervised

primaryUpdateStrategy が supervised に設定されている場合、ローリング更新プロセスは、すべてのレプリカがアップグレードされた直後に一時停止されます。

このフェーズは、手動スイッチオーバーまたはインプレースリスタートのいずれかでのみ完了できます。イメージのアップグレードはインプレース再起動では適用できないため、このような場合にはスイッチオーバーが必要であることに注意してください。

次の方法でスイッチオーバーをトリガーできます。

```
kubectl cnp promote [cluster] [new_primary]
```

次の方法で再起動をトリガーできます。

```
kubectl cnp restart [cluster] [current_primary]
```

詳細については、 *CNPG-I* を参照してください。

5.11 Replication

物理的レプリケーションは PostgreSQL の強みの 1 つであり、世界最大の組織の一部がビジネス継続性のコンテキストでのデータの管理にそれを選択した理由の 1 つです。物理レプリケーションは、主に高可用性を実現するために使用されますが、読み取り専用ワークロードのスケールアウトと、プライマリから一部の作業をオフロードすることもできます。

5.11.1 アプリケーションレベルのレプリケーション

PostgreSQL のレプリケーション機能に長年貢献してきましたが、私たちはネイティブ物理レプリケーションテクノロジーに基づいて CloudNativePG クラスターの EDB Postgres AI の高可用性を構築し、Kubernetes API に直接統合することにしました。

Kubernetes の用語では、これは_ストレージレベルのレプリケーション_とは対照的に、**アプリケーションレベルのレプリケーション** と呼ばれます。

5.11.2 非常に成熟したテクノロジー

PostgreSQL には、プライマリインスタンスから 1 つ以上のレプリカにデータをレプリケートするための非常に堅牢で成熟したネイティブフレームワークがあり、WAL ライトアヘッドログに継続的に保存されるトランザクションの変更の概念を中心に構築されています。

クラッシュリカバリーとポイントインタイムリカバリーテクノロジーの進化として始まった物理的レプリケーションは、継続的リカバリーでのプライマリからウォームスタンバイへの WAL シッピングを介して PostgreSQL 8.22006 で最初に導入されました。

PostgreSQL 9.0 2010 は、_ホットスタンバイ_を介した WAL ストリーミングと読み取り専用レプリカを導入しました。2011 年、PostgreSQL 9.1 はトランザクションレベルで同期レプリケーションをもたらし、リリース候補の目的=0 クラスターをサポートしました。カスケードレプリケーションは PostgreSQL 9.22012 で追加されました。 宣言的なパブリケーションおよびサブスクリプションオブジェクトを介した論理レプリケーション の基礎は PostgreSQL 9.42014 で確立され、バージョン 102017 では、オリジンから宛先にデータをレプリケートするためのパブリッシャー/サブスクライバーパターンのネイティブサポートを導入しました。以下の表は、これらのマイルストーンをまとめたものです。

Version	Year	Feature
8.2	2006	Warm Standby with WAL shipping
9.0	2010	Hot Standby and physical streaming replication
9.1	2011	Synchronous replication (priority-based)
9.2	2012	Cascading replication
9.4	2014	Foundations of logical replication
10	2017	Logical publisher/subscriber and quorum-based synchronous replication

この表は、主要な PostgreSQL レプリケーション機能とその各バージョンを強調表示しています。

5.11.3 ストリーミングレプリケーションのサポート

現時点では、CloudNativePG クラスターの EDB Postgres AI は、spec で提供された instances の数に基づいて、クラスター内の物理的ストリーミングレプリカを宣言的方法でネイティブかつ透過的に管理します。

```
replicas = instances - 1 (where  instances > 0)
```

クラスターの初期化の直後、オペレーターは次のように streaming_replica というユーザーを作成します。

```
CREATE USER streaming_replica WITH REPLICATION;
- - NOSUPERUSER INHERIT NOCREATEROLE NOCREATEDB NOBYPASSRLS
```

すぐに使える、オペレーターは、暗号化されたチャネルを介してクラスター内のストリーミングレプリケーションを自動的に設定し、streaming_replica ユーザーの TLS クライアント証明書認証を適用します。pg_hba.conf からの次の抜粋で強調表示されています。

```
#  Require client certificate authentication for the streaming_replica user

hostssl postgres streaming_replica all cert map=cnp_streaming_replica
hostssl replication streaming_replica all cert map=cnp_streaming_replica
```

ドキュメントで。

構成されている場合、オペレーターは HA クラスター内のすべてのレプリカのレプリケーションスロットを管理し、各スタンバイに必要な WAL ファイルが、フェールオーバーまたはスイッチオーバーの後でもプライマリのストレージに保持されるようにします。

以下。

継続的なバックアップ統合

クラスターで継続的バックアップが構成されている場合、CloudNativePG クラスターの EDB Postgres AI は、継続的リカバリー時に `restore_command` を利用するようにレプリカを透過的に構成します。その結果、PostgreSQL は、ストリーミングレプリケーションを介した WAL のプルが失敗するときは常に、フォールバックオプションとして WAL アーカイブを使用できます。

5.11.4 同期レプリケーション

CloudNativePG クラスターの EDB Postgres AI は、両方の quorum-based and priority-based synchronous replication for PostgreSQL をサポートしています。

> **警告**
>
> デフォルトでは、トランザクションのコミット中に WAL レプリケーションに必要な数のスタンバイノードを使用できない場合、同期レプリケーションは書き込み操作を一時停止します。この動作はデータの耐久性を優先し、PostgreSQL DBA のベストプラクティスに沿っています。ただし、セットアップでの厳密なデータの耐久性よりも自己修復の方が優先される場合、この設定を調整できます。この動作の管理の詳細については、 **データの耐久性と同期レプリケーション** を参照してください。

セクション。

同期レプリケーションと並行して使用して、フェールオーバーイベント中のデータの耐久性と安全性を向上させることができます。

`synchronous_standby_names` オプションの直接構成は許可されていません。ただし、CloudNativePG クラスターの EDB Postgres AI は、このオプションにローカルポッドの名前を自動的に入力しますが、カスタマイズで `Cluster` リソースを超えて同期レプリケーションを拡張することもできます。これは、*.spec.postgresql.synchronous* を介して実現できます。

同期レプリケーションはデフォルトで無効になっています。 `synchronous` スタンザは定義されていません。定義すると、2 つのオプションが必須です。

- `method` any クォーラムまたは `first` プライオリティ
- `number` トランザクションが応答を待機する必要がある同期スタンバイサーバーの数

クォーラムベースの同期レプリケーション

PostgreSQL では、クォーラムベースの同期レプリケーションにより、トランザクションのコミットは、WAL レコードが指定された数のスタンバイにレプリケートされるまで待機します。これを有効にするには、`method` を `any` に設定します。

このレプリケーション方法は、CloudNativePG Cluster クラスターの EDB Postgres AI の最も一般的なセットアップです。

例

次の例は、3 つのインスタンスを使用した一般的な cluster-example 構成に基づいて、少なくとも 1 つのインスタンスを使用してクォーラムベースの同期レプリケーションを設定します。

```
postgresql:
  synchronous:
    method: any
    number: 1
```

この構成では、CloudNativePG クラスターの EDB Postgres AI は、次のように synchronous_standby_names のコンテンツを自動的に設定します。

```
ANY 1 (cluster-example-2, cluster-example-3, cluster-example-1)
```

非推奨の同期レプリケーション実装からの移行

このセクションでは、非推奨のクォーラムベースの同期レプリケーション形式から、CloudNativePG クラスターの EDB Postgres AI の新しいより堅牢な実装に移行する方法について概要を説明します。

次のマニフェストを考えると

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: angus
spec:
  instances: 3
  minSyncReplicas: 1
  maxSyncReplicas: 1

  storage:
    size: 1Gi
```

次のとおりに、新しい形式に更新できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: angus
spec:
  instances: 3

  storage:
    size: 1Gi

  postgresql:
```

(次のページに続く)

(前のページからの続き)

```
  synchronous:
    method: any
    number: 1
    dataDurability: required
```

厳密なデータの耐久性よりも自己修復を優先するには、代わりに dataDurability を preferred に設定します。

優先度ベースの同期レプリケーション

PostgreSQL の優先度ベースの同期レプリケーションは、WAL レコードが、優先度に基づいて選択された要求された数の同期スタンバイにレプリケートされるまで、トランザクションコミットを待機させます。synchronous_standby_names オプションで前にリストされたスタンバイには、より高い優先順位が与えられ、同期とみなされます。現在の同期スタンバイが切断されると、次に優先度の高いスタンバイにすぐに置き換えられます。この方法を使用するには、method を first に設定します。

synchronous_standby_names コンテンツの制御

デフォルトでは、CloudNativePG クラスターの EDB Postgres AI は、Cluster リソースのローカルポッドの名前を synchronous_standby_names に設定し、PostgreSQL クラスター内の同期レプリケーションを保証します。.spec.postgresql.synchronous スタンザの次のオプショナルパラメーターを使用して、要件とレプリケーション方法クォーラムまたは優先順位に基づいて synchronous_standby_names のコンテンツをカスタマイズできます。

- maxStandbyNamesFromCluster PostgreSQL の synchronous_standby_names オプションに自動的に含めることができるローカル Cluster オブジェクトのポッド名の最大数。
- standbyNamesPre オペレーターによって自動的にリストされたローカルポッド名のリストの前に追加されるスタンバイ名のリスト特に application_name 。
- standbyNamesPost オペレーターによって自動的にリストされたローカルポッド名のリストに追加されるスタンバイ名のリスト特に application_name 。

> **警告**
>
> *standbyNamesPre* および `standbyNamesPost` の名前が正しいことを確認する責任があります。 CloudNativePG クラスターの EDB Postgres AI は、ここにリストされている `application_name` でスタンバイを管理し、高可用性を保証することを想定しています。エントリが間違っていると、PostgreSQL データベースのアップタイムが危険にさらされる可能性があります。

例

次に、いくつかの例を示します。すべては、3 つのインスタンスを持つ cluster-example に基づいています。

設定すると

```yaml
postgresql:
  synchronous:
    method: any
    number: 1
    maxStandbyNamesFromCluster: 1
    standbyNamesPre:
      - angus
```

synchronous_standby_names の内容は次のとおりです。

```
ANY 1 (angus, cluster-example-2)
```

設定すると

```yaml
postgresql:
  synchronous:
    method: any
    number: 1
    maxStandbyNamesFromCluster: 0
    standbyNamesPre:
      - angus
      - malcolm
```

synchronous_standby_names の内容は次のとおりです。

```
ANY 1 (angus, malcolm)
```

設定すると

```yaml
postgresql:
  synchronous:
    method: first
    number: 2
    maxStandbyNamesFromCluster: 1
    standbyNamesPre:
      - angus
    standbyNamesPost:
      - malcolm
```

synchronous_standby_names オプションは次のようになります。

```
FIRST 2 (angus, cluster-example-2, malcolm)
```

データの耐久性と同期レプリケーション

.spec.postgresql.synchronous スタンザの dataDurability オプションは、同期レプリケーションのデータの安全性と可用性間のトレードオフを制御します。required または preferred に設定できます。指定しない場合、デフォルトは required になります。

必要なデータの耐久性

dataDurability が required に設定されている場合、PostgreSQL は、WAL 書き込み先行ログレコードが指定された数の同期スタンバイにレプリケートされた後にコミットされたトランザクションのみを考慮します。この設定では、可用性よりデータの安全性が優先されます。つまり、必要な数の同期スタンバイが使用できない場合、書き込み操作は一時停止します。これにより、ゼロデータ損失 RPO=0 が保証されますが、ネットワークの中断またはスタンバイ障害が発生するとデータベースの可用性が低下する場合があります。

同期スタンバイは次の優先順位で選択されます。

1. 正常インスタンス
2. 異常なインスタンス
3. プライマリー

リストはこの値が設定されている場合は maxStandbyNamesFromCluster に基づいて切り捨てられ、正常なインスタンスを優先し、synchronous_standby_names が設定されていることを確認します。

例

次の例を考えます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: foo
spec:
  instances: 3
  postgresql:
    synchronous:
      method: any
      number: 1
      dataDurability: required
```

1. 初期状態。 synchronous_standby_names の内容は次のとおりです。

```
ANY 1 ("foo-2","foo-3","foo-1")
```

2. foo-2 は使用できなくなります。優先的にプッシュバックされます。

```
ANY 1 ("foo-3","foo-2","foo-1")
```

3. foo-3 も使用できなくなります。リストには正常なスタンバイは含まれません。

```
ANY 1 ("foo-2","foo-3","foo-1")
```

この時点で、少なくとも 1 つのスタンバイが再び利用可能になるまで、書き込み操作は許可されません。

4. スタンバイが再び利用できるようになると、synchronous_standby_names は初期状態に戻ります。

優先データの耐久性

dataDurability が preferred に設定されている場合、必要な同期インスタンスの数は、使用可能なスタンバイの数に基づいて調整されます。PostgreSQL は、指定された数の同期スタンバイに WAL レコードをレプリケートしようとしますが、要求された数よりも少ないスタンバイが使用可能な場合でも、書き込み操作は続行されます。

この設定は、データの安全性と可用性のバランスをとり、アプリケーションが一時的にスタンバイが使用不可になっているときに書き込みを継続できるようにします。したがって、_self-healing モード_としても知られています。

> **警告**
>
> このモードでは、すべてのスタンバイが使用不可になるとデータが損失する可能性があります。

preferred データ持続性では、 **正常なレプリカのみ**が synchronous_standby_names に含まれます。

例

次の例を考えます。デモンストレーションでは、5 つのインスタンスと 2 つの同期スタンバイを持つ bar という名前のクラスターを使用します。

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: bar
spec:
  instances: 5
  postgresql:
    synchronous:
      method: any
      number: 2
      dataDurability: preferred
```

1. 初期状態。 synchronous_standby_names の内容は次のとおりです。

```
ANY 2 ("bar-2","bar-3", "bar-4", "bar-5")
```

2. bar-2 および bar-3 は使用できなくなります。これらはリストから削除されます。

```
ANY 2 ("bar-4", "bar-5")
```

3. bar-4 も使用できなくなります。リストから削除されます。使用可能なスタンバイの数は要求された数より少ないため、要求された量は削減されます。

```
ANY 1 ("bar-5")
```

4. bar-5 も使用できなくなります。 synchronous_standby_names は空になり、同期レプリケーションを完全に無効にします。書き込み操作は継続されますが、プライマリ障害が発生した場合に潜在的なデータ損失のリスクが伴います。

5. レプリカが戻ると、synchronous_standby_names は初期状態に戻ります。

5.11.5 同期レプリケーション非推奨

> **警告**
>
> CloudNativePG クラスター 1.24 の EDB Postgres AI より前は、クォーラムベースの同期レプリケーション実装のみがサポートされていました。この方法は現在非推奨ですが、すぐに削除されることはありません。新しい方法は、自己修復よりもデータの耐久性を優先し、優先度ベースの同期レプリケーションや `synchronous_standby_names` オプションの完全な制御など、より堅牢な機能を提供します。前の段落で説明したように、同期レプリケーションの新しい構成方法に徐々に移行することをお勧めします。

CloudNativePG クラスターの EDB Postgres AI は、 minSyncReplicas および maxSyncReplicas と呼ばれる 2 つの構成オプションを介して**クォーラムベースの同期ストリーミングレプリケーション* の構成をサポートします。これらは、任意の時点で使用可能な同期スタンバイレプリカの予想される最小数と最大数です。自己修復の目的で、オペレーターは常にこれらの 2 つの値を使用可能なレプリカの数と比較して、クォーラムを決定します。

同期レプリケーションはデフォルトで無効になっています minSyncReplicas および maxSyncReplicas は定義されていません。 minSyncReplicas と maxSyncReplicas の両方が設定されている場合、CloudNativePG クラスターの EDB Postgres AI は、PostgreSQL の synchronous_standby_names オプションを次の値に自動的に更新します。

```
ANY q (pod1, pod2, ...)
```

そこで

- q は、演算子によって自動的に計算された整数です。1 ≤ minSyncReplicas ≤ q ≤ maxSyncReplicas ≤ readyReplicas
- pod1, pod2, ... は、クラスター内のすべての PostgreSQL ポッドのリストです

> **警告**
>
> 自己修復機能を提供するために、オペレーターは `minSyncReplicas` そのような値が現在使用可能なレプリカの数より大きい場合は無視できます。 *readyReplicas* が `0` の場合、同期レプリケーションは自

動的に無効になります。

PostgreSQL documentation で述べているように、_method ANY は、クォーラムベースの同期レプリケーションを指定し、WAL レコードが少なくとも list_内の要求された数の同期スタンバイにレプリケートされるまで、トランザクションコミットを待機させます。

同期レプリケーションのノードを選択します

CloudNativePG クラスターの EDB Postgres AI を使用すると、PGDATA と Postgres ポッドを保持する PVC が存在するノードラベルに基づいて、アンチアフィニティルールを介してクォーラムベースの同期レプリケーションセットに参加する資格のある PostgreSQL インスタンスを選択できます。

警告

.spec.postgresql.syncReplicaElectionConstraint オプションは、同期レプリケーションの古い実装にのみ適用されます **同期レプリケーション非推奨** を参照してください。

この機能のユースケースの例として単一の同期レプリカを持つクラスターでは、同期レプリカがプライマリインスタンスとは別のアベイラビリティーゾーンにあることを確認できます。通常は `topology.kubernetes.io/zone` label on a node で識別されます。これにより、単一のアベイラビリティーゾーンで停止が発生した場合のクラスターのロバスト性、特に目標復旧ポイント**リリース候補の目的** の点で向上します。

アンチアフィニティのアイデアは、クォーラムに参加する同期レプリカが、選択したラベルこの場合、アベイラビリティーゾーンラベルの異なる値を持つノードで実行されているポッドから選択されるようにすることです。プライマリが現在存在するノード実行中。このような基準に一致するノードがない場合、レプリカは同期レプリケーションの対象となります。

次の例は、`.spec.postgresql` 内の `syncReplicaElectionConstraint` セクションを介してこれを行う方法を示しています。`nodeLabelsAntiAffinity` では、現在のプライマリと、アベイラビリティーゾーンラベルの値が異なるノードにあるレプリカの間で、オペレーターによって同期レプリケーションが動的に構成されることを確認するために、評価する必要があるノードラベルを指定できます。プライマリが存在するノード

```
spec:
  instances: 3
  postgresql:
    syncReplicaElectionConstraint:
      enabled: true
      nodeLabelsAntiAffinity:
        - topology.kubernetes.io/zone
```

ご想像のとおり、アベイラビリティーゾーンは一例にすぎませんが、ストレージ、CPU、メモリなど、ノードを説明する他のラベルに基づいてこの動作をカスタマイズできます。

5.11.6 レプリケーションスロット

Replication slots

は、9.4 で導入されたネイティブの PostgreSQL 機能で、接続されているすべてのストリーミングレプリケーションクライアントが WAL セグメントを受信するまで、プライマリが WAL セグメントを削除しないこと、およびプライマリが次の場合でもリカバリ競合を発生させる可能性のある行を削除しないことを保証する自動方法を提供しますスタンバイは一時的に切断されています。

レプリケーションスロットは、それを作成したインスタンスにのみ存在し、PostgreSQL はスタンバイサーバーでレプリケートしません。その結果、フェールオーバーまたはスイッチオーバー後、新しいプライマリには、古いプライマリのレプリケーションスロットが含まれません。これにより、古いプライマリに接続され、スロットを失ったストリーミングレプリケーションクライアントに問題が発生する可能性があります。

CloudNativePG クラスターの EDB Postgres AI は、プライマリから各スタンバイに物理レプリケーションスロットのコンテンツを同期するターンキーソリューションを提供し、2 つのユースケースに対応します。

- Postgres クラスターの高可用性のために自動的に作成されるレプリケーションスロット**高可用性のためのレプリケーションスロット** を参照してください

詳細についてはこちら

- プライマリで作成された **ユーザー定義レプリケーションスロット**

高可用性のためのレプリケーションスロット

CloudNativePG クラスターの EDB Postgres AI は、高可用性クラスターから始めるクラスター管理レプリケーションスロットの概念を導入することにより、このギャップを埋めます。この機能は、プライマリとスタンバイの両方で、高可用性クラスター内の各ホットスタンバイレプリカの物理レプリケーションスロットを自動的に管理します。

CloudNativePG クラスターの EDB Postgres AI では、次の用語を使用します。

- **プライマリ HA スロット**ライフサイクルがクラスターの現在のプライマリによって完全に管理され、目的がストリーミングレプリケーションで特定のスタンバイにマッピングすることである物理レプリケーションスロット。このようなスロットは、プライマリにのみ存在します。
- **スタンバイ HA スロット**スタンバイの物理レプリケーションスロット。そのライフサイクルは、プライマリの `pg_replication_slots` ビューの内容に基づいて、クラスター内の別のスタンバイによって完全に管理され、`pg_replication_slot_advance()` を使用して定期的に更新されます。

この機能はデフォルトで有効になっており、構成から無効にできます。詳細については、*ReplicationSlotsConfiguration* を参照してください。以下に、メインオプションの簡単な説明を示します。

`.spec.replicationSlots.highAvailability.enabled`

`true` の場合、機能は有効になっています `true` がデフォルトです

`.spec.replicationSlots.highAvailability.slotPrefix`

この機能のオペレーターが管理するレプリケーションスロットを識別するプレフィックスデフォルト_cnp_

`.spec.replicationSlots.updateInterval`

スタンバイがレプリケーションスロットのローカルコピーの位置を現在のプライマリ上の位置と同期する頻度。秒単位で表現されますデフォルト 30

お勧めしませんが、別の動作が必要な場合は、上記のオプションをカスタマイズできます。

たとえば、次のマニフェストは、レプリケーションスロットを無効にしたクラスターを作成します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3
  # Disable replication slots for HA in the cluster
  replicationSlots:
    highAvailability:
      enabled: false

  storage:
    size: 1Gi
```

ユーザー定義レプリケーションスロット

CloudNativePG クラスターの EDB Postgres AI は、物理レプリケーションスロットを宣言的に定義する方法をサポートしていませんが、create your own slots via SQL は可能です。

CloudNativePG クラスターの EDB Postgres AI は、上記で説明した HA レプリケーションスロットの場合と同様に、プライマリとスタンバイ間のユーザー管理の物理レプリケーションスロットの同期を管理できます。唯一の違いは、レプリケーションスロットを作成する必要があることです。

この機能はデフォルトで有効になっていますが、レプリケーションスロットが同期されることを意味しますが、synchronizeReplicas スタンザを介して無効にするか、正規表現を使用して一部のスロットを除外するなど、その動作をさらにカスタマイズできます。例

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3
  replicationSlots:
    synchronizeReplicas:
      enabled: true
      excludePatterns:
        - "^foo"
```

詳細については、*ReplicationSlotsConfiguration* を参照してください。以下に、メインオプションの簡単な説明を示します。

`.spec.replicationSlots.synchronizeReplicas.enabled`

true または指定されない場合、プライマリのすべてのユーザー定義のレプリケーションスロットは、各スタンバイで同期されます。false に変更した場合、オペレーターは、各スタンバイで自分自身が以前に作成したレプリケーションスロットを削除します。

`.spec.replicationSlots.synchronizeReplicas.excludePatterns`

同期から除外するユーザー定義のレプリケーションスロットの名前と一致する正規表現パターンのリスト。これは、命名規則に基づいて特定のスロットを除外する場合に役立ちます。

> **警告**
>
> この機能を利用するユーザーは、ユーザー定義のレプリケーションスロットを注意深く監視して、運用要件を満たし、フェイルオーバープロセスを妨げないことを確認する必要があります。

同期周波数

この例のように、スタンバイがプライマリで `pg_replication_slots` ビューを照会し、レプリケーションスロットのローカルコピーを更新する頻度を制御することもできます。

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3
  # Reduce the frequency of standby HA slots updates to once every 5 minutes
  replicationSlots:
    updateInterval: 300

  storage:
    size: 1Gi
```

論理デコードスロットの同期

CloudNativePG クラスターの EDB Postgres AI は、高可用性クラスター内のすべてのノードで論理デコーディングレプリケーションスロットを同期でき、フェールオーバーまたはスイッチオーバー後の論理レプリケーションのシームレスな継続を保証します。この機能はデフォルトでは無効になっており、有効にするには 2 つの手順が必要です。

最初のステップは、論理デコードスロットの同期を有効にすることです。

```yaml
#  ...

replicationSlots:
  highAvailability:
    synchronizeLogicalDecoding: true
```

2 番目のステップには、PostgreSQL パラメーターの構成が含まれます。以下で説明するように、必要な構成は PostgreSQL バージョンによって異なります。

有効にすると、オペレーターはフェールオーバーとスイッチオーバー中に論理デコードスロット状態を自動的に管理し、スロットの無効化を防止し、論理レプリケーションクライアントのデータ損失を回避します。

PostgreSQL 17 以降での動作

PostgreSQL 17 以降の場合、CloudNativePG クラスターの EDB Postgres AI は

synchronized_standby_slots を透過的に管理します。

PostgreSQL 構成で `sync_replication_slots` と `hot_standby_feedback` の両方を有効にする必要があります。

```
#  ...

postgresql:
  parameters:
    # ...
    hot_standby_feedback: "on"
    sync_replication_slots: "on"
```

さらに、 `failover` オプションを有効にして論理レプリケーション `Subscription` を作成する必要があります。例

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Subscription

#  ...

spec:
  # ...
  parameters:
    failover: "true"

#  ...
```

構成すると、論理 WAL 送信プロセスは、指定されたレプリケーションスロットが関連する WAL の受信とフラッシュを確認した後のみ、デコードされた変更をプラグインに送信し、次のことを保証します。

- 論理レプリケーションスロットは、パブリッシャーのレプリカが安全に受信するまで変更を消費しません。
- ロジカルレプリケーションクライアントは、フェールオーバー後にデータを欠落することなく、昇格したスタンバイにシームレスに再接続できます。

論理レプリケーションスロットの同期の詳細については、Logical Replication Failover の PostgreSQL ドキュメントを参照してください。

PostgreSQL 16 以前の動作

PostgreSQL 16 以前のバージョンの場合、CloudNativePG クラスターの EDB Postgres AI は pg_failover_slots を使用します

フェールオーバー全体で論理レプリケーションスロットの同期を維持するため。

レプリケーションスロット用に保持される WAL サイズのキャップ

レプリケーションスロットが有効になっている場合、PostgreSQL がレプリケーションスロットによって要求された WAL ファイルを保持しようとするため、ディスク領域が不足する場合があります。これは、スタンバイが一時的にダウンしている、遅延している、または単に孤立したレプリケーションスロットが原因で発生する可能性があります。

PostgreSQL 13 以降、max_slot_wal_keep_size を利用できます。

構成オプションは、レプリケーションスロットがチェックポイント時に `pg_wal` ディレクトリに保持できる WAL ファイルの最大サイズを制御します。デフォルトでは、PostgreSQL では `max_slot_wal_keep_size` は `-1` に設定されています。これは、レプリケーションスロットが無制限の量の WAL ファイルを保持できることを意味します。その結果、レプリケーションスロットのサポートが有効になっているときに `max_slot_wal_keep_size` を明示的に設定することをお勧めします。例

```
# ...
postgresql:
  parameters:
    max_slot_wal_keep_size: "10GB"
# ...
```

レプリケーションスロットの監視

インフラストラクチャでレプリケーションスロットは注意深く監視する必要があります。デフォルトでは、Prometheus エクスポーターの `pg_replication_slots` メトリックに、スロットの名前、タイプ、アクティブかどうか、プライマリからの遅延などの重要な情報を提供します。

5.12 Logical Replication

PostgreSQL は、logical replication を提供することにより、レプリケーション機能を物理的レプリケーションを超えて拡張します。これは、正確なブロックアドレスとバイトごとのコピーのレベルで動作します。論理レプリケーションは、定義されたレプリケーション ID 通常は主キーに基づいて、データオブジェクトとその変更をレプリケートします。

論理レプリケーションでは、パブリッシュアンドサブスクライブモデルを使用し、サブスクライバーはパブリッシャーノードでパブリケーションに接続します。サブスクライバーはこれらのパブリケーションからデータの変更をプルし、再公開できるため、カスケードレプリケーションと複雑なトポロジが有効になります。

この柔軟なモデルは、次の場合に特に役立ちます。

- オンラインデータの移行

- ライブ PostgreSQL バージョンのアップグレード
- システム間のデータ分散
- リアルタイム分析
- 外部アプリケーションとの統合

> **注釈**
>
> の詳細、例、および制限については、

official PostgreSQL documentation on Logical Replication を参照してください。

CloudNativePG クラスターの EDB Postgres AI は、主要な PostgreSQL 論理レプリケーションオブジェクトの宣言サポートを提供することにより、この機能を強化します。

- `Publication` リソースを介した **出版物**
- `Subscription` リソースを介した **サブスクリプション**

5.12.1 出版物

PostgreSQL のパブリッシュアンドサブスクライブレプリケーションモデルでは、

publication

データ変更のソースです。これは、データベース内の 1 つ以上のテーブルから生成された変更セット_replication set_としても知られる論理コンテナとして機能します。パブリケーションは、_publisher_として機能する PostgreSQL 10+インスタンスで定義できます。パブリッククラウドで人気の DBaaS ソリューションによって管理されているインスタンスを含みます。各パブリケーションは単一のデータベースに関連付けられ、どのテーブルと変更をレプリケートするかについての詳細な制御を提供します。

Kubernetes 外部のパブリッシャーの場合、次のことができます create publications using SQL

または *cnp publication create* を活用します。

EDB Postgres AI for CloudNativePG Cluster で `Cluster` オブジェクトを管理する場合、PostgreSQL パブリケーションは `Publication` リソースを介して宣言的に定義できます。

> **注釈**
>
> *API Reference - v1.30.0* を参照してください。

各 `Publication` オブジェクトに定義できる属性の完全なリストについては、

`freddie` という名前のクラスターがあり、 `app` データベース内のすべてのテーブルをレプリケートするとします。これは `Publication` マニフェストです。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Publication
```

(次のページに続く)

(前のページからの続き)

```
metadata:
  name: freddie-publisher
spec:
  cluster:
    name: freddie
  dbname: app
  name: publisher
  target:
    allTables: true
```

上記の例では

- パブリケーションオブジェクトの名前は freddie-publisher metadata.name です。
- パブリケーションは、publisher spec.name という名前の freddie クラスター spec.cluster.name のプライマリを介して作成されます。
- app データベース spec.dbname のすべてのテーブル spec.target.allTables: true が含まれます。

パブリケーションテーブルのきめの細かい制御

allTables オプションは、データベース内のすべてのテーブルをレプリケートする便利な方法を提供しますが、PostgreSQL バージョン 15 以降では、CREATE PUBLICATION を介して柔軟性の強化が導入されました。

コマンド。これにより、どのテーブル、またはどのタイプのデータ変更をパブリケーションに含めるかを正確に定義できます。

複雑なまたは調整されたレプリケーションセットアップについては、

PostgreSQL logical replication documentation を参照してください。

さらに、 *EDB Postgres AI for CloudNativePG Cluster API reference* を参照してください。

レプリケーションターゲットの宣言によるカスタマイズの詳細については、

次の例では、app データベースの portal スキーマ内のすべてのテーブルと、access スキーマの users テーブルをレプリケートするパブリケーションを定義します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Publication
metadata:
  name: publisher
spec:
  cluster:
    name: freddie
  dbname: app
  name: publisher
  target:
```

(次のページに続く)

(前のページからの続き)

```
    objects:
      - tablesInSchema: portal
      - table:
          name: users
          schema: access
```

Publication マニフェストの必須フィールド

Publication オブジェクトには次のフィールドが必要です。

- metadata.name Kubernetes Publication オブジェクトの一意の名前。
- spec.cluster.name PostgreSQL クラスターの名前。
- spec.dbname パブリケーションが作成されるデータベース名。
- spec.name PostgreSQL でのパブリケーション名。
- spec.target パブリケーションに含めるテーブルまたは変更を指定します。

Publication オブジェクトは特定の Cluster を参照し、パブリケーションを作成する場所を決定する必要があります。これはクラスターのプライマリインスタンスによって管理され、必要に応じてパブリケーションが作成または更新されます。

> **警告**
>
> *spec.cluster* フィールドは、作成後は不変です。別の `Cluster` でパブリケーションを作成するには、既存のリソースを更新する代わりに、新しい `Publication` リソースを作成します。

Publication を作成した後、CloudNativePG クラスターの EDB Postgres AI は、指定されたクラスターのプライマリインスタンスでそれを管理します。リコンシリエーションサイクルが成功すると、Publication ステータスは次を反映します。

- applied: true 、構成が正常に適用されたことを示します。
- observedGeneration は metadata.generation と一致し、適用された構成が最新の変更に対応することを確認します。

リコンシリエーション中にエラーが発生した場合、status.applied は false になり、エラーメッセージは status.message フィールドに含まれます。

パブリケーションの削除

publicationReclaimPolicy フィールドは、Publication オブジェクトを削除するときの動作を制御します。

- retain デフォルト手動管理のために PostgreSQL にパブリケーションを残します。
- delete PostgreSQL からパブリケーションを自動的に削除します。

次の例を考えます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Publication
metadata:
  name: freddie-publisher
spec:
  cluster:
    name: freddie
  dbname: app
  name: publisher
  target:
    allTables: true
  publicationReclaimPolicy: delete
```

この場合、 Publication オブジェクトを削除すると、 freddie クラスターの app データベースから publisher パブリケーションも削除されます。

レプリカクラスターでは、データベースは読み取り専用であるため、Publication オブジェクトを削除すると、ファイナライザーが解放され、publicationReclaimPolicy: delete を使用しても PostgreSQL でパブリケーションをドロップせずに Kubernetes オブジェクトが削除されます。パブリケーションの削除は、それを所有するプライマリクラスターに任されます。

5.12.2 サブスクリプション

PostgreSQL のパブリッシュアンドサブスクライブレプリケーションモデルでは、

subscription

は、データの変更を消費するダウンストリームコンポーネントを表します。サブスクリプションは、パブリッシャーのデータベースへの接続を確立し、サブスクライブするパブリケーションのセット 1 つ以上を指定します。サブスクリプションは、_subscriber_として機能するサポートされている PostgreSQL インスタンスで作成できます。

CloudNativePG クラスターの EDB Postgres AI は、 Subscription リソースを使用して宣言的に定義できることにより、サブスクリプション管理を簡素化します。

> **注釈**
>
> *API Reference - v1.30.0* を参照してください。

各 Subscription オブジェクトに定義できる属性の完全なリストについては、

freddie クラスター_publisher_の app データベースの publisher パブリケーションから king クラスター_subscriber_の app データベースに変更をレプリケートするとします。次に、Subscription マニフェストの例を示します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Subscription
metadata:
  name: freddie-to-king-subscription
spec:
  cluster:
    name: king
  dbname: app
  name: subscriber
  externalClusterName: freddie
  publicationName: publisher
```

上記の例では

- サブスクリプションオブジェクトの名前は freddie-to-king-subscriber metadata.name です。
- サブスクリプションは、king クラスター spec.cluster.name の app データベース spec.dbname に、subscriber spec.name という名前で作成されます。
- spec.externalClusterName によって参照される、外部 freddie クラスターの publisher パブリケーションに接続します。

この設定を容易にするには、 king クラスターの構成で freddie 外部クラスターを定義する必要があります。以下は、 king マニフェストで外部クラスターを定義する方法を示す抜粋例です。

```
externalClusters:
  - name: freddie
    connectionParameters:
      host: freddie-rw.default.svc
      user: postgres
      dbname: app
```

注釈

externalClusters セクションの構成の詳細については、ドキュメントの *BootstrapConfiguration* を参照してください。

ご覧のとおり、サブスクリプションは、ネットワーク経由でアクセス可能な PostgreSQL データベースに接続できます。この柔軟性により、ほぼゼロのダウンタイムでデータを Kubernetes にシームレスに移行できます。これは、人気のクラウドベースの DBaaS プラットフォームを含むさまざまな環境から移行するための優れたオプションです。

Subscription マニフェストの必須フィールド

次のフィールドは Subscription オブジェクトを定義するために必須です。

- metadata.name 名前空間内の Kubernetes Subscription オブジェクトの一意の名前。
- spec.cluster.name サブスクリプションが作成される PostgreSQL クラスターの名前。
- spec.dbname サブスクリプションが作成されるデータベースの名前。
- spec.name PostgreSQL に表示されるサブスクリプションの名前。
- spec.externalClusterName spec.cluster.name クラスターの構成で定義された外部クラスターの名前。これは、発行者データベースを参照します。
- spec.publicationName サブスクリプションが接続するパブリッシャーデータベース内のパブリケーションの名前。

Subscription オブジェクトは特定の Cluster を参照し、サブスクリプションを管理する場所を決定する必要があります。CloudNativePG クラスターの EDB Postgres AI は、指定されたクラスターのプライマリインスタンスでサブスクリプションが作成または更新されるようにします。

> **警告**
>
> *spec.cluster* フィールドは、作成後は不変です。別の `Cluster` でサブスクリプションを管理するには、既存のリソースを更新する代わりに、新しい `Subscription` リソースを作成します。

Subscription を作成した後、CloudNativePG クラスターの EDB Postgres AI は、指定されたクラスターのプライマリインスタンスでそれを管理します。リコンシリエーションサイクルが成功すると、Subscription ステータスは次を反映します。

- applied: true 、構成が正常に適用されたことを示します。
- observedGeneration は metadata.generation と一致し、適用された構成が最新の変更に対応することを確認します。

リコンシリエーション中にエラーが発生した場合、status.applied は false になり、エラーメッセージは status.message フィールドに含まれます。

サブスクリプションの削除

subscriptionReclaimPolicy フィールドは、Subscription オブジェクトを削除するときの動作を制御します。

- retain デフォルト手動管理のために PostgreSQL にサブスクリプションを残します。
- delete PostgreSQL からサブスクリプションを自動的に削除します。

次の例を考えます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Subscription
```

(次のページに続く)

(前のページからの続き)

```
metadata:
  name: freddie-to-king-subscription
spec:
  cluster:
    name: king
  dbname: app
  name: subscriber
  externalClusterName: freddie
  publicationName: publisher
  subscriptionReclaimPolicy: delete
```

この場合、Subscription オブジェクトを削除すると、king クラスターの app データベースから subscriber サブスクリプションも削除されます。

レプリカクラスターでは、データベースは読み取り専用であるため、Subscription オブジェクトを削除すると、ファイナライザーが解放され、subscriptionReclaimPolicy: delete を使用しても PostgreSQL でサブスクリプションをドロップせずに Kubernetes オブジェクトが削除されます。サブスクリプションの削除は、それを所有するプライマリクラスターに任されます。

フェイルオーバーに対する復元力

パブリッシャーのフェールオーバー後も論理レプリケーションサブスクリプションが動作し続けるように保証には、クラスター全体で論理デコードスロットを同期するように CloudNativePG クラスターの EDB Postgres AI を構成します。手順の詳細については、**論理デコードスロットの同期** を参照してください。

5.12.3 制限事項

official documentation で概要を示しているように、PostgreSQL の論理レプリケーションには、いくつかの固有の制限があります。特に、次のオブジェクトはレプリケートされません。

- **データベーススキーマと DDL コマンド**
- **シークエンスデータ**
- **大きなオブジェクト**

スキーマレプリケーションのアドレス指定

スキーマレプリケーションに関連する最初の制限は、CloudNativePG クラスターの機能の EDB Postgres AI を使用して簡単に対処できます。たとえば、 import ブートストラップ機能を活用して、レプリケートが必要なテーブルのスキーマをコピーできます。または、PostgreSQL データベースの場合と同様に、スキーマを手動で作成できます。

シーケンスの処理

シーケンスは、論理レプリケーションを介して自動的に同期は保たれませんが、CloudNativePG クラスターの EDB Postgres AI は、ライブマイグレーションで使用されるソリューションを提供します。*CNPG-I* を使用できます

シーケンス値を同期し、パブリッシャーとサブスクライバーデータベース間の一貫性を保証します。

5.12.4 論理レプリケーションを使用したライブ移行とメジャー Postgres アップグレードの例

論理レプリケーションの強力な機能を強調するために、この例では、PostgreSQL 16 を実行しているパブリッシャーデータベース `freddie` から最新の PostgreSQL バージョンを実行しているサブスクライバデータベース `king` にデータをレプリケートする方法を示します。このセットアップは、評価と実践学習のために Kubernetes クラスターに展開できます。

この例は、論理レプリケーションがデータの一貫性を保証しながら、PostgreSQL バージョン間のライブマイグレーションとアップグレードを促進する方法を示しています。CloudNativePG クラスターの EDB Postgres AI と論理レプリケーションを組み合わせることで、Kubernetes 環境でのこのようなシナリオを簡単にセットアップ、管理、および評価できます。

ステップ 1 パブリッシャーの設定 `freddie`

最初のステップでは、バージョン 16 での `freddie` PostgreSQL クラスターを作成します。クラスターには、単一のインスタンスが含まれ、10,000 の数値を保存するテーブル `n` で初期化された `app` データベースが含まれます。 `publisher` という名前の論理レプリケーションパブリケーションも、データベース内のすべてのテーブルを含むように構成されています。

`freddie` クラスターとそのパブリケーションリソースをセットアップするためのマニフェストは次のとおりです。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: DatabaseRole
metadata:
  name: freddie-app
spec:
  cluster:
    name: freddie
  name: app
  login: true
  replication: true
- --
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: freddie
spec:
  instances: 1
```

(次のページに続く)

(前のページからの続き)

```
  imageName: docker.enterprisedb.com/k8s/postgresql:18-standard-ubi9

  storage:
    size: 1Gi

  bootstrap:
    initdb:
      postInitApplicationSQL:
        - CREATE TABLE n (i SERIAL PRIMARY KEY, m INTEGER)
        - INSERT INTO n (m) (SELECT generate_series(1, 10000))
        - ALTER TABLE n OWNER TO app
- --
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Publication
metadata:
  name: freddie-publisher
spec:
  cluster:
    name: freddie
  dbname: app
  name: publisher
  target:
    allTables: true
```

ステップ 2 サブスクライバー king の設定

次に、最新バージョンの PostgreSQL を実行して、king PostgreSQL クラスターを作成します。このクラスターは、外部クラスター構成を使用して freddie クラスター上の app データベースからスキーマをインポートすることにより初期化します。Subscription リソース freddie-to-king-subscription は、freddie の publisher によって公開された変更を消費するように構成されます。

以下は、king クラスターとそのサブスクリプションをセットアップするためのマニフェストです。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: king
spec:
  instances: 1

  imageName: docker.enterprisedb.com/k8s/postgresql:18-standard-ubi9

  storage:
    size: 1Gi
```

(次のページに続く)

(前のページからの続き)

```
  bootstrap:
    initdb:
      import:
        type: microservice
        schemaOnly: true
        databases:
          - app
        source:
          externalCluster: freddie

  externalClusters:
    - name: freddie
      connectionParameters:
        host: freddie-rw.default.svc
        user: app
        dbname: app
      password:
        name: freddie-app
        key: password
- --
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Subscription
metadata:
  name: freddie-to-king-subscription
spec:
  cluster:
    name: king
  dbname: app
  name: subscriber
  externalClusterName: freddie
  publicationName: publisher
```

king クラスターが実行されたら、 app データベースに接続し、n テーブルのレコードをカウントすることにより、レプリケーションが機能していることを確認できます。次の例では、簡単にするために cnp プラグインが提供する psql コマンドを使用しています。

```
kubectl cnp psql king -- app -qAt -c SELECT count(*) FROM n
10000
```

このコマンドは 10000 を返し、 freddie クラスターからのデータが king クラスターに正常にレプリケートされたことを確認します。

cnp プラグインを使用して、既存のシーケンスを同期して、パブリッシャーとサブスクライバー間の一貫性を保証することもできます。次の例は、king クラスターのシーケンスを同期する方法を示しています。

```
kubectl cnp subscription sync-sequences king --subscription=subscriber
SELECT setval("public"."n_i_seq", 10000);

10000
```

このコマンドは、現在の値と一致するように king クラスターのシーケンス n_i_seq を更新し、ソースデータベースと同期していることを確認します。

5.13 Backup

> **注釈**
>
> このセクションでは、PostgreSQL の**物理バックアップ**について説明します。 PostgreSQL は、*pg_dump* ユーティリティを使用した論理バックアップもサポートしていますが、これらは**ビジネス継続性に適しておらず**、CloudNativePG クラスターの EDB Postgres AI によって**管理されません**。 *pg_dump* を使用する場合は、 *_Troubleshooting / Emergency backup_ section* を参照してください。

ガイダンス用に。

CloudNativePG クラスターの EDB Postgres AI は、現在 2 つの主な方法で **PostgreSQL クラスターの物理バックアップ** をサポートしています。

- ** CNPG-I プラグイン経由** CloudNativePG クラスターコミュニティの EDB Postgres AI は **Barman Cloud Plugin** を公式にサポート

オブジェクトストレージサービスとの統合用。

- **ネイティブ** 、以下をサポート

*(**ただし、** Barman Cloud **プラグインを優先して** 1.26 **から非推奨ですが**)* - *Kubernetes Volume Snapshots* 、基になるストレージクラスでサポートされている場合

CloudNativePG クラスターの EDB Postgres AI を使用したバックアップ戦略を選択する前に、**メインコンセプト** でカバーされている基本的な概念を理解することが重要です

セクション。これらには、WAL アーカイブ、ホットおよびコールドバックアップ、スタンバイからのバックアップの実行などが含まれます。

5.13.1 メインコンセプト

PostgreSQL は、ファイルシステムレベルの物理コピーに基づいて、ファーストクラスのバックアップおよびリカバリー機能をネイティブに提供します。これらはミッションクリティカルな運用データベースで 15 年以上使用され、世界中の組織が Postgres でディザスターリカバリー目標を達成するのを支援してきました。

CloudNativePG クラスターの EDB Postgres AI では、各 PostgreSQL クラスターのバックアップインフラストラクチャは次のリソースで構成されています。

- **WAL アーカイブ** Postgres によって継続的に書き込まれ、データ耐久性のためにアーカイブされる WAL ファイルトランザクションログを含む場所

- **物理ベースバックアップ** PostgreSQL がデータベースにデータを保存するために使用するすべてのファイルのコピー主に `PGDATA` およびテーブルスペース

CNPG-I は、WAL アーカイブアーカイブとリストア操作の両方、およびベースバックアップおよび対応する復元プロセスを管理するための汎用かつ拡張可能なインターフェイスを提供します。

WAL アーカイブ

PostgreSQL の WAL アーカイブは、 **継続的バックアップ**の中心であり、次の理由で基本的なものです。

- **ホットバックアップ**サーバーをシャットダウンせずに、Postgres クラスタープライマリまたはスタンバイのインスタンスから物理ベースのバックアップを取得する可能性。オンラインバックアップとしても知られています
- **ポイント インタイム リカバリー** PITR システムで最初に使用可能なベースバックアップから任意の時点で復旧する可能性。

> **警告**
>
> WAL アーカイブだけでは役に立ちません。物理ベースのバックアップがないと、PostgreSQL クラスターを復元できません。

一般に、WAL アーカイブの存在により、PostgreSQL クラスターの復元力が強化され、各インスタンスは、必要に応じてアーカイブから必要な WAL ファイルを取得できます。通常、WAL アーカイブの保持期間は、通常これらのファイルをリサイクルする Postgres インスタンスよりも長くなります。。

このユースケースは、WAL アーカイブに依存するだけで長距離にわたって同期でき、さまざまなリージョンにわたってディザスターリカバリー目標を拡張できるため、**レプリカクラスター** に拡張することもできます。

configure a WAL archive 、 CloudNativePG クラスターの EDB Postgres AI は、リージョンを超えても、ディザスターリカバリーに **リリース候補の目的** ≤ 5 分をすぐに提供します。

コールドおよびホットバックアップ

ホットバックアップは前のセクションで既に定義されています。これらは WAL アーカイブの存在を必要とし、最新のデータベース管理システムでは標準です。

コールドバックアップは、オフラインバックアップとしても知られ、代わりに、PostgreSQL インスタンス スタンバイまたはプライマリがシャットダウンしたときに取得される物理ベースバックアップです。これらは定義ごとに一貫しており、シャットダウンされた時点のデータベースのスナップショットを表します。

その結果、PostgreSQL インスタンスは、WAL アーカイブを必要とせずにコールドバックアップから再起動できますが、利用可能な場合はそれを利用できます前のセクションで強調表示した復旧面のすべての利点を備えています。

RPO が高く 1 時間または 24 時間など、保持期間が短い状況では、コールドバックアップは、ディザスターリカバリー計画で検討される実行可能なオプションを表します。

5.13.2 使用可能なバックアップオプションの比較: オブジェクトストアとボリュームスナップショット

CloudNativePG クラスターの EDB Postgres AI は、現在物理バックアップの 2 つの主要なアプローチをサポートしています。

- **オブジェクトストアベースのバックアップ** 、**Barman Cloud Plugin** または ***deprecated native integration*** 経由
- ***Volume Snapshots*** 、Kubernetes CSI インターフェイスとサポートされているストレージクラスを使用

オブジェクトストアベースのバックアップ

オブジェクトストアへのバックアップ AWS S3、Azure Blob、GCS など

- 常に WAL アーカイブが必要
- ホットバックアップのみをサポート
- インクリメンタルまたは差分コピーをサポートしていません
- リテンションポリシーのサポート

ボリュームスナップショット

ネイティブボリュームのスナップショット

- WAL アーカイブは必要ありませんが、運用環境での使用を強くお勧めします
- 基になるストレージクラスの機能に応じて、インクリメンタルおよび差分コピーをサポート
- ホットおよびコールドバックアップの両方をサポート
- リテンションポリシーをサポートしていません

2 つの間の選択

最適なアプローチは、環境と操作要件によって異なります。次の要素を考慮します。

- **オブジェクトストアの可用性** Kubernetes クラスターが、安定したネットワーキングレイヤーを含む信頼できるオブジェクトストレージソリューションにアクセスできることを確認します。
- **ストレージクラス機能**ストレージクラスが、インクリメンタル/差分機能を備えた CSI ボリュームスナップショットをサポートしていることを確認します。
- **データベースサイズ**非常に大規模なデータベース VLDB の場合、コピーオンライトテクノロジーによる高速なリカバリーが可能になるため、**ボリュームスナップショットが一般的に優先されます** 。これにより、*Recovery Time Objective (RTO)* が大幅に向上します。
- **データモビリティ**オブジェクトストアベースのバックアップは、リージョンまたは環境を超えてバックアップをレプリケートまたは保存するためのより高い柔軟性を提供する場合があります。
- **操作の慣れ**ストレージの管理におけるチームの経験と自信に最も適した方法を選択します。

比較の概要

Feature	Object Store	Volume Snapshots
WAL archiving	Required	Recommended^1^
Cold backup	No	☒
Hot backup	☒	☒
Incremental copy	No	☒^2^
Differential copy	No	☒^2^
Backup from a standby	☒	☒
Snapshot recovery	No^3^	☒
Retention policies	☒	No
Point-in-Time Recovery (PITR)	☒	Requires WAL archive
Underlying technology	Barman Cloud	Kubernetes API

注意事項 > > 1. 現在、WAL アーカイブでは、プラグインまたは非推奨のネイティブ > を介してオブジェクトストアを使用する必要があります。 > 2. インクリメンタルおよび差分コピーの可用性は、PostgreSQL ボリュームに使用されるストレージクラスの機能によって異なります。> 3. スナップショットリカバリーは、`bootstrap.recovery.recoveryTarget.targetImmediate` オプションを使用してエミュレートできます。

5.13.3 スケジュールされたバックアップ

スケジュールされたバックアップは、CloudNativePG クラスターの EDB Postgres AI で信頼性の高いバックアップ戦略を実装するための推奨される方法です。これらは `ScheduledBackup` カスタムリソースを使用して定義されます。

注釈

構成オプションの完全なリストについては、 *ScheduledBackupSpec* を参照してください。

API リファレンスで。

Cron スケジュール

`schedule` フィールドは、秒を含む_6 フィールドの cron 式_を使用して、バックアップを発生する**時期** を定義します。このフォーマットは *Go `cron* package specification <https://pkg.go.dev/github.com/robfig/cron#hdr-CRON_Expression_Format>`_ に従います。

警告

このフォーマットは、従来の Unix/Linux *crontab* とは異なります。最初のエントリとして**seconds** フィールドが含まれます。

毎日のスケジュールされたバックアップの例

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: ScheduledBackup
metadata:
  name: backup-example
spec:
  schedule: "0 0 0  **  *" # At midnight every day
  backupOwnerReference: self
  cluster:
    name: pg-backup
  # method: plugin, volumeSnapshot, or barmanObjectStore (default)
```

スケジュール"0 0 0 ** *" は、毎日深夜 00:00:00 にバックアップをトリガーします。 Kubernetes CronJob では、秒がサポートされていないため、同等の式は 0 0* ** になります。

> **警告**
>
> *spec.cluster* フィールドは、作成後は不変です。別の `Cluster` のバックアップをスケジュールするには、既存のリソースを更新する代わりに、新しい `ScheduledBackup` リソースを作成します。

継続的なバックアップに基づいてディザスターリカバリー戦略を最適化するには

- バックアップからの復元を定期的にテストします。
- 完全なリカバリーに必要な時間を測定します。
- ベースバックアップのサイズと、取得および再生する必要がある WAL ファイルの数を考慮します。

ほとんどの場合、 **毎週のベースバックアップ** で十分です。1 日 1 回よりも多くの頻度で完全なバックアップをスケジュールすることはほとんどありません。

即時バックアップ

ScheduledBackup の作成時にすぐにバックアップをトリガーするには

```
spec:
  immediate: true
```

スケジュールされたバックアップを一時停止する

スケジュールされたバックアップの実行を一時的に停止するには

```
spec:
  suspend: true
```

バックアップ所有者リファレンス .spec.backupOwnerReference

どの Kubernetes オブジェクトをバックアップリソースの所有者として設定するかを制御します。

- none 所有者参照なし 古い動作
- self ScheduledBackup オブジェクトが所有者になります
- cluster PostgreSQL クラスターが所有者になります

5.13.4 オンデマンド バックアップ

オンデマンドバックアップでは、 Backup リソースを作成することにより、いつでもバックアップ操作を手動でトリガーできます。

> **注釈**
>
> 使用可能なオプションの完全なリストについては、API リファレンスの *BackupSpec* を参照してください。

例オンデマンドバックアップの要求

オンデマンドバックアップを開始するには、次のような Backup 要求カスタムリソースを適用します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Backup
metadata:
  name: backup-example
spec:
  method: barmanObjectStore
  cluster:
    name: pg-backup
```

この例では、オペレーターは barman-cloud-backup ツールを使用してバックアッププロセスをオーケストレーションし、構成済みのオブジェクトストアにバックアップを保存します。

バックアップの進行状況のモニタリング

次を使用してバックアップのステータスを確認できます。

```
kubectl describe backup backup-example
```

バックアップの進行中、次のような出力が表示されます。

```
Name:         backup-example
Namespace:    default
...
Spec:
```

(次のページに続く)

(前のページからの続き)

```
  Cluster:
    Name:  pg-backup
Status:
  Phase:       running
  Started At:  2020-10-26T13:57:40Z
Events:        <none>
```

バックアップが正常に完了すると、phase は completed に設定され、出力には追加のメタデータが含まれます。

```
Name:         backup-example
Namespace:    default
...
Status:
  Backup Id:         20201026T135740
  Destination Path:  s3://backups/
  Endpoint URL:      http://minio:9000
  Phase:             completed
  S3 Credentials:
    Access Key Id:
      Name:  minio
      Key:   ACCESS_KEY_ID
    Secret Access Key:
      Name:  minio
      Key:   ACCESS_SECRET_KEY
  Server Name:       pg-backup
  Started At:        2020-10-26T13:57:40Z
  Stopped At:        2020-10-26T13:57:44Z
```

5.13.5 バックアップ方法

CloudNativePG クラスターの EDB Postgres AI は、現在、スケジュールされたバックアップとオンデマンドバックアップとして次のバックアップ方法をサポートしています。

- plugin – CNPG-I プラグインを使用します.spec.pluginConfiguration が必要
- volumeSnapshot – ネイティブ *Kubernetes volume snapshots* を使用します
- barmanObjectStore – *Barman Cloud for object storage* を使用します

(`Barman Cloud Plugin <https://cloudnative-pg.io/plugin-barman-cloud/> `_ を優先して v1.26 から非推奨ですが、下位互換性の為のデフォルトです)

.spec.method フィールドデフォルト barmanObjectStore を使用して方法を指定します。

クラスターがボリュームスナップショットをサポートするように構成されている場合、次のようなスケジュールされたスナップショットバックアップを有効にできます。

```
spec:
  method: volumeSnapshot
```

バックアップ方法として Barman Cloud プラグインを使用するには、`method: plugin` を設定し、それに応じてプラグインを構成します。

Performing a Base Backup に例があります。

5.13.6 スタンバイからのバックアップ

ベースバックアップの取得には、PostgreSQL インスタンスのディスク上のデータセット全体の読み取りが含まれます。これにより、I/O 競合が発生し、アクティブなワークロードのパフォーマンスに影響を与える可能性があります。

この影響を軽減するために、 **CloudNativePG クラスターの EDB Postgres AI は、読み取り専用レプリカからバックアップを実行する PostgreSQL のビルトイン機能を活用して、スタンバイインスタンス**からのバックアップの取得をサポートします。

デフォルトでは、バックアップはクラスター内の **最新のレプリカ**で実行されます。使用可能なレプリカがない場合、バックアップは**プライマリインスタンス** にフォールバックします。

> **注釈**
>
> このセクションの例では、バックアップターゲットの選択に焦点を当てており、バックアップ方法 *spec.method* は、説明しているスコープに関連していないため、考慮されていません。

仕組み

`prefer-standby` がターゲットである場合デフォルトの動作、CloudNativePG クラスターの EDB Postgres AI は次のことを試行します。

1. 最も同期されたスタンバイノードを特定します。
2. そのスタンバイでバックアッププロセスを実行します。
3. 使用可能なスタンバイがない場合、プライマリにフォールバックします。

この戦略により、プライマリのワークロードへの干渉が最小限に抑えられます。

> **警告**
>
> スタンバイはプライマリと常に最新の状態であるとは限りませんが、最初に使用可能なバックアップから最後にアーカイブされた WAL までの時間連続性では、これは通常無関係です。ベースバックアップは、確かに、PITR を含む復旧操作を開始する開始ポイントを表します。 pg_basebackup で起こることと同様に、オンラインスタンバイからバックアップする場合、プライマリで WAL のスイッチを強制しません。これにより、書き込みアクティビティが低い展開では、短期的に`archive_timeout` がキックインする前に予期しない結果が発生する可能性があります。

プライマリでのバックアップの強制

常にプライマリインスタンスでバックアップを実行するには、クラスター構成でバックアップターゲットを primary に明示的に設定します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata: [...]
spec:
  backup:
    target: "primary"
```

> **警告**
>
> *primary* を**ボリュームスナップショットを使用したコールドバックアップ**のターゲットとして使用する場合、プライマリインスタンスを一時的にシャットダウンする必要があるため、すべての書き込み操作を中断する必要があるため、注意してください。ターゲットを明示的に設定していない場合でも、同じ注意が単一インスタンスのクラスターに適用されます。

クラスター全体のターゲットのオーバーライド

Backup または ScheduledBackup リソースを使用して、バックアップごとにクラスターレベルのターゲットをオーバーライドできます。次に、オンデマンドバックアップの例を示します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Backup
metadata: [...]
spec:
  cluster:
    name: [...]
  target: "primary"
```

この例では、クラスターのデフォルトターゲットが prefer-standby であっても、バックアップはプライマリインスタンスから取得されます。

5.13.7 リテンションポリシー

CloudNativePG クラスターの EDB Postgres AI は、**バックアップに依存しないアーキテクチャ**に向けて進化しており、バックアップの責任が外部 **CNPG-I プラグイン**に委任されます。これらのプラグインは、CloudNativePG クラスターの EDB Postgres AI のビルトイン機能と範囲を超える、高度なリテンション管理を含む高度なカスタマイズ可能なデータ保護機能を提供することが期待されています。

この移行の一環として、Cluster リソースの spec.backup.retentionPolicy フィールドは **非推奨**であり、将来のリリースでは削除される予定です。

使用可能な保持機能の詳細については、選択したプラグインのドキュメントを参照してください。例 Retention Policies 。

5.14 Recovery

PostgreSQL では、**リカバリ**は、既存の物理バックアップからインスタンスを起動するプロセスを指します。PostgreSQL のリカバリシステムは堅牢で機能が豊富で、**ポイントインタイムリカバリー PITR** をサポートしています。これは、使用可能な最も古いバックアップから最新のアーカイブされた WAL ファイルまで、クラスターを特定の瞬間に復元する機能です。

CloudNativePG クラスターの EDB Postgres AI では、復旧は既存のクラスターで**インプレースで実行されません**。代わりに、物理バックアップから**新しいクラスター**をブートストラップするために使用されます。

> **注釈**
>
> *bootstrap* スタンザの構成の詳細については、 *BootstrapConfiguration* を参照してください。

`recovery` ブートストラップモードでは、物理ベースバックアップからクラスターを初期化し、関連する WAL ファイルを再生して、システムを一貫したオプションでポイントインタイムの状態にすることができます。

CloudNativePG クラスターの EDB Postgres AI は、以下を介した復旧をサポートしています

- **プラグ可能バックアップおよびリカバリーインターフェイス CNPG-I** 、Barman Cloud Plugin などの外部ツールとの統合を可能にします。
- **ボリュームスナップショットからのネイティブリカバリ**、基になる Kubernetes ストレージインフラストラクチャでサポートされている場合。
- **Barman Cloud** を介したオブジェクトストアからのネイティブリカバリー。これは、プラグインベースのアプローチを優先して、バージョン 1.26 の時点で**非推奨**です。

バージョン 1.26 でのネイティブ Barman Cloud サポートの非推奨に伴い、このセクションでは、サポートされている 2 つのリカバリ方法に焦点を当てます。オブジェクトストアからのリカバリには **Barman Cloud Plugin** を使用し、ボリュームスナップショットからのリカバリには**ネイティブインターフェイス**を使用します。

5.14.1 Barman Cloud プラグインを使用したオブジェクトストアからのリカバリー

このセクションでは、推奨される Barman Cloud プラグインを使用してオブジェクトストアから PostgreSQL クラスターを復旧する方法の概要を説明します。

> **注釈**
>
> 詳細については、

“Recovery of a Postgres Cluster” section in the Barman Cloud Plugin documentation を参照してください。

最初に、ベースバックアップと WAL ファイルの両方を保持するオブジェクトストアを定義します。Barman Cloud プラグインは、この目的にカスタム `ObjectStore` リソースを使用します。次の例は、Azure Blob Storage 用に構成する方法を示しています。

```yaml
apiVersion: barmancloud.cnpg.io/v1
kind: ObjectStore
metadata:
  name: cluster-example-backup
spec:
  configuration:
    destinationPath: https://STORAGEACCOUNTNAME.blob.core.windows.net/CONTAINERNAME/
    azureCredentials:
      storageAccount:
        name: recovery-object-store-secret
        key: storage_account_name
      storageKey:
        name: recovery-object-store-secret
        key: storage_account_key
    wal:
      maxParallel: 8
```

次に、定義した ObjectStore を使用するように Cluster リソースを構成します。 bootstrap セクションで、リカバリソースを指定し、プラグインを参照する externalCluster エントリーを定義します。

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-restore
spec:
  [...]

  superuserSecret:
    name: superuser-secret

  bootstrap:
    recovery:
      source: origin

  externalClusters:
    - name: origin
      plugin:
        name: barman-cloud.cloudnative-pg.io
        parameters:
          barmanObjectName: cluster-example-backup
          serverName: cluster-example
```

5.14.2 VolumeSnapshot オブジェクトからのリカバリー

> **警告**
>
> *VolumeSnapshot* からプライマリインスタンスをリカバリした後にレプリカを作成する場合、オペレーターは`pg_basebackup`を使用して同期することにフォールバックする場合があります。このプロセスには、完全なベースバックアップが含まれるため、特に大規模なデータベースの場合、かなり遅くなる場合があります。この制限は、スケールアッププロセスでのオンラインバックアップと PVC クローンのサポートにより、将来的に解決される予定です。

CloudNativePG クラスターの EDB Postgres AI を使用すると、既存の `Cluster` に属する `PersistentVolumeClaim` PVC の `VolumeSnapshot` から新しいクラスターを作成できます。これらのスナップショットは、**ボリュームスナップショットバックアップを構成する方法** の宣言的 API を使用して作成されます。

リカバリプロセスを完了するには、新しいクラスターが、変更を再適用し、リカバリを最終化するために必要な WAL アーカイブへのアクセスを提供する外部クラスターを参照する必要があります。

次の例は、ベースバックアップの `VolumeSnapshot` と、 Barman Cloud プラグインを介してアクセスされる WAL アーカイブの両方を使用して、リカバリされるクラスターを示しています。

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-restore
spec:
  [...]

  bootstrap:
    recovery:
      source: origin
      volumeSnapshots:
        storage:
          name: <snapshot name>
          kind: VolumeSnapshot
          apiGroup: snapshot.storage.k8s.io

  externalClusters:
    - name: origin
      plugin:
        name: barman-cloud.cloudnative-pg.io
        parameters:
          barmanObjectName: cluster-example-backup
          serverName: cluster-example
```

バックアップされたクラスターが別の PVC を使用して WAL ファイルを保存していた場合、リカバリーにはそれも含める必要があります。

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-restore
spec:
  [...]

  bootstrap:
    recovery:
      volumeSnapshots:
        storage:
          name: <snapshot name>
          kind: VolumeSnapshot
          apiGroup: snapshot.storage.k8s.io

        walStorage:
          name: <snapshot name>
          kind: VolumeSnapshot
          apiGroup: snapshot.storage.k8s.io
```

前の例では、アプリケーションデータベースとその所有ユーザーの名前がデフォルトで app であることを前提としています。リストアする PostgreSQL クラスターが別の名前を使用している場合、**アプリケーションデータベースの構成** に記載されているように、リカバリフェーズを終了する前にこれらの名前を指定する必要があります。

> **警告**
>
> スナップショットからレプリカモードクラスターをブートストラップして、プライマリだけでなくスタンバイインスタンスのスナップショットを活用する場合、次のことをお勧めします。

1. 単一インスタンスのレプリカクラスターから開始します。プライマリインスタンスは、スナップショットとソースクラスタからの使用可能な WAL を使用して復旧されます。
2. レプリカクラスター内のプライマリのスナップショットを取得します。
3. 必要に応じて、レプリカクラスター内のインスタンスの数を増やします。 .. :: ## *Backup* オブジェクトからのリカバリー

クラスターを作成する必要がある名前空間で Backup リソースが既に利用可能な場合、次の例のように. `spec.bootstrap.recovery.backup.name` を使用して名前を指定できます。

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-initdb
spec:
  instances: 3
```

(次のページに続く)

(前のページからの続き)

```
  bootstrap:
    recovery:
      backup:
        name: backup-example

  storage:
    size: 1Gi
```

このブートストラップ方法では、復元する必要があるバックアップへの参照のみを指定できます。

前の例では、アプリケーションデータベースとその所有ユーザーの名前がデフォルトで app であることを前提としています。リストアする PostgreSQL クラスターが別の名前を使用している場合、**アプリケーションデータベースの構成** に文書されているように、リカバリフェーズを終了する前にこれらの名前を指定する必要があります。

5.14.3 追加の考慮事項

オブジェクトストア、ボリュームスナップショット、または既存の Backup リソースからリカバリーするかどうかにかかわらず、Cluster が完全にプライマリに昇格し、書き込み操作を受け入れるまで、カタログを含むデータベースを変更することはできません。この制限には、Cluster がプライマリに移行するまで遅延されるロールオーバーライドが含まれます。その結果、次の考慮事項が適用されます。

- アプリケーションデータベース名とユーザーは、復元されるバックアップからコピーされます。これは、Kubernetes クラスターの通常のメンテナンスアクティビティの一部であるため、現在基になるシークレットのバックアップは行っていません。
- 元の postgres ユーザーパスワードを保持するには、enableSuperuserAccess を構成し、superuserSecret を提供します。

デフォルトでは、リカバリはデフォルトのターゲットタイムライン latest で使用可能な最新の WAL まで続行されます。オプションで recoveryTarget を指定して、ポイントインタイムリカバリーを実行できます**ポイント インタイム リカバリー** *PITR* を参照してください。

5.14.4 ポイント インタイム リカバリー PITR

ベースバックアップを抽出した後、最新の WAL までのすべての WAL を再生する代わりに、任意の時点で WAL の再生を停止するように PostgreSQL に依頼できます。PostgreSQL は、この技術を使用して PITR を達成します。WAL アーカイブの存在は必須です。

リカバリターゲットが指定された場合、オペレーターは、この機能が動作するために必要な構成パラメーターを生成します。

オブジェクトストアからの PITR

この例では、 Barman Cloud プラグイン用に前に定義した Azure の同じリカバリオブジェクトストアを使用します。ベースバックアップと WAL アーカイブの両方が含まれます。リカバリターゲットは、要求されたタイムスタンプに基づいています。

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-restore-pitr
spec:
  instances: 3

  storage:
    size: 5Gi

  bootstrap:
    recovery:
      # Recovery object store containing WAL archive and base backups
      source: origin
      recoveryTarget:
        # Time base target for the recovery
        targetTime: "2023-08-11 11:14:21.00000+02"

  externalClusters:
    - name: origin
      plugin:
        name: barman-cloud.cloudnative-pg.io
        parameters:
          barmanObjectName: cluster-example-backup
          serverName: cluster-example
```

この例では、タイムスタンプの形式で `targetTime` のみを指定する必要がありました。リカバリーを開始するベースバックアップを指定する必要はありません。

`backupID` オプションは、リカバリプロセスを開始するベースバックアップを指定できるオプションです。デフォルトでは、この値は空です。

Barman バックアップ ID の形式で値を割り当てると、オペレーターはそのバックアップをリカバリのベースとして使用します。

バックアップ ID を指定しない場合、オペレーターは次のようにリカバリのベースバックアップを検出します。

- `targetTime` または `targetLSN` を使用する場合、オペレーターは、そのターゲットより前に完了した最も近いバックアップを選択します。
- それ以外の場合、オペレーターは最後の利用可能なバックアップを時系列で選択します。

VolumeSnapshot オブジェクトからのポイントインタイムリカバリー PITR

次の例は、以下を使用して **ポイントインタイムリカバリー PITR** を実行する方法を示しています。

- ベースバックアップを提供する PGDATA ディレクトリの Kubernetes VolumeSnapshot 。このスナップショットは recovery.volumeSnapshots セクションで指定され、test-snapshot-1 名前付けです。
- アーカイブされた WAL ファイルを含むリカバリオブジェクトストアこの場合、MinIO。オブジェクトストアは、Barman Cloud Plugin ObjectStore リソースここには表示されていませんを介して定義され、外部クラスター構成をポイントする recovery.source フィールドを使用して参照されます。

クラスターは、 recoveryTarget.targetTime オプションを使用して特定のポイントに復元されます。

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-snapshot
spec:
  # ...
  bootstrap:
    recovery:
      source: origin
      volumeSnapshots:
        storage:
          name: test-snapshot-1
          kind: VolumeSnapshot
          apiGroup: snapshot.storage.k8s.io
      recoveryTarget:
        targetTime: "2023-07-06T08:00:39Z"
  externalClusters:
    - name: origin
      plugin:
        name: barman-cloud.cloudnative-pg.io
        parameters:
          barmanObjectName: minio-backup
          serverName: cluster-example
```

このセットアップにより、CloudNativePG クラスターの EDB Postgres AI は、ボリュームスナップショットからベースデータを復元し、オブジェクトストアから WAL セグメントを適用して、目的のリカバリターゲットに到達できます。

> **注釈**
>
> バックアップクラスターで`walStorage` が有効になっていた場合、*Recovery from VolumeSnapshot objects* で述べたように、*PGWAL* ディレクトリを含むボリュームスナップショットも指定する必要があります。

警告

ボリュームスナップショットのベースバックアップの終了時刻がリカバリターゲットのタイムスタンプより前であることを確認するのは自分の責任です。

警告

最後のベースバックアップ以降にクラスター内の 宣言的テーブルスペース を追加または削除した場合、WAL の再生は失敗します。テーブルスペースの変更からリカバリターゲットのタイムスタンプまでのベースバックアップが必要です。

リカバリーターゲット

次に、使用可能なリカバリターゲット基準を示します。

ターゲットタイム

リカバリーが進行するまでのタイムスタンプ。 RFC 3339 形式または timestamp として表現されます。正確な停止ポイントは、 `exclusive` オプションの影響も受けます。

注釈

明示的なタイムゾーンサフィックスのないタイムスタンプたとえば *2023-07-06 08:00:39* は、UTC として解釈されます。

警告

あいまいさを回避するには、タイムスタンプに常に明示的なタイムゾーンを指定します。たとえば、*2023-07-06 08:00:39* の代わりに `2023-07-06T08:00:39Z` または `2023-07-06T08:00:39+02:00` を使用します。

警告

PostgreSQL リカバリーは、指定された時間の後に発生する最初のトランザクションに遭遇すると停止します。ターゲット時間後にそのようなトランザクションが存在しない場合、リカバリプロセスは失敗します。

ターゲット XID

リカバリを続行するトランザクション ID。正確な停止ポイントは、`exclusive` オプションの影響も受けます。回復されるトランザクションは、指定されたトランザクションより前にコミットされたトランザクション、およびオプションで含まれるトランザクションです。

ターゲット名

リカバリーが続行する名前付けリストアポイント pg_create_restore_point() で作成されます。

targetLSN

リカバリが続行されるログ先行書き込みの場所の LSN。正確な停止ポイントは、exclusive オプションの影響も受けます。

targetImmediate

リカバリーは、一貫した状態に到達するとすぐに、つまりできるだけ早く終了します。オンラインバックアップから復元する場合、これはバックアップの取得が終了したポイントを意味します。

この例では、targetName ベースのリカバリターゲットを使用します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
[...]
  bootstrap:
    recovery:
      source: origin
      recoveryTarget:
        backupID: 20220616T142236
        targetName: restore_point_1
[...]
```

各 recoveryTarget 構成のターゲットから 1 つだけを選択できます。

さらに、 targetTLI を指定して、特定のタイムラインに強制的にリカバリーできます。

デフォルトでは、以前のパラメーターは包括的であるとみなされ、リカバリターゲットの直後で停止し、the behavior in PostgreSQL と一致します。

exclusive パラメーターを true に設定することにより、リカバリターゲットの直前で停止する排他的動作を要求できます。次の例は、この動作を示しており、ベースバックアップと WAL アーカイブの両方に Azure の BLOB コンテナーに依存します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-restore-pitr
spec:
  instances: 3

  storage:
    size: 5Gi

  bootstrap:
    recovery:
      source: origin
      recoveryTarget:
```

(次のページに続く)

(前のページからの続き)

```
      backupID: 20220616T142236
      targetName: "maintenance-activity"
      exclusive: true

  externalClusters:
    - name: origin
      plugin:
        name: barman-cloud.cloudnative-pg.io
        parameters:
          barmanObjectName: cluster-example-backup
          serverName: cluster-example
```

5.14.5 アプリケーションデータベースの構成

復旧したクラスターの場合、追加の構成でアプリケーションデータベース名と資格情報を構成できます。アプリケーションデータベースの資格情報を更新するには、独自のパスワードを生成、シークレットとして保存し、シークレットを使用するようにデータベースを更新できます。または、オペレーターにランダムで安全なパスワードを使用してシークレットを生成させて使用することもできます。空のクラスターをブートストラップする `initdb <空のクラスターをブートストラップする *initdb*>` を参照

シークレットの詳細については、

次の例では、ライブクラスターからブートストラップに従って、所有者 app と提供されたシークレット app-secret に保存されたパスワードを使用して app データベースを構成します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
[...]
spec:
  bootstrap:
    recovery:
      database: app
      owner: app
      secret:
        name: app-secret
      [...]
```

上記の構成では、次は **復旧が完了した後** にのみ発生します。

1. app データベースが存在しない場合、作成されます。
2. app ユーザーが存在しない場合、作成されます。
3. app ユーザーが app データベースの所有者でない場合、所有権は app ユーザーに付与されます。
4. owner 値がシークレットの username 値と一致する場合、アプリケーションユーザーこの場合は app ユーザーのパスワードはシークレットの password 値に更新されます。

5.14.6 リカバリーの内部的な仕組み

オブジェクトストレージにアップロードされたデータを使用して、既存のバックアップから新しいクラスターを_bootstrap_できます。オペレーターは、`barman-cloud-restore` ツールベースバックアップ用と `barman-cloud-wal-restore` ツール WAL ファイル用、要求に応じて並列サポートを含むを使用して、リカバリプロセスをオーケストレーションします。

`recovery` ブートストラップ方法の詳細と手順については、バックアップからのブートストラップ `recovery <バックアップからのブートストラップ *recovery*>` を参照してください。

は動作しますが、 `.spec.postgresql.parameters` に関しては、リカバリクラスターを元のクラスターとして構成することをお勧めします。新しいクラスターが復元されたら、必要に応じて設定を変更できます。

その仕組みは、オペレーターが新しいクラスターの最初のインスタンスに init コンテナを注入し、init コンテナがオブジェクトストレージからバックアップのリカバリを開始することです。

ベースバックアップ復旧プロセスが完了すると、オペレーターは Postgres インスタンスをリカバリモードで起動します。このフェーズでは、PostgreSQL は起動していますが、接続を受け入れることはできず、liveness プローブによれば、ポッドは正常です。`restore_command` を介して、PostgreSQL はアーカイブから WAL ファイルの取得を開始します。`maxParallel` オプションを設定し、並列 WAL リストア機能を有効にすることにより、このフェーズを高速化できます。

このフェーズは、PostgreSQL がターゲット、WAL の終わりまたは PITR の場合は必要なターゲットに到達すると終了します。オプションで `recoveryTarget` を指定して、PITR を実行できます。指定しない場合、リカバリはデフォルトのターゲットタイムライン `latest` で最新の使用可能な WAL まで続行されます。

復旧が完了すると、オペレーターは必要なスーパーユーザーのパスワードをインスタンスに設定します。新しいプライマリインスタンスは通常どおり起動し、残りのインスタンスはレプリカとしてクラスターに参加します。

プロセスはユーザーに対して透過的であり、ポッドで実行されているインスタンスマネージャーによって管理されます。

5.14.7 バックアップセクションを使用したクラスターへの復元

クラスターを復元する場合、マニフェストには、_backup_オブジェクトストアリソースを指す Barman Cloud プラグインを含む `plugins` セクションが含まれる場合があります。これにより、新しく作成されたクラスターは、バックアップポリシーが構成されている場合、復旧直後に WAL ファイルのアーカイブとバックアップの取得を開始できます。

同じクラスター内の_backup_と_recovery_の両方で同じ `ObjectStore` 構成を再利用しないでください。必要に応じて、各クラスターが一意の `serverName` を使用していることを確認して、バックアップまたは WAL アーカイブデータの誤って上書きを防ぎます。

> **警告**
>
> CloudNativePG クラスターの EDB Postgres AI には、クラスターが共有ストレージバケットの既存のデータを上書きしないようにする安全チェックが含まれています。競合が検出された場合、クラスターは `Setting up primary` 状態のままで、関連付けられたポッドはエラーで失敗します。ポッドログには `ERROR: WAL archive check failed for server recoveredCluster: Expected empty archive` が表示されます。

5.15 WAL archiving

CloudNativePG クラスターの EDB Postgres AI での書き込み先行ログ WAL アーカイブは、PostgreSQL プライマリから指定されたオブジェクトストアに WAL ファイルを継続的にシッピングするプロセスです。これらのアーカイブは、ポイントインタイムリカバリー PITR を有効にするために必要であり、オブジェクトストアとボリュームスナップショットベースのバックアップ戦略の両方の基本的なコンポーネントです。

5.15.1 プラグインベースのアーキテクチャ

CloudNativePG クラスターの EDB Postgres AI は、*spec.pluginConfiguration* で定義された**プラグインベースのメカニズム** を介して WAL アーカイブをサポート

`Cluster` リソースのセクション。

一度に 1 つのプラグインのみが WAL アーカイブを担当できます。これは、プラグイン構成内で `isWALArchiver` フィールドを `true` に設定することにより構成されます。

5.15.2 サポートされているプラグイン

現在、**Barman Cloud Plugin** は、CloudNativePG クラスターコミュニティの EDB Postgres AI によって維持されている唯一の公式にサポートされている WAL アーカイブプラグインです。完全なドキュメント、構成オプション、およびベストプラクティスについては、Barman Cloud Plugin documentation を参照してください。

5.15.3 非推奨の通知 ネイティブ Barman Cloud

CloudNativePG クラスターの EDB Postgres AI は、`.spec.backup.barmanObjectStore` フィールドを介してネイティブに WAL アーカイブをサポートしています。まだ機能していますが、**このインターフェイスは非推奨**であり、将来のリリースでは削除される予定です。

現在ネイティブの`.spec.backup.barmanObjectStore` アプローチを使用している場合、スムーズな移行のための公式ガイド Migrating from Built-in EDB Postgres AI for CloudNativePG Cluster Backup を参照してください。

5.15.4 アーカイブタイムアウトについて

デフォルトでは、CloudNativePG クラスターの EDB Postgres AI は `archive_timeout` を `5min` に設定し、ワークロードが低い場合でも WAL ファイルが少なくとも 5 分ごとに閉じてアーカイブされ、目標リカバリポイントの決定論的な時間ベースの値を提供します**リリース候補の目的**)。

archive_timeout の値を変更する場合でも、私たちの経験から、オペレーターが設定したデフォルト値がほとんどのユースケースに適していることが示唆されています。

5.16 Service management

PostgreSQL クラスターには、CloudNativePG クラスターの EDB Postgres AI によって直接管理される標準の Kubernetes ネットワークサービスを介してのみアクセスする必要があります。詳細については、Service を参照してください。

CloudNativePG クラスターの EDB Postgres AI は、Cluster リソースごとに 3 種類のサービスを定義します。

- rw クラスターのプライマリインスタンスへのポイント読み取り/書き込み。
- ro レプリカを指します使用可能な場合読み取り専用。
- r クラスター内の PostgreSQL インスタンスを指します読み取り。

デフォルトでは、CloudNativePG クラスターの EDB Postgres AI は、次の規則で Cluster リソースの上記のすべてのサービスを作成します。

- サービスの名前は、<CLUSTER_NAME>-<SERVICE_NAME> の形式に従います。
- すべてのサービスは ClusterIP タイプです。

この設定は、同じ Kubernetes クラスター内の PostgreSQL にアクセスするためのほとんどのユースケースをカバーしますが、CloudNativePG クラスターの EDB Postgres AI は、次のような柔軟性を提供します。

- ro および/または r デフォルトサービスの作成を無効にします。
- Kubernetes が提供する標準の Service API を使用して、独自のサービスを定義します。

これら 2 つのオプションを組み合わせることができます。

一般的なシナリオは、Kubernetes クラスターの外部からデータベースへのアクセスが必要な DBaaS コンテキストで CloudNativePG クラスターの EDB Postgres AI を使用する場合に発生します。このような場合、Kubernetes 環境で使用可能な場合、タイプ LoadBalancer の独自のサービスを作成できます。

5.16.1 デフォルトサービスの無効化

managed.services.disabledDefaultServices を介して、 ro および r デフォルトサービスの一部またはすべてを無効にできます。

たとえば、 ro 読み取り専用と r 読み取りサービスの両方を削除する場合は、次の構成を使用できます。

```
#  <snip>

managed:
  services:
    disabledDefaultServices: ["ro", "r"]
```

5.16.2 独自のサービスの追加

managed.services.additional を介して追加サービスのリストを定義できます。

selectorType フィールドでサービスタイプ rw などを指定し、オプションで updateStrategy を指定します。

serviceTemplate フィールドは、ネットワーク Service リソースの標準の Kubernetes API にアクセスし、必要に応じて metadata および spec セクションの両方を定義できます。

name をサービスに提供し、selector フィールドはオペレーターによって管理されるため、定義しないでください。

> **警告**
>
> サービステンプレートは、PostgreSQL データベースへのネットワークアクセスの構成という点で無限の可能性を提供します。これは、サービスが期待どおりに動作することを保証するための、あなたの側の責任の増大を意味します。 CloudNativePG クラスターの EDB Postgres AI は、セレクターを尊重することを除き、サービス構成を制御できません。

`updateStrategy` フィールドを使用すると、オペレーターがサービス定義を更新する方法を制御できます。デフォルトでは、オペレーターは `patch` ストラテジーを使用し、変更をサービスに直接適用します。または、`replace` ストラテジーは、既存のサービスを削除し、テンプレートから再作成します。

> **警告**
>
> *replace* ストラテジーは、変更されるたびにサービスの中断を発生させます。ただし、サービスの作成中にのみ設定できる特定のパラメーターを変更するために必要になる場合があります。

たとえば、PostgreSQL データベースプライマリに単一の `LoadBalancer` サービスを使用する場合、次の抜粋を使用できます。

```yaml
#  <snip>

managed:
  services:
    additional:
      - selectorType: rw
        serviceTemplate:
          metadata:
            name: "mydb-lb"
            labels:
              test-label: "true"
            annotations:
              test-annotation: "true"
          spec:
            type: LoadBalancer
```

上記の例は、作成したサービスのアノテーションやラベルなどのメタデータを設定する方法も示しています。

Postgres サービスの公開について

PostgreSQL サービスを Kubernetes クラスターの外部に公開するには、主に 3 つのユースケースがあります。

- 一時的に、テスト用。
- 永続的に、 **DBaaS 目的** 。
- 長期/永久簡単または持続的にコンテナ化できず、Kubernetes の外部の仮想マシンまたは物理マシンに存在する必要がある**古いアプリケーション** の場合。この使用例は DBaaS に非常に似ています。

パブリックネットワークからデータベースへのアクセスを許可すると、データベースが悪意のあるユーザーからの潜在的な攻撃にさらされる可能性があることに注意してください。

警告

外部アクセスを許可する前にデータベースを保護していることを確認するか、Kubernetes クラスターがプライベートネットワークからのみ到達可能であることを確認してください。

5.17 PostgreSQL Role management

注釈

CloudNativePG クラスターの EDB Postgres AI は、PostgreSQL インスタンスで必要な特定のロールの作成を管理してきました。

- `postgres` スーパーユーザー、`streaming_replica` 、`cnp_pooler_pgbouncer` などの一部の予約済みユーザー PgBouncer `Pooler` を使用する場合
- アプリケーションデータベースの低特権の所有者として設定されたアプリケーションユーザー

このプロセスは、 *BootstrapConfiguration* セクションで説明しています。

1. as standalone `DatabaseRole resources <#the-databaserole-resource>` 推奨 、 または
2. インライン管理対象ロール 経由 。

5.17.1 共存と優先

2 つの方法は相互排他的ではありません。それぞれで異なるロールを同時に管理できます。これが、インラインスタンザから `DatabaseRole` リソースへの段階的な移行が可能です。同じロール名が両方の場所で定義されている場合のルールのみが必要です。

その場合、 **クラスター仕様 ``managed.roles`` が常に優先されます** 。`DatabaseRole` はリコンサイル化されず、そのステータスの競合を報告します Status of `DatabaseRole resources <#status-of-databaserole-resources>` を参照してください。

重要

宣言的なロール管理は、データベースには存在するが、クラスター仕様にも `DatabaseRole` にも含まれていないロールを無視します。これらのロールのライフサイクルは PostgreSQL 内で引き続き管理されるため、都合の良いときにこの機能を採用できます。

5.17.2 一般的なロール構成の注意事項

使用される管理方法に関係なく、ロール仕様は PostgreSQL structure and naming conventions に従います。

> **Tip**
>
> *API Reference - v1.30.0* を参照してください。

属性の完全なリストについては、

1. `ensure` 属性は PostgreSQL の **ではありません**。宣言的ロール管理がロールを作成する `present` デフォルト `absent` を有効にし、インライン **インライン管理対象ロール** スタンザで **のみ**使用できます。`DatabaseRole` は `ensure` をサポートしていません。代わりに **ロール再利用ポリシー** を介してロールの削除を表現します。
2. `inherit` 属性は、PostgreSQL の規則に従って、デフォルトで true です。
3. `connectionLimit` 属性は、PostgreSQL の規約に従って、デフォルトの-1 に設定されます。
4. `inRoles` のロールメンバーシップは、デフォルトでメンバーシップなしです。

5.17.3 DatabaseRole リソース

`DatabaseRole` カスタムリソースは、PostgreSQL データベースロールを管理する専用の Kubernetes ネイティブ方法を提供します。

これは、クラスターインフラストラクチャからロールのライフサイクルを分離するため、最新の環境と GitOps ワークフローに推奨されるアプローチです。

> **注釈**
>
> *DatabaseRole* は、仕様またはパスワードの Secret が変更されたときに適用されます。手動`ALTER ROLE`など、データベースに直接行われた変更は、次回リソースが適用されるまで検出または元に戻されません。対照的に、インライン管理ロールは定期的にデータベースカタログと比較され、仕様に戻されます。

`DatabaseRole` は名前空間スコープです。リソース、`spec.cluster` を介して参照する `Cluster` 、および消費する `passwordSecret` はすべて同じ名前空間に存在する必要があります。

マニフェストの例

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: DatabaseRole
metadata:
  name: role-dante
spec:
  cluster:
```

(次のページに続く)

(前のページからの続き)

```
  name: cluster-example
name: dante
comment: "Dante Alighieri"
login: true
superuser: false
createdb: true
databaseRoleReclaimPolicy: delete
inRoles:
  - pg_monitor
passwordSecret:
  name: cluster-example-dante
```

ロール定義のマニフェストの例は、ファイル role-examples.yaml にあります。

ロール再利用ポリシー

`databaseRoleReclaimPolicy` フィールドは、 `DatabaseRole` カスタムリソースが Kubernetes API から削除されるときのオペレーターの「最終動作」を定義します。これは、Kubernetes の永続ボリュームの動作を反映しています。

- **``retain`` デフォルト**ロールはデータベースに残ります。これは運用環境にとって最も安全な設定であり、マニフェストが誤って削除された場合でも、データベースユーザーおよびその所有するオブジェクトは変更されないままです。
- **``delete`` :** オペレーターは、Kubernetes オブジェクトがファイナライズされる前に、PostgreSQL で `DROP ROLE` を実行しようとします。これは、一時的な環境または自動化された環境に最適です。

注釈

ロールがオブジェクトテーブル、スキーマなどを所有している場合、*DROP ROLE* は失敗し、*DatabaseRole* は`Terminating` に残り、これらのオブジェクトが再割り当てまたは削除されるまで定期的に再試行されます。オペレーターは、あなたに代わって所有オブジェクトをドロップすることはありません。PostgreSQL でそれらを再割り当てまたはドロップするか、*retain* に切り替えて、削除を完了します。

ロールを削除する方法は、ロールの作成方法によって異なります。

- **``DatabaseRole`` によって作成されます**リソースを削除します。ロールが PostgreSQL から削除されるかどうかは、その**ロール再利用ポリシー** によって制御されます。
- **既存の、または他の場所で管理されている** `DatabaseRole` は、ドロップするツールではありません。インライン **インライン管理対象ロール** を介して `absent` を宣言します

スタンザを削除するか、 `DROP ROLE` を直接実行します。

警告

既に存在するロールの `DatabaseRole` を作成する **adopts** it: オペレーターは、**すべての**属性がマニフェストと一致するように既存のロールを変更します。省略した属性を含み、デフォルトに強制的に戻されます。特に、*inRoles* にリストされていないメンバーシップは取り消され、省略された `connectionLimit` は `-1` 無制限にリセットされ、ロールに有効期限がある場合、省略された `validUntil` は `infinity` になります。ロールを採用する前に、ロールの現在の属性とメンバーシップを確認し、削除する前に変更されるため、ドロップするだけのロールで `DatabaseRole` をポイントしないでください。

DatabaseRole リソースには、ロールごとの可観測性のための専用の status セクションが含まれています。

```
status:
  applied: true
  observedGeneration: 3
  conditions:
    - lastTransitionTime: "2026-04-04T15:06:23Z"
      message: "2051"
      reason: ChangeDetected
      status: "True"
      type: PasswordSecretChange
```

PasswordSecretChange 条件は、インスタンスマネージャーの内部信号としてオペレーターによって維持されます。そのメッセージには、オペレーターが最後に観察したパスワードシークレットの resourceVersion が含まれ、その値の変更がパスワードの再適用をトリガーします。この条件は、パスワードシークレットが使用されると表示され、passwordSecret が仕様から削除されると削除されます。

DatabaseRole がクラスター仕様で既に管理されている名前をターゲットとしている場合共存と優先 を参照してください、 applied フィールドは false になり、次のメッセージが表示されます。

```
database role is already managed by the CNP cluster
```

レプリカクラスター では、ロールはプライマリクラスターが所有し、ローカルにリコンサイルされません。その場合、インスタンスマネージャーは、ロールを failed ではなく_unknown_として報告します。applied フィールドは説明メッセージとともに設定されない nil のままです。クラスターがプライマリに昇格すると、ロールは通常調整されます。

クライアント証明書の生成

DatabaseRole リソースは、クラスターのクライアント CA によって署名され、Kubernetes Secret に保存されている TLS クライアント証明書のオプトイン生成をサポートしています。これにより、*PostgreSQL `cert* authentication <https://www.postgresql.org/docs/current/auth-cert.html>`_ が有効になります

パスワードの代替としてパスワードを手動でローテーションする必要はなく、プライベートキーは Kubernetes Secret として保存され、クラスターの外部に送信されることはありません。

これを有効にするには、clientCertificate ブロックを仕様に追加します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: DatabaseRole
```

(次のページに続く)

(前のページからの続き)

```
metadata:
  name: role-dante
spec:
  cluster:
    name: cluster-example
  name: dante
  login: true
  clientCertificate:
    enabled: true
  databaseRoleReclaimPolicy: retain
```

ブロックが存在する場合、clientCertificate.enabled はデフォルトの true になるため、clientCertificate: {} はそれを有効にすることと同じです。 enabled: false を設定して、ブロックを維持したまま発行をオフにします。

重要

clientCertificate 発行が有効になっている場合、*login: true* は必要です。オペレーターは検証を介してこれを適用し、それ以外の場合はリソースを拒否します。

オペレーターは、同じ名前空間に<databaserole-name>-client-cert という名前のシークレットを作成します。これには 2 つのキーが含まれています。

Key	Contents
tls.crt	PEM-encoded client certificate, signed by the cluster's client CA
tls.key	PEM-encoded private key

証明書の有効期限は status.clientCertificate.expiration に表示されます。

```
status:
  clientCertificate:
    expiration: "2026-07-01T12:00:00Z"
```

pg_hba.conf の構成

オペレーターは証明書を生成しますが、pg_hba.conf は自動的に変更されません**。ロールが認証できるようにするには、 cert メソッドを使用して hostssl ルールをクラスターに追加する必要があります。

```
spec:
  postgresql:
    pg_hba:
      - hostssl all dante all cert
```

生成されたシークレットを使用した作業的な接続文字列は次のようになります。

```
psql "host=<cluster>-rw.<namespace>.svc port=5432 dbname=<db> user=dante \
  sslcert=/path/to/tls.crt sslkey=/path/to/tls.key \
  sslrootcert=/path/to/ca.crt sslmode=verify-full"
```

リニューアル

クライアント証明書は、オペレーターのグローバル証明書設定を継承します。デフォルトで **90 日** の有効期間で発行され、有効期限から **7 日**以内になると自動的に更新されます。どちらの値もオペレーター全体で、`CERTIFICATE_DURATION` および `EXPIRING_CHECK_THRESHOLD` オペレーター設定を介して構成できます。 `DatabaseRole` ごとに構成はできません。

更新はリコンサイルループによって駆動されます。オペレーターは、証明書の有効期限が近づいているかどうかを確認し、必要に応じて再署名します。`clientCertificate` 発行が有効になっている場合、リコンサイルは少なくとも 1 時間に 1 回スケジュールされるため、トリガーイベントがなくても有効期限のかなり前に更新が発生します。現在の有効期限は常に `status.clientCertificate.expiration` に反映されます。

削除とオプトアウト

Scenario	Result
clientCertificate.enabled set to *false*, or the *clientCertificate* block removed	The cert Secret is deleted; *status.clientCertificate* is cleared
DatabaseRole deleted	The cert Secret is garbage-collected via owner reference, regardless of *databaseRoleReclaimPolicy*

注釈

databaseRoleReclaimPolicy: retain は、生成されたシークレットではなく、PostgreSQL ロールを保持します。シークレットは、オペレーターがロールを管理しているときにのみ意味があるため、削除時に常にクリーンアップされます。

クラスターのクライアント CA シークレットにプライベートキーが含まれていない場合つまり、`spec.certificates.clientCASecret` を介して独自の CA を指定した場合、オペレーターは新しい証明書に署名できません。`status.clientCertificate.message` に理由を記録し、再試行を停止します。

```
status:
  clientCertificate:
    message: client CA secret "my-ca" has no private key; bring-your-own-CA
      clusters require manual certificate management
```

この場合、クライアント証明書を手動で発行および更新する必要があります。

注釈

CNP は、証明書失効リスト CRL を管理しません。証明書の有効期限が切れる前に無効にする必要がある場合は、クラスターのクライアント CA をローテーションします。次のリコンサイルで、オペレーターは、既存の証明書が現在の CA によって署名されていないことを検出し、すべての管理対象クライアント証明書を再発行します。または、証明書のシークレットを削除して、現在の CA によって署名された新しいものをオペレーターに発行させます。

5.17.4 インライン管理対象ロール

クラスター仕様の `managed` スタンザを使用すると、CloudNativePG クラスターの EDB Postgres AI は、`.spec.managed.roles` で指定されたロールの管理を提供します。この機能により、既存のロールの宣言的管理と、新しいロールがまだ存在しない場合は作成が可能になります。

マニフェストの例

宣言的ロール管理を使用するクラスターのマニフェストの例は、ファイル cluster-example-with-roles.yaml にあります。

次に、そのファイルからの抜粋を示します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
spec:
  managed:
    roles:
      - name: dante
        ensure: present
        comment: Dante Alighieri
        login: true
        superuser: false
        inRoles:
          - pg_monitor
          - pg_signal_backend
```

インライン管理対象ロールのステータス

インラインメソッドを使用する場合、`Cluster` ステータスには包括的な概要が含まれます。

```
status:
  managedRolesStatus:
    byStatus:
      reconciled:
        - dante
      reserved:
        - postgres
```

(次のページに続く)

(前のページからの続き)

```
      - streaming_replica
    cannotReconcile:
      petrarca:
        - could not perform UPDATE_MEMBERSHIPS on role petrarca: role "poets" does␣
↪not exist
```

データベースおよび CloudNativePG クラスターの EDB Postgres AI が受け入れられない操作、および人間の介入を必要とする操作用の特別なサブセクション `cannotReconcile` に注意してください。

このセクションでは、オペレーター使用のために予約されているロールと、宣言的管理の下ではない**ロールをカバーし、データベースインスタンスのロールの包括的なビューを提供します。

kubectl **プラグインのサポート** は、 `status` サブコマンドで管理対象ロールのステータスも表示します。

```
Managed roles status
Status                  Roles
- ------                  -----
pending-reconciliation  petrarca
reconciled              app,dante
reserved                postgres,streaming_replica

Irreconcilable roles
Role      Errors
- ---      ------
petrarca  could not perform UPDATE_MEMBERSHIPS on role petrarca: role "poets" does␣
↪not exist
```

5.17.5 インライン管理ロールから DatabaseRole への移行

ロールをインライン `managed.roles` スタンザからスタンドアロンの `DatabaseRole` に中断せずに移動できます。

1. 目的の仕様で `DatabaseRole` を作成します。両方のメソッドは同じを共有します *RoleConfiguration*

構造なので、スタンザをそのままコピーできます。

2. `Cluster` マニフェストの`.spec.managed.roles` から一致するエントリーを削除します。

3. オペレーターは変更を検出し、管理を `DatabaseRole` に引き継ぎます。

両方が存在する間はクラスター仕様が優先されるため**共存と優先** を参照してください、ハンドオーバーはインラインエントリがなくなったときにのみ発生するため、ロールが非管理のままになるウィンドウはありません。

インラインスタンザが `ensure: absent` で削除したロールを変換する場合、`DatabaseRole` は `ensure: absent` をサポートしていないことに注意してください。代わりに **ロール再利用ポリシー** を介した高速削除。`databaseRoleReclaimPolicy: delete` でリソースを削除してロールを削除するか、デフォルトの `retain` をそのままにしておきます。完全な動作については、Removing a role を参照してください。

5.17.6 パスワード管理

宣言的ロール管理機能 `DatabaseRole` およびインライン `managed.roles` スタンザの両方を使用する場合には、ロールパスワードの調整が含まれます。管理対象ロール構成では、オプションでユーザー名とパスワードが保存される**シークレット** の名前を指定できます。

```
passwordSecret:
  name: cluster-example-dante
```

シークレットのタイプは `kubernetes.io/basic-auth` である必要があります。ユーザー名 Kubernetes で通常のように_Base64_でエンコードされたは、パスワードを設定しているロールと一致する必要があります。例

```
apiVersion: v1
data:
  username: ZGFudGU=
  password: ZGFudGU=
kind: Secret
metadata:
  name: cluster-example-dante
  labels:
    k8s.enterprisedb.io/reload: "true"
type: kubernetes.io/basic-auth
```

重要

上記のように、シークレットに`k8s.enterprisedb.io/reload: "true"` のラベルを付けます。ラベル付きシークレットのパスワードの変更はすぐに適用されますが、ラベルのないシークレットの変更は、後続の調整時、たとえば、オペレーターが内部キャッシュを更新するときのみに適用されます。

重要

passwordSecret を使用せずに作成された新しいロールは、PostgreSQL 内に `NULL` パスワードがあります。

PostgreSQL でパスワードを `NULL` に明示的に設定するにはパスワードの更新を単に省略することと区別され、`disablePassword` フィールドを使用します。

```
disablePassword: true
```

> **注釈**
>
> 特定のロールで `passwordSecret` と `disablePassword` の両方を設定するのはエラーです。

PostgreSQL の `VALID UNTIL` ロール属性は、パスワードの有効期限を制御します。 `VALID UNTIL` を指定せずに作成されたロールは、PostgreSQL でデフォルトで NULL を取得します。これは、パスワードの有効期限が無期限であることを意味します。

PostgreSQL は、 `VALID UNTIL` のタイムスタンプタイプを使用します。これには、パスワードの有効期限が無期限であることを示す値`'infinity'` のサポートが含まれます。 PostgreSQL documentation を参照してください。

参考までに。

宣言的ロール管理では、管理対象ロールの `validUntil` 属性がパスワードの有効期限を制御します。`validUntil` は以下のみを取ることができます。

- Kubernetes タイムスタンプ、または
- 省略しますデフォルト `null`

最初のケースでは、指定された `validUntil` タイムスタンプは、ロールの `VALID UNTIL` 属性としてデータベースに設定されます。

2 番目のケース `validUntil` 省略オペレーターは、パスワードの有効期限が決してないことを保証し、PostgreSQL の動作をミラーリングします。具体的には

- 新しいロールの場合、ロール作成ステートメントの `VALID UNTIL` 句を省略します
- 既存のロールの場合、データベースで `VALID UNTIL` が `NULL` に設定されていない場合、`VALID UNTIL` を `infinity` に設定します。これは、PostgreSQL が `ALTER ROLE` SQL ステートメントで `VALID UNTIL NULL` を許可していないためです。

事前ハッシュされたパスワード

MD5 / SCRAM-SHA-256 ハッシュ形式でパスワードを指定することにより、事前に暗号化されたパスワードを提供することもできます。

```
kind: Secret
type: kubernetes.io/basic-auth
metadata:
  name: cluster-example-cavalcanti
  labels:
    k8s.enterprisedb.io/reload: "true"
apiVersion: v1
stringData:
  username: cavalcanti
  password: SCRAM-SHA-256$<iteration count>:<salt>$<StoredKey>:<ServerKey>
```

> **警告**
>
> 上記の例では、 *stringData:* を使用します。ここで、Kubernetes は値をエンコードします。これは、事前ハッシュされたパスワードの最も安全なパスです。 *data:* を使用する必要がある場合は、*printf '%s' "$hash"* | *base64* または `echo -n "$hash" | base64` でバイトを正確にエンコードします。単純な `echo "$hash" | base64` の末尾の改行により、値は SCRAM / MD5 シャドウフォーマットチェックをミスします。そのため、オペレーターはそれを平文として扱うことにフォールバックして再ハッシュし、ログインは機能を停止します。

ロールパスワードは、Kubernetes でシークレットを使用して安全に管理されますが、オペレーターと PostgreSQL 間の SQL パスも懸念されます。PostgreSQL documentation で述べたように

> パスワードは平文でサーバーに送信され、 > クライアントのコマンド履歴またはサーバーログに記録される場合があります

CloudNativePG クラスターの EDB Postgres AI は、2 つの補完的な方法でこのパスを保護します。

1. `CREATE / ALTER ROLE ... PASSWORD '...'` を発行する前に、オペレーター SCRAM-SHA-256 は、クリアテキストパスワードをオペレーターサイド PostgreSQL の観点からクライアントサイドでエンコードします。これは、サーバーログと `pg_stat_statements` または `pgaudit` のような拡張機能からクリアテキストを保持するための標準の PostgreSQL 実践であり、`psql \password` および libpq の `PQencryptPasswordConn` が実行するエンコードと同じです。PostgreSQL が受け取るリテラルは、`pg_authid.rolpassword` に保存されている SCRAM-SHA-256 ベリファイアです。MD5 または SCRAM-SHA-256 シャドウフォームで既に提供されているパスワードは、変更せずに転送されます。
2. 同じ `CREATE /ALTER ROLE` ステートメントは、ステートメントログ `log_statement` とエラーステートメントログ `log_min_error_statement` の両方を一時的に抑制するトランザクション内で実行され、成功と失敗の両方のシナリオでの PostgreSQL ログへの漏洩を防ぎます。

クラスターの Status セクションは、管理対象ロール操作のクエリステートメントを出力しません。

オペレーターサイドのエンコーディングのオプトアウト

パスワードをハッシュする方法を決定するために演算子ではなく PostgreSQL が必要な場合たとえば、`password_encryption = md5` を実行しているクラスターで、basic-auth シークレットにアノテーション `k8s.enterprisedb.io/passwordPassthrough: "enabled"` を設定します。オペレーターは、パスワード値をそのまま転送します。

> **警告**
>
> *k8s.enterprisedb.io/passwordPassthrough* アノテーションは、*Cluster* リソースではなく、**basic-auth Secret**自分自身に設定する必要があります。 *Cluster* に配置しても効果はなく、オペレーターは PostgreSQL に送信する前にパスワードに SCRAM-SHA-256 エンコードを適用し続けます。

> **警告**

k8s.enterprisedb.io/passwordPassthrough: "enabled" を使用すると、オペレーターはシークレットの`password` 値をそのまま転送します。その値がクリアテキストの場合 *password_encryption = md5* クラスターでの一般的なケース *pg_stat_statements* や`pgaudit` などの拡張機能はそれを監視します。これは、PostgreSQL にハッシュ形式を選択させることで予想されるトレードオフです。

PostgreSQL では、場合によっては、コマンドはデータベースが受け入れられず、拒否されます。PostgreSQL documentation on error codes を参照してください。

詳細については。

ロール操作では、このような基本的なエラーが発生する可能性があります。2 つの例

- PostgreSQL に、ロールグループ `poets` のメンバーとしてロール `petrarca` を作成するように依頼しますが、`poets` は存在しません。
- PostgreSQL にロール `dante` を削除するように求めますが、ロール `dante` はデータベース `inferno` の所有者です。

これらの基本的なエラーは、人間の管理者の明確な説明がない限り、データベースまたは CloudNativePG クラスターオペレーターの EDB Postgres AI で修正できません。上記の 2 つの例は、それぞれロール `poets` を作成するか、データベース `inferno` を削除することにより修正できますが、それらは人的エラーが原因で発生した可能性があり、そのような場合、提案された「修正」は間違ったものである可能性があります。

CloudNativePG クラスターの EDB Postgres AI は、**インライン管理対象ロールのステータス** で説明しているように、このような基本的なエラーが発生したときを記録し、クラスターステータスに表示します。

role-examples.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: DatabaseRole
metadata:
  name: cluster-example-reporting
spec:
  cluster:
    name: cluster-example
  name: reporting
  login: true
- --
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: DatabaseRole
metadata:
  name: role-dante
spec:
  cluster:
    name: cluster-example
  name: dante
  comment: my database-side comment
  login: true
```

(次のページに続く)

(前のページからの続き)

```
  superuser: false
  createdb: true
  createrole: false
  inherit: false
  clientCertificate:
    enabled: true
  replication: false
  bypassrls: false
  connectionLimit: 4
  validUntil: "2053-04-12T15:04:05Z"
  inRoles:
    - pg_monitor
    - pg_signal_backend
  passwordSecret:
    name: cluster-example-dante
- --
apiVersion: v1
data:
  username: ZGFudGU=
  password: ZGFudGU=
kind: Secret
metadata:
  name: cluster-example-dante
  labels:
    k8s.enterprisedb.io/reload: "true"
type: kubernetes.io/basic-auth
```

cluster-example-with-roles.yaml

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-with-roles
spec:
  instances: 3
  storage:
    size: 1Gi

  managed:
    roles:
      - name: app
        createdb: true
        login: true
      - name: dante
```

(次のページに続く)

(前のページからの続き)

```
        ensure: present
        comment: my database-side comment
        login: true
        superuser: false
        createdb: true
        createrole: false
        inherit: false
        replication: false
        bypassrls: false
        connectionLimit: 4
        validUntil: "2053-04-12T15:04:05Z"
        inRoles:
          - pg_monitor
          - pg_signal_backend
        passwordSecret:
          name: cluster-example-dante
- --
apiVersion: v1
data:
  username: ZGFudGU=
  password: ZGFudGU=
kind: Secret
metadata:
  name: cluster-example-dante
type: kubernetes.io/basic-auth
```

5.18 PostgreSQL Database management

CloudNativePG クラスターの EDB Postgres AI は、デフォルトで app という名前のアプリケーションデータベースを自動的に作成することにより、PostgreSQL データベースのプロビジョニングを簡素化します。このデフォルトの動作については、空のクラスターをブートストラップする `initdb <空のクラスターをブートストラップする *initdb*>` で説明しています

セクション。

より高度なユースケースとして、CloudNativePG クラスターの EDB Postgres AI は、**宣言的データベース管理**を導入します。これにより、ユーザーは `Database` カスタムリソース定義 CRD を使用して PostgreSQL データベースのライフサイクルを定義および制御できます。この方法は Kubernetes とシームレスに統合し、PostgreSQL データベースを管理するためのスケーラブルで自動化された一貫したアプローチを提供します。

5.18.1 主要なコンセプト

管理範囲

宣言的な Database マニフェスト

次の例は、 Database リソースが Cluster とどのように相互作用するかを示しています。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Database
metadata:
  name: cluster-example-one
spec:
  name: one
  owner: app
  cluster:
    name: cluster-example
  extensions:
    - name: bloom
      ensure: present
```

適用されると、このマニフェストは、 cluster-example PostgreSQL クラスター内の app ロールが所有する one という名前のデータベースを要求する cluster-example-one と呼ばれる Database オブジェクトを作成します。

注釈

API Reference - v1.30.0 を参照してください。

各 Database オブジェクトに定義できる属性の完全なリスト。

Database マニフェストの必須フィールド

- metadata.name 名前空間内の Kubernetes オブジェクトの一意の名前。
- spec.name PostgreSQL に表示されるデータベースの名前。
- spec.owner データベースを所有する PostgreSQL ロール。
- spec.cluster.name ターゲット PostgreSQL クラスターの名前。

Database オブジェクトは特定の Cluster を参照し、データベースを作成する場所を決定する必要があります。クラスターのプライマリインスタンスによって管理され、データベースが必要に応じて作成または更新されます。

警告

spec.cluster フィールドは、作成後は不変です。別の `Cluster` をターゲットにするには、既存のリソースを更新する代わりに、新しい `Database` リソースを作成します。

5.18.2 予約されたデータベース名

PostgreSQL は、 postgres 、template0 、template1 などのデータベースを自動的に作成します。これらの名前は予約されており、CloudNativePG クラスターの EDB Postgres AI の新しい Database オブジェクトには使用できません。

5.18.3 調整とステータス

Database オブジェクトが正常にリコンサイルされると、

- status.applied は true に設定されます。
- status.observedGeneration は、最後に適用された構成の metadata.generation と一致します。

リコンサイルされた Database オブジェクトの例

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Database
metadata:
  generation: 1
  name: cluster-example-one
spec:
  cluster:
    name: cluster-example
  name: one
  owner: app
status:
  observedGeneration: 1
  applied: true
```

リコンシリエーション中にエラーが発生した場合、status.applied は false になり、エラーメッセージは status.message フィールドに含まれます。

5.18.4 データベースの削除

CloudNativePG クラスターの EDB Postgres AI は、データベース削除の 2 つの方法をサポートしています。

1. delete 再利用ポリシーの使用
2. データベースの ensure フィールドを absent に宣言的に設定する

delete 再利用ポリシーを介した削除

databaseReclaimPolicy フィールドは、Database オブジェクトが削除されたときの動作を決定します。

- retain デフォルトデータベースは、手動管理のために PostgreSQL に残ります。
- delete データベースは PostgreSQL から自動的に削除されます。

例

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Database
metadata:
  name: cluster-example-two
spec:
  databaseReclaimPolicy: delete
  name: two
  owner: app
  cluster:
    name: cluster-example
```

この Database オブジェクトを削除すると、cluster-example クラスターから two データベースが自動的に削除されます。

レプリカクラスターでは、データベースは読み取り専用であるため、databaseReclaimPolicy: delete を使用しても、PostgreSQL データベースをドロップせずに、Database オブジェクトを削除するとファイナライザーが解放され、Kubernetes オブジェクトが削除されます。データベースの削除は、それを所有するプライマリクラスターに任されます。

ensure: absent の宣言的設定

データベースを削除するには、次の例のように ensure フィールドを absent に設定します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Database
metadata:
  name: cluster-example-database-to-drop
spec:
  cluster:
    name: cluster-example
  name: database-to-drop
  owner: app
  ensure: absent
```

このマニフェストは、 database-to-drop データベースが cluster-example クラスターから削除されることを保証します。

5.18.5 データベース内の拡張機能の管理

> **注釈**
>
> 拡張機能はグローバルオブジェクトではなくデータベーススコープですが、CloudNativePG クラスターの EDB Postgres AI は、それらを管理するための宣言型インターフェイスを提供します。特定の拡張機能のインストールにはスーパーユーザー権限が必要になる場合があるため、このアプローチが必要です。CloudNativePG クラスターの EDB Postgres AI は、デフォルトで無効にすることをお勧めします。

この API を活用することにより、ユーザーは昇格した特権を必要とせずに、スケーラブルで制御された方法で拡張機能を効率的に管理できます。

CloudNativePG クラスターの EDB Postgres AI は、ターゲットデータベース内の PostgreSQL 拡張機能の管理を簡素化および自動化します。

この機能を有効にするには、次の例に示すように、拡張仕様のリストを使用して `spec.extensions` フィールドを定義します。

```
#  ...

spec:
  extensions:
    - name: bloom
      ensure: present

#  ...
```

各拡張エントリは、次のプロパティをサポートしています。

- `name` *(**必須**)* 拡張機能の名前。
- `ensure` 拡張機能がデータベースに存在するかどうかを指定します。
- `present` 拡張機能がインストールされていることを確認しますデフォルト
- `absent` 拡張機能が削除されたことを確認します。
- `version` インストールまたはアップグレードする拡張機能の特定のバージョン。
- `schema` 拡張機能をインストールするスキーマ。

注釈

CloudNativePG クラスターの EDB Postgres AI は、次の PostgreSQL の SQL コマンドを使用して拡張機能を管理します。 CREATE EXTENSION 、 DROP EXTENSION 、 ALTER EXTENSION

`UPDATE TO` および `SET SCHEMA` に限定します。

オペレーターは、`spec.extensions` に明示的にリストされている拡張子のみをリコンサイルします。このリストで指定されていない既存の拡張子は変更されないままです。

警告

宣言的拡張機能管理を導入する前、CloudNativePG クラスターの EDB Postgres AI は、構成から拡張機能を作成する簡単な方法を提供していませんでした。これに対処するために、 **マネージド拡張機能**

機能が導入され、`pg_stat_statements` のような主要な拡張機能の自動かつ透過的な管理が可能になります。現在、`Database` CRD の拡張機能サポートと管理対象拡張機能の間で競合がないことを確認するのは

あなたの責任です。

5.18.6 データベースでのスキーマの管理

> **注釈**
>
> PostgreSQL のスキーマ管理は、グローバルオブジェクトの管理に主に焦点を当てている CloudNativePG クラスターの EDB Postgres AI の例外です。スキーマはデータベース内に存在するため、通常はアプリケーション開発プロセスの一部として管理されます。ただし、CloudNativePG クラスターの EDB Postgres AI は、主にスキーマ内での拡張機能展開のサポートを完了するために、スキーマ管理の宣言型インターフェイスを提供します。

CloudNativePG クラスターの EDB Postgres AI は、ターゲットデータベース内の PostgreSQL スキーマの管理を簡素化および自動化します。

この機能を有効にするには、次の例に示すように、スキーマ仕様のリストを使用して `spec.schemas` フィールドを定義します。

```
#  ...

spec:
  schemas:
    - name: app
      owner: app

#  ...
```

各スキーマエントリは、次のプロパティをサポートしています。

- `name` *(**必須**)* スキーマの名前。
- `owner` スキーマの所有者。
- `ensure` データベースにスキーマが存在するかどうかを指定します。
- `present` スキーマがインストールされていることを確認しますデフォルト。
- `absent` スキーマが削除されたことを確認します。

> **注釈**
>
> CloudNativePG クラスターの EDB Postgres AI は、次の PostgreSQL の SQL コマンドを使用してスキーマを管理します。 CREATE SCHEMA 、 DROP SCHEMA 、 ALTER SCHEMA 。

5.18.7 データベースでの外部データラッパー FDW の管理

> **注釈**
>
> 外部データラッパー FDW は、データベーススコープのオブジェクトであり、通常、作成または変更するにはスーパーユーザー権限が必要です。 CloudNativePG クラスターの EDB Postgres AI は、FDW を管理するための宣言型 API を提供し、ユーザーが SQL コマンドを直接実行したり、特権をエスカレートしたりせずに、制御された Kubernetes ネイティブの方法で FDW を定義および保守できます。

CloudNativePG クラスターの EDB Postgres AI は、宣言的構成を使用して、ターゲットデータベース内の PostgreSQL 外部データラッパーのシームレスで自動管理を可能にします。

この機能を有効にするには、次の例に示すように、FDW 仕様のリストを使用して `spec.fdws` フィールドを定義します。

```
#  ...

spec:
  fdws:
    - name: postgres_fdw
      usage:
        - name: app
          type: grant

#  ...
```

各 FDW エントリは、次のプロパティをサポートしています。

- `name` 外部データラッパーの名前 **必須)** 。
- `ensure` データベースで FDW が `present` または `absent` であるかどうかを示しますデフォルトは `present` です。
- `handler` FDW が使用するハンドラーファンクションの名前。指定しない場合、FDW 拡張機能がある場合に定義されたデフォルトハンドラーが使用されます。
- `validator` FDW が使用するバリデーターファンクションの名前。指定しない場合、FDW 拡張機能がある場合に定義されたデフォルトのバリデーターが使用されます。
- `owner` FDW の所有者 **スーパーユーザーである必要があります** 。
- `usage` 次のフィールドを含む FDW の `USAGE` 権限のリスト。
- `name` 使用権限を付与するまたは取り消すロールの名前 **(必須)** .
- `type` : 使用許可の種類。 `grant` および `revoke` をサポートします。
- `options` 管理する FDW 固有のオプションのマップ。各キーはオプションの名前です。各オプションは次のフィールドをサポートします。
- `value` オプションの文字列値。

- ensure オプションが present または absent であるかどうかを示します。

> **注釈**
>
> *handler* と `validator` は両方ともオプショナルであり、指定しない場合、FDW 拡張機能存在する場合に定義されたデフォルトのハンドラーとバリデーターが使用されます。 *handler* または `validator` を `"-"` に設定すると、それぞれ FDW からハンドラーまたはバリデータが削除されます。これは PostgreSQL の規則に従います。ここで、「-」はハンドラーまたはバリデーターが存在しないことを示します。

> **警告**
>
> PostgreSQL は、外部データラッパーの所有権を**スーパーユーザー権限のみを持つロール**に制限しています。 PostgreSQL は外部データラッパーの非スーパーユーザーの所有権を許可していないため、所有権を非スーパーユーザーアプリロールなどに割り当てようとすると無視されるか、拒否されます。デフォルトでは、これらは `postgres` ユーザーが所有しています。

オペレーターは、spec.fdws に明示的にリストされた FDW のみをリコンサイルします。このリストで宣言されていない既存の FDW はそのまま残されます。

> **注釈**
>
> CloudNativePG クラスターの EDB Postgres AI は、 PostgreSQL のネイティブ SQL コマンド CREATE FOREIGN DATA WRAPPER 、 ALTER FOREIGN DATA WRAPPER 、および DROP FOREIGN DATA WRAPPER を使用して FDW を管理します。 *ALTER* コマンドは、オプションの更新をサポートしています。

データベース内の外部サーバーの管理

CloudNativePG クラスターの EDB Postgres AI は、宣言的構成を使用して、ターゲットデータベース内の PostgreSQL 外部サーバーのシームレスな自動管理を提供します。

外部サーバーは、外部データラッパー FDW が外部データソースにアクセスするために使用する接続の詳細をカプセル化します。ユーザー固有の接続の詳細については、user mappings を定義できます。

この機能を有効にするには、外部サーバー仕様のリストを使用して Database リソースの spec.servers フィールドを宣言します。例

```
#  ...

spec:
  servers:
    - name: angus
      fdw: postgres_fdw
      ensure: present
      usage:
```

(次のページに続く)

(前のページからの続き)

```
        - name: app
          type: grant
      options:
        - name: host
          value: angus-rw
        - name: dbname
          value: app

#  ...
```

各外部サーバーエントリは、次のプロパティをサポートしています。

- name 外部サーバーの名前 **必須)** 。
- fdw サーバーが属する外部データラッパーの名前 **必須** 。
- ensure 外部サーバーをデータベース内の present または absent にするかどうかデフォルト present 。
- usage 次のフィールドを含む外部サーバーの USAGE 権限のリスト。
- name 使用権限を付与するまたは取り消すロールの名前 **(必須)** .
- type 使用権限の種類。 grant および revoke をサポートします。
- options FDW 固有のオプション仕様のリスト。リスト内の各エントリは、次のキーをサポートします。
- name オプションの名前 **必須** .
- value オプションの文字列値。
- ensure オプションが present または absent かどうかを示します。

CloudNativePG クラスターの EDB Postgres AI は、PostgreSQL のネイティブ SQL コマンド CREATE SERVER 、 ALTER SERVER 、および DROP SERVER を使用して外部サーバーを管理します。ALTER SERVER コマンドは、サーバーオプションを更新するために使用されます。

オペレーターは、spec.servers に明示的にリストされている外部サーバーのみをリコンサイルします。このリストに含まれない既存のサーバーは変更されないままです。

5.18.8 制限と注意事項

データベースの名前の変更

CloudNativePG クラスターの EDB Postgres AI は、PostgreSQL の CREATE DATABASE および ALTER DATABASE に準拠していますが

コマンド、**データベースの名前の変更はサポートされていません**。既存の Database オブジェクトの spec.name を変更しようとすると、Kubernetes によって拒否されます。

データベースの作成と変更

- 新しいデータベースの場合、CloudNativePG クラスターの EDB Postgres AI は `CREATE DATABASE` ステートメントを使用します。
- 既存のデータベースの場合、`ALTER DATABASE` を使用して変更を適用します。

これら 2 つの Postgres コマンドにはいくつかの違いがあることに注意することが重要です。特に、`ALTER` によって受け入れられるオプションは、`CREATE` によって受け入れられるオプションのサブセットです。

> **警告**
>
> エンコードや照合設定などの一部のフィールドは、PostgreSQL で不変です。既存のデータベースのこれらのフィールドを変更しようとすると、無視されます。

レプリカクラスター

レプリカには書き込み特権がないため、レプリカクラスターで宣言されたデータベースオブジェクトは適用できません。これらのオブジェクトは、レプリカが昇格するまで保留状態のままです。

競合の解決

同じ名前空間内の 2 つの `Database` オブジェクトが同じ PostgreSQL データベースつまり同一の `spec.name` および `spec.cluster.name` を管理している場合、2 番目のオブジェクトは拒否されます。

ステータスメッセージの例

```
status:
  applied: false
  message: reconciliation error: database "one" is already managed by Database object
↪"cluster-example-one"
```

Postgres バージョンの違い

CloudNativePG クラスターの EDB Postgres AI は、PostgreSQL の機能に準拠しています。たとえば、PostgreSQL 16 で導入された `ICU_RULES` のような機能は、以前のバージョンでは使用できません。PostgreSQL からのエラーは、`Database` オブジェクトの `status` に反映されます。

手動変更

CloudNativePG クラスターの EDB Postgres AI は、データベースへの手動変更を上書きしません。リコンサイルが完了すると、`Database` オブジェクトは、`metadata.generation` が変更しない限り再適用されないため、PostgreSQL を直接変更するための柔軟性が得られます。

5.19 Tablespaces

テーブルスペースは、データベース管理システムで堅牢で広く採用されている機能です。データの物理的および論理モデリングを分離することにより、データベースの垂直スケーラビリティを強化する強力な手段を提供します。基本的に、これは物理データベースモデリングの技術として機能し、個別のストレージ上のマルチプルのボリューム全体に I/O 操作を効率的に分散できます。これにより、並列ディスク上の読み取り/書き込み操作を介してパフォーマンスが最適化されます。

データベース業界のコンテキストでは、テーブルスペースは、特に論理データベースモデリング技術であるテーブルパーティショニングと組み合わせた場合、戦略的な役割を果たします。これらは大規模なデータベースの管理に役立つことが証明されており、インデックスからテーブルを分離したり、一時的な操作を実行したりするようなタスクにも使用されます。

PostgreSQL のテーブルスペースは、2005 バージョン 8.0 から重要な役割を果たしていますが、宣言的パーティショニングは 2017 バージョン 10 に導入されました。その結果、テーブルスペースは、サポートされている PostgreSQL のすべてのリリースにシームレスに統合されています。

PostgreSQL documentation on tablespaces からの引用

> テーブルスペースを使用することにより、管理者は PostgreSQL インストールのディスクレイアウトを制御できます。これは、少なくとも 2 つの方法で役立ちます。> > - まず、クラスターが初期化されたパーティションまたはボリュームが領域不足で実行され、拡張できない場合、別のパーティションにテーブルスペースを作成し、システムが再構成できるまで使用できます。> - 第二に、テーブルスペースにより、管理者はデータベースオブジェクトの使用パターンに関する知識を使用して、パフォーマンスを最適化できます。

5.19.1 宣言的テーブルスペース

CloudNativePG クラスターの EDB Postgres AI は、2 つの異なるレベルで動作する_declarativetablespaces_を介して PostgreSQL テーブルスペースのサポートを提供します。

- Kubernetes、PGDATA および WAL ボリュームが処理される方法と同じように、永続ボリューム要求を管理します
- PostgreSQL、 PostgreSQL インスタンスで `TABLESPACE` グローバルオブジェクトを管理する

Kubernetes エコシステムの一部であるため、CloudNativePG クラスターの EDB Postgres AI の宣言的テーブルスペースは、永続ボリューム要求および永続ボリュームを活用して実装されます。クラスターで定義された各テーブルスペースは、独自の永続ボリュームに格納されます。CloudNativePG クラスターの EDB Postgres AI は、PVC の生成を担当します。正規化された場所のインスタンスポッドに必要なボリュームをマウントし、プライマリでアクティブにする前にレプリカがテーブルスペースをサポートする準備ができていることを確認します。

要求されたときにストレージが利用可能な場合、クラスターの作成時にテーブルスペースを設定するか、後で追加できます。現在、それらを削除することはできません。ただし、この制限は、CloudNativePG クラスターの EDB Postgres AI の将来のマイナーまたはパッチバージョンで対処される予定です。

5.19.2 宣言的テーブルスペースの使用

宣言的テーブルスペースの使用は簡単です。完全な例は cluster-example-with-tablespaces.yaml にあります。

これらを使用するには、新しいまたは既存の `Cluster` リソースで新しい `tablespaces` スタンザを使用します。

```
spec:
  instances: 3

  # ...

  tablespaces:
    - name: tbs1
      storage:
        size: 1Gi
    - name: tbs2
      storage:
        size: 2Gi
    - name: tbs3
      storage:
        size: 2Gi
```

各テーブルスペースには、生成された PVC のサイズとストレージクラスを構成できる独自のストレージセクションがあります。したがって、管理者は、**ストレージクラスとテーブルスペース** で説明しているように、さまざまな種類のワークロードに異なるストレージクラスの使用を計画できます。

CloudNativePG クラスターの EDB Postgres AI は、高可用性 Postgres クラスターの各インスタンスの永続ボリューム要求を作成します。プロビジョニングが完了すると、各ポッドにマウントします。次に、`CREATE TABLESPACE` コマンドを使用してプライマリ PostgreSQL インスタンスに `tbs1` 、`tbs2` 、および `tbs3` 表領域が作成されることを確認します。このプロセスは迅速で、これは Postgres に反映されます。

```
app=# SELECT oid, spcname FROM pg_tablespace;
  oid  |      spcname
- ------+--------------------
  1663 | pg_default
  1664 | pg_global
 16387 | tbs1
 16388 | tbs2
 16389 | tbs3
(5 rows)
```

すぐに使用を開始できます。

```
app=# CREATE TABLE fibonacci(num INTEGER) TABLESPACE tbs1;
CREATE TABLE
```

クラスターステータスには、テーブルスペースのセクションがあります。

```
status:

  <- snipped ->
  tablespacesStatus:
  - name: atablespace
    state: reconciled
  - name: another_tablespace
    state: reconciled
  - name: tablespacea1
    state: reconciled
```

5.19.3 ストレージクラスとテーブルスペース

PGDATA および WAL ボリュームと同じように、テーブルスペースにさまざまなストレージクラスを使用できます。これは、リソースを最適化し、データアクセスの使用量と予想に基づいてストレージのパフォーマンスとコストのバランスを保つ便利な方法です。

この例は、機能を説明するのに役立ちます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: yardbirds
spec:
  instances: 3

  storage:
    size: 10Gi
  walStorage:
    size: 10Gi
  tablespaces:
    - name: current
      storage:
        size: 100Gi
        storageClass: fastest
    - name: this_year
      storage:
        size: 500Gi
        storageClass: balanced
```

yardbirds クラスター例は、3 つの異なるストレージクラスを使用して 4 つの永続ボリューム要求を要求します。

- デフォルトのストレージクラス – PGDATA および WAL ボリュームによって使用されます。
- fastest – current テーブルスペースによって使用され、データベースで最もアクティブで要求の高いデータセットを保存します。

- balanced – this_year 表領域によって使用され、ユーザーがほとんどアクセスせず、パフォーマンスの期待が最も高くないデータの古いパーティションを保存します。

次に、ホリゾンタルテーブルパーティショニングを利用して、current 表領域に当月のテーブル 2023 年 12 月のファクトなどを作成できます。

```
CREATE TABLE facts_202312 PARTITION OF facts
    FOR VALUES FROM (2023-12-01) TO (2024-01-01)
    TABLESPACE current;
```

5.19.4 テーブルスペースの所有権

デフォルトでは、特に指定しない限り、表領域は.spec.bootstrap.initdb.owner で定義されているように、app アプリケーションユーザーが所有します。詳細は、*Bootstrap a new cluster* を参照してください。このデフォルトの動作は、ほとんどのマイクロサービスデータベースのユースケースで動作します。

次の抜粋のように、 owner スタンザで表領域の所有者、たとえば postgres ユーザーを設定できます。

```
#  ...

tablespaces:
  - name: clapton
    owner:
      name: postgres
    storage:
      size: 1Gi
```

存在しない所有者でテーブルスペースを定義すると、CloudNativePG クラスターの EDB Postgres AI はテーブルスペースを作成できず、これをクラスターステータスに反映します。

```
spec:
  instances: 3

  # ...

  tablespaces:
    - name: tbs1
      storage:
        size: 1Gi
    - name: tbs2
      storage:
        size: 2Gi
    - name: tbs3
      owner:
        name: badhombre
      storage:
        size: 2Gi
```

(次のページに続く)

(前のページからの続き)

```
      status:

<- snipped ->
tablespacesStatus:
- name: tbs1
  status: reconciled
- name: tbs2
  status: reconciled
- error: while creating tablespace tbs3: ERROR: role "badhombre" does
    not exist (SQLSTATE 42704)
  name: tbs3
  status: pending
```

5.19.5 バックアップ リカバリー

CloudNativePG クラスターの EDB Postgres AI は、オブジェクトストアとボリュームスナップショットの両方でテーブルスペースおよび相対テーブルスペースマップのバックアップを処理します。

> **警告**
>
> デフォルトでは、バックアップはレプリカノードから取得されます。クラスターで表領域を作成した直後にバックアップを作成すると、レプリカからの表領域の不完全なビュー、したがって不完全なバックアップが生成される場合があります。遅延は、次の調整で最大 5 分で解決されます。

> **警告**
>
> 既存のクラスターで表領域を追加または削除すると、新しいベースバックアップを作成するまで WAL からのリカバリは失敗します。

テーブルスペースを含むクラスターにベースバックアップがあると、そこから新しいクラスターを復元できます。復旧側については、復旧したデータベースの `Cluster` 定義に表領域の正確なリストが含まれていることを確認するのは自分の責任です。

5.19.6 レプリカクラスター

レプリカクラスターには、オリジンと同じ表領域定義が必要です。その理由は、`CREATE TABLESPACE` のようなテーブルスペース管理コマンドは WAL ログに記録され、物理レプリケーションクライアントストリーミングまたは WAL シッピング経由で再生されるためです。

レプリカクラスターに同じ名前のテーブルスペースの同じリストがあることを確認するのはあなたの責任です。ストレージクラスとサイズは異なる場合があります。

例

```
spec:
  # ...
  bootstrap:
    recovery:
      # ... your selected recovery method

  tablespaces:
    - name: tbs1
      storage:
        size: 1Gi
    - name: tbs2
      storage:
        size: 2Gi
    - name: tbs3
      storage:
        size: 2Gi
```

5.19.7 一時テーブルスペース

PostgreSQL では、1 つ以上の一時表領域を定義して、CREATE コマンドが明示的に表領域を指定しない場合に、一時オブジェクト一時テーブルと一時テーブルのインデックスを作成したり、大規模なデータセットのソートなどの目的で一時ファイルを作成したりできます。一時表領域が指定されない場合、PostgreSQL は、現在メインの PGDATA ボリュームであるデータベースのデフォルトの表領域を使用します。

複数の一時テーブルスペースを指定すると、PostgreSQL は、トランザクションで最初に一時オブジェクトを作成する必要があるときにランダムに 1 つを選択します。次に、リストを順番に反復します。

一時テーブルスペースは、バックアップに関して通常のテーブルスペースと同様に動作します。

CloudNativePG クラスターの EDB Postgres AI は、 temp_tablespaces PostgreSQL パラメーターにテーブルスペースを追加するかどうかを決定する.spec.tablespaces[*].temporary オプションを提供し、明示的なテーブルスペースの割り当てがない一時データを保存できるようになります。

```
spec:
  [...]
  tablespaces:
    - name: atablespace
      storage:
        size: 1Gi
      temporary: true
```

これらは初期化時に作成するか、後で追加できますが、ローリング更新が必要です。temporary: true/false オプションは、 temp_tablespaces オプションの表領域のリストに表領域名を追加または削除します。この変更では、PostgreSQL の再起動は必要ありません。

一時表領域は通常の表領域として機能することもできますが、ユーザーが一時的な操作に使用しながら通常のデータをホストすることもできますが、2 つのワークロードを混合しないことをお勧めします。

*PostgreSQL documentation on `temp_tablespaces* <https://www.postgresql.org/docs/current/runtime-config-client.html#GUC-TEMP-TABLESPACES>`_ を参照してください。

詳細については。

5.19.8 kubectl プラグインのサポート

kubectl status プラグインには、表領域のステータス、所有者、一時フラグ、エラーなどの便利な概要を提供する表領域専用のセクションが含まれています。

```
[...]

Tablespaces status
Tablespace          Owner  Status      Temporary  Error
- ---------          -----  ------      ---------  -----
atablespace         app    reconciled  true
another_tablespace  app    reconciled  true
tablespacea1        app    reconciled  false

Instances status
[...]
```

5.19.9 制限事項

現在、CloudNativePG Cluster クラスターの既存の EDB Postgres AI からテーブルスペースを削除することはできません。

cluster-example-with-tablespaces.yaml

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-with-tablespaces
spec:
  instances: 3

  storage:
    size: 1Gi

  tablespaces:
    - name: atablespace
      storage:
        size: 1Gi
        storageClass: standard
      temporary: true
    - name: another_tablespace
```

(次のページに続く)

(前のページからの続き)

```
    storage:
      size: 2Gi
      storageClass: standard
    temporary: true
  - name: tablespacea1
    storage:
      size: 2Gi
      storageClass: standard
```

5.20 Storage

ストレージは、データベースワークロードで最も重要なコンポーネントです。ストレージは常に利用でき、スケーリング、パフォーマンスが良く、一貫性と耐久性を保証する必要があります。仮想マシンやベアメタルなどの従来の環境に適用されるのと同じ期待と要件は、Kubernetes によって管理されるコンテナコンテキストでも有効です。

ストレージへのアクセスには、主に 2 つの方法があります。

- **ネットワーク** – 直接または間接的に。(Kubernetes を実行しているホストにローカルにマウントされた NFS ボリュームを考えます。)
- **ローカル** – ポッドが実行されているノードに直接接続されます。これには、Kubernetes のベアメタルインストールに直接接続されたディスクも含まれます。

Kubernetes で最も一般的な使用パターンであるネットワークストレージは、従来の環境で経験する可能性のあるスループットとレイテンシと同じ問題を示します。これらの問題は、複数のアプリケーションとの I/O 競合によりパフォーマンス結果のばらつきが増加する共有環境では強調される可能性があります。

ローカルストレージは、シェアードナッシングアーキテクチャを有効にします。これは、より高く予測可能なパフォーマンスを保証するため、高トランザクションで非常に大規模なデータベース VLDB ワークロードに適しています。

> **警告**
>
> CloudNativePG クラスターの EDB Postgres AI を使用して PostgreSQL クラスターを展開する前に、使用しているストレージがデータベースワークロードに推奨されていることを確認してください。 fio などのツールを使用してストレージをベンチマークし、パフォーマンスの期待値を明確に設定することをお勧めします

次に、 pgbench を使用したデータベース。

> **注釈**
>
> CloudNativePG クラスターの EDB Postgres AI は、データの永続性の管理に `StatefulSet` を使用しません。むしろ、PVC を直接管理します。詳細については、 *Custom Pod Controller* を参照してください。

5.20.1 バックアップ リカバリー

CloudNativePG クラスターの EDB Postgres AI は、バックアップとリカバリーの両方のボリュームスナップショットをサポートしているため、ストレージソリューションを選択するときに、特に非常に大規模なデータベースを管理する場合、この側面も考慮することをお勧めします。

スナップショット機能を提供します。

5.20.2 CloudNativePG クラスターの EDB Postgres AI のベンチマーク

運用環境にデータベースを展開する前に、制御された Kubernetes 環境で CloudNativePG クラスターの EDB Postgres AI をベンチマークすることをお勧めします。*Benchmarking* のガイドラインに従ってください。

手短に言うと、2 つのレベルで動作することをお勧めします。

- シークエンシャル読み取り、シークエンシャル書き込み、ランダム読み取り、ランダム書き込みのスループットなど、データベースワークロードの関連メトリックを使用して、fio を使用して基になるストレージのパフォーマンスを測定する
- PostgreSQL で配布されるデフォルトのベンチマークツール、pgbench を使用してデータベースのパフォーマンスを測定する

自分のシステムを知り、ベンチマークします。

CloudNativePG クラスターの EDB Postgres AI では、保存時の暗号化が可能です。オペレーターは、それを基になるストレージクラスに委任します。この重要なセキュリティ機能の詳細については、ストレージクラスを参照してください。

5.20.3 Persistent Volume Claim Claim PVC

オペレーターは、PGDATA を保存することを目的に、各 PostgreSQL インスタンスの PVC を作成します。次に、それを各ポッドにマウントします。

さらに、次のクラスターの作成をサポートしています。

- *WAL のボリューム* で説明しているように、PostgreSQL WAL を保存する別の PVC
- 宣言的テーブルスペース で説明しているように、PostgreSQL テーブルスペース用に予約された追加の個別ボリューム

CloudNativePG クラスターの EDB Postgres AI では、単一の PostgreSQL インスタンスに接続されたボリュームは_PVC group_として定義されます。

5.20.4 ストレージクラスを介した構成

PostgreSQL クラスのストレージを構成する最も簡単な方法は、次の例のように、特定のサイズのストレージを要求することです。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
```

(次のページに続く)

(前のページからの続き)

```
  name: postgresql-storage-class
spec:
  instances: 3
  storage:
    size: 1Gi
```

前の構成を使用すると、生成された PVC はデフォルトのストレージクラスで満たされます。ターゲット Kubernetes クラスターにデフォルトのストレージクラスがない場合、または既知のストレージクラスで PVC を満たす必要がある場合でも、カスタムリソースに設定できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: postgresql-storage-class
spec:
  instances: 3
  storage:
    storageClass: standard
    size: 1Gi
```

5.20.5 PVC テンプレートを介した構成

生成された PVC をさらにカスタマイズするには、次の例のように、カスタムリソース内に PVC テンプレートを提供できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: postgresql-pvc-template
spec:
  instances: 3

  storage:
    pvcTemplate:
      accessModes:
        - ReadWriteOnce
      resources:
        requests:
          storage: 1Gi
      storageClassName: standard
      volumeMode: Filesystem
```

5.20.6 WAL のボリューム

デフォルトでは、PostgreSQL はすべてのデータをいわゆる PGDATA ディレクトリに保存します。 PGDATA 内のコアディレクトリの 1 つが pg_wal で、これには、データベースで発生したトランザクション変更のログがセグメントファイルの形式で含まれています。pg_wal は、PostgreSQL では歴史的に pg_xlog として知られています。

> **注釈**
>
> 通常、各セグメントのサイズは 16MB ですが、*walSegmentSize* オプションを使用してサイズを構成できます。 空のクラスターをブートストラップする `initdb <空のクラスターをブートストラップする *initdb*>` で説明しているように、このオプションはクラスターの初期化時に適用されます。

ほとんどの場合、PGDATA が存在するのと同じボリュームに pg_wal があっても問題ありません。ただし、WAL を別のボリュームに保存すると、いくつかの利点があります。

- **I/O パフォーマンス** – WAL ファイルを PGDATA とは別のストレージに保存することにより、PostgreSQL は WAL 操作通常はシークエンシャル書き込みとデータファイルテーブルやインデックスなどの並列 I/O を利用でき、したがって、垂直方向を向上させることができますスケーラビリティ。
- **信頼性の向上** – WAL ファイルに専用のディスク領域を予約することにより、PGDATA ボリュームの領域を使い果たしても WAL の書き込みが妨げられることはありません。この動作により、PostgreSQL プライマリが正しくシャットダウンされます。
- **より細かい制御** – PGDATA と pg_wal の両方専用の領域の量を定義でき、WAL configuration を微調整できます。

とチェックポイント、さらにはコスト最適化のために別のストレージクラスを使用します。

- **I/O モニタリングの向上** – PGDATA と pg_wal の両方で負荷とディスク使用量を常にモニタできます。たとえば、PGDATA のサイズ変更が必要な場合に通知するアラートを設定することもできます。

詳細については、PostgreSQL ドキュメントを参照してください。

.spec.walStorage オプションを使用して、WAL の別のボリュームを追加できます。 storage フィールドで説明したものと同じルールに従い、専用の PVC をプロビジョニングします。例

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: separate-pgwal-volume
spec:
  instances: 3
  storage:
    size: 1Gi
  walStorage:
    size: 1Gi
```

5.20.7 テーブルスペースのボリューム

CloudNativePG クラスターの EDB Postgres AI は、宣言的テーブルスペースをサポートしています。それぞれが単一の PostgreSQL テーブルスペース専用の 1 つ以上のボリュームを追加できます。詳細は、宣言的テーブルスペース を参照してください。

5.20.8 ボリュームの拡張

Kubernetes が expanding PVCs を許可する API を公開

それはデフォルトで有効になっています。ただし、基になる `StorageClass` でサポートされている必要があります。

特定の `StorageClass` がボリューム拡張をサポートしているかどうかを確認するには、ストレージクラスの `allowVolumeExpansion` フィールドを読み取ることができます。

```
$ kubectl get storageclass -o jsonpath={$.allowVolumeExpansion} premium-storage
true
```

ボリューム拡張 Kubernetes 機能の使用

ストレージクラスがボリューム拡張をサポートしている場合、 `Cluster` のサイズ要件を変更できます。オペレーターはすべての PVC に変更を適用します。

`StorageClass` が online volume resizing をサポートしている場合、変更はすぐにポッドに適用されます。基になるストレージクラスがそれをサポートしていない場合、ポッドを削除してサイズ変更をトリガーする必要があります。

続行する最良の方法は、一度に 1 つのポッドを削除し、レプリカから始めて、各ポッドがバックアップされるまで待機することです。

ストレージの再作成

ストレージクラスがボリューム拡張をサポートしていない場合でも、別の PVC でクラスターを再生成できます。増加したストレージに新しい PVC を割り当て、データベースをそこに移動します。この操作は、クラスターに複数のノードが含まれる場合にのみ実行可能です。

その際、次の例のように、`resizeInUseVolumes` フラグを無効にすることにより、オペレーターが既存の PVC を変更しないようにする必要があります。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: postgresql-pvc-template
spec:
  instances: 3

  storage:
    storageClass: standard
```

(次のページに続く)

(前のページからの続き)

```
    size: 1Gi
    resizeInUseVolumes: False
```

クラスター全体を別の記憶領域に移動するには、すべての PVC とすべてのポッドを再作成する必要があります。次の例のように、3 つのレプリカを含むクラスターがあるとします。

```
$ kubectl get pods
NAME                READY   STATUS    RESTARTS   AGE
cluster-example-1   1/1     Running   0          2m37s
cluster-example-2   1/1     Running   0          2m22s
cluster-example-3   1/1     Running   0          2m10s
```

別の PVC を使用してクラスターを再作成するには、クラスター定義を編集して `resizeInUseVolumes` を無効にできます。次に、すべてのインスタンスを別の PVC で再作成します。

たとえば、 `cluster-example-3` のストレージを再作成します。

```
$ kubectl delete pvc/cluster-example-3 pod/cluster-example-3
```

たとえば、WAL ストレージ専用の PVC が存在する場合

```
$ kubectl delete pvc/cluster-example-3 pvc/cluster-example-3-wal pod/cluster-example-3
```

それが完了したら、オペレーターは、サイズ変更した PVC を使用して別のレプリカの作成を調整します。

```
$ kubectl get pods
NAME                          READY   STATUS      RESTARTS   AGE
cluster-example-1             1/1     Running     0          5m58s
cluster-example-2             1/1     Running     0          5m43s
cluster-example-3-join-v2     0/1     Completed   0          17s
cluster-example-3             1/1     Running     0          10s
```

5.20.9 ボリュームの削減

Kubernetes は PVC を縮小するための API を提供しておらず、CloudNativePG クラスターの検証 Webhook の EDB Postgres AI は、`.spec.storage.size` 、`.spec.walStorage.size` 、または`.spec.tablespaces` のテーブルスペースストレージサイズを削減しようとする試みを拒否します。クラスターのストレージは引き続き削減できますが、以下で説明するように、より小さいボリュームで各インスタンスを再作成する場合にのみです。

> **警告**
>
> CloudNativePG クラスターの EDB Postgres AI は、自動ボリュームの縮小は、誤って実行するとデータの損失につながる可能性のある繊細な操作であるため、サポートしていません。当分の間、これは、以下で説明する監視対象手順を介して手動でのみ達成できます。この手順では、検証 Webhook を一時的に無効にする必要があります。検証が無効になっている間、オペレーターは、安全でないまたは破壊的

なものを含む、通常は拒否される仕様変更を受け入れます。自己の責任で注意して続行し、できるだけ早く検証を再度有効にします。

開始する前に、クラスターの現在のデータ、WAL、およびテーブルスペースデータが新しい小さいサイズに快適に収まることを確認してください。そうしないと、より小さいボリュームで再作成されたインスタンスが再参加に失敗するか、すぐに領域が不足する場合があります。

1. `Cluster` に `k8s.enterprisedb.io/validation: disabled` アノテーションを設定して検証 Webhook を無効にし、`.spec.storage.size` および存在する場合は`.spec.walStorage.size` または`.spec.tablespaces` のテーブルスペースストレージサイズを新しい小さい値に設定し、`.spec.instances` を 1 つ増やしてスペアインスタンスを提供しますロールアウト。

2. `k8s.enterprisedb.io/validation` アノテーションを削除または `enabled` に設定して、検証を再度有効にします。新しい小さいサイズがスペックに保存され、この時点からオペレーターが再作成するすべてのインスタンスに適用されます。既存のインスタンスは、現在のボリュームを保持します。それぞれのインスタンスについて、オペレーターは、そのインスタンスが再作成されるまで、情報 `cannot decrease storage requirement` メッセージをログに記録します。これは予想されることであり、無害です。

3. 古いサイズのボリュームがまだあるスタンバイを 1 つ破棄します。オペレーターは、代替インスタンスをプロビジョニングします。インスタンスシリアルはリサイクルされるため、破棄した名前を再利用し、新しい小さいボリュームにプロビジョニングします。

```
kubectl-cnp destroy CLUSTER INSTANCE
```

4. オペレーターが代替インスタンスを作成し、正常になるまで待ちます。

5. 古いサイズのボリュームがまだある残りのすべてのスタンバイに対して、手順 3 および 4 を繰り返します。

6. 新しく作成したスタンバイのいずれかを昇格させて、古いサイズのボリュームが残っている現在のプライマリがスタンバイに降格されます。

```
kubectl-cnp promote CLUSTER INSTANCE
```

7. オペレーターが新しい小さなボリュームにその代替をプロビジョニングし、すべてのインスタンスが正常になるまで待機するように、元のプライマリ現在は古いサイズのボリュームを持つスタンバイを破壊します。

8. `.spec.instances` を元の値に減らします。

5.20.10 永続ボリュームの静的プロビジョニング

CloudNativePG クラスターの EDB Postgres AI は、動的ボリュームプロビジョニングで動作するように設計されました。この機能により、ストレージクラスと PVC テンプレートを介してユーザーが要求したときに、ストレージボリュームをオンデマンドで作成できます。ストレージの再作成 を参照してください。

ただし、場合によっては、Kubernetes 管理者がストレージボリュームを手動で作成し、Kubernetes クラスター内での表現形式に関連する `PersistentVolume` オブジェクトを作成することを優先する場合があります。これは、ボリュームの_pre-provisioning_としても知られています。

CloudNativePG クラスターの EDB Postgres AI で事前プロビジョニングされたボリュームを使用するには

1. Kubernetes の外部でボリュームを手動で作成します。
2. 実際の CSI ドライバーつまり `volumeHandle` 、`fsType` 、`storageClassName` などに必要な正しいパラメーターを使用して、このボリュームと一致するように `PersistentVolume` オブジェクトを作成します。
3. ストレージセクションごとに、コヒーレントな *pvcTemplate* を使用して Postgres `Cluster` を作成します

このセクションは、Kubernetes が `PersistentVolume` とマッチし、CloudNativePG クラスターの EDB Postgres AI を有効にして必要な `PersistentVolumeClaim` を作成するのに役立ちます。

> **警告**
>
> 静的プロビジョニングでは、事前プロビジョニングされたボリュームが存在する Kubernetes によって Postgres ポッドを正しくスケジュールできることを確認するのはあなたの責任です。スケジューリング構成は、クラスターのアフィニティルールにもとづいています。クラスターを展開した後、*Pending* にスタックしているポッドがないかを確認してください。状態が続く場合は、それが発生した理由を調査します。

5.20.11 ブロックストレージの考慮事項 Ceph/Longhorn

Longhorn や Ceph など、Kubernetes のほとんどのブロックストレージソリューションでは、復元力を強化するためにボリュームの複数のレプリカを持つことをお勧めします。このアプローチは、組み込みの復元力が不足しているワークロードに適しています。

ただし、 CloudNativePG クラスターの EDB Postgres AI は、**状態の同期** で説明しているように、インスタンスの数とそれらにアタッチされている永続ボリュームを介して、この復元力を Postgres `Cluster` に直接統合します。

その結果、ストレージレベルで追加のレプリカを定義すると、書き込み増幅が発生し、ディスク I/O と領域の使用量が不必要に増加する可能性があります。

CloudNativePG クラスター用の EDB Postgres AI を使用する場合、ブロックストレージレベルのレプリカの数を 1 つに減らしながら、`Cluster` リソース全体のストレージレベルで単一障害点 SPoF が存在しないことを確認します。これは通常、単一のストレージホスト、そして最終的には物理ディスクが、より広範な _shared-nothing アーキテクチャ_原則に沿って、同じ `Cluster` の異なるインスタンスからのブロックをホストしないことを意味します。

Longhorn では、カスタムストレージクラスを作成するときに厳密なローカルデータの局所性を有効にすることにより、このリスクを軽減できます。厳密なローカルデータの局所性を使用してボリュームを作成するための詳細な手順は、here を参照してください。この設定により、ポッドのデータボリュームがポッド自分自身と同じノードに存在します。

さらに、Postgres `Cluster` には *pod anti-affinity rules* が必要です

オペレーターがさまざまなノードにポッドを展開できるように、Longhorn は対応するホストにデータボリュームを配置できます。必要に応じて、ボリュームレプリカ数を一時的に 2 に設定し、後で減らして、古

いレプリカを削除することにより、Longhorn のボリュームを手動で再配置できます。ホストが破損した場合は、*CNPG-I*

影響を受けるインスタンス。 CloudNativePG クラスターの EDB Postgres AI は、別のホストでインスタンスを再作成し、データをレプリケートします。

Ceph では、これは CRUSH ルールを介して構成できます。CRUSH ルールの構成に関するドキュメントは here で入手できます。これらのルールは、ノードごとにポッドごとに 1 つのボリュームを保証することを目的としています。ボリュームを別のプールにインポートすることにより、再配置することもできます。

5.21 Labels and Annotations

Kubernetes のリソースはフラット構造で編成され、階層情報やそれらの間の関係はありません。ただし、このようなリソースとオブジェクトは、_labels_および_annotations_を介してリンクし、関係を置くことができます。

> **注釈**
>
> 詳細については、 annotations および labels に関する Kubernetes ドキュメントを参照してください。

簡単に言うと

- アノテーションは、外部ツールとの統合を促進することを目的に、追加の非識別情報をリソースに割り当てるために使用されます。
- ラベルは、オブジェクトをグループ化し、Kubernetes ネイティブセレクター機能を介して照会するために使用されます。

CloudNativePG クラスター展開の EDB Postgres AI で使用する 1 つ以上のラベルまたはアノテーションを選択できます。次に、クラスターのメタデータでこれらのラベルまたはアノテーションを定義すると、それによって作成されたすべてのリソースポッドを含むすべてに継承されるようにオペレーターを構成する必要があります。

> **注釈**
>
> ラベルとアノテーションの継承は、ポッドテンプレートなどの代替アプローチの代わりに CloudNativePG クラスターの EDB Postgres AI によって採用された技術です。

5.21.1 事前定義されたラベル

CloudNativePG クラスターの EDB Postgres AI は、次の事前定義されたラベルを管理します。

`k8s.enterprisedb.io/backupDate`

ISO 8601 形式 `YYYYMMDD` のバックアップの日付。このラベルは、`VolumeSnapshot` リソースでのみ使用できます。

`k8s.enterprisedb.io/backupName`

バックアップ識別子。このラベルは、VolumeSnapshot リソースでのみ使用できます。

k8s.enterprisedb.io/backupMonth

バックアップが取られた年/月。このラベルは、VolumeSnapshot リソースでのみ使用できます。

k8s.enterprisedb.io/backupTimeline

バックアップが作成されたときのインスタンスのタイムライン。このラベルは、VolumeSnapshot リソースでのみ使用できます。

k8s.enterprisedb.io/backupYear

バックアップが取られた年。このラベルは、VolumeSnapshot リソースでのみ使用できます。

k8s.enterprisedb.io/cluster

クラスターの名前。

k8s.enterprisedb.io/immediateBackup

バックアップが immediate が true に設定されている ScheduledBackup オブジェクトから作成された最初のバックアップである場合、 Backup リソースに適用されます。

k8s.enterprisedb.io/instanceName

PostgreSQL インスタンスの名前古い非推奨の postgresql ラベルを置き換えます。

k8s.enterprisedb.io/jobRole

ジョブのロールつまり import 、initdb 、join , …

k8s.enterprisedb.io/majorVersion

バックアップのデータディレクトリの整数 PostgreSQL メジャーバージョンたとえば 17 。このラベルは、VolumeSnapshot リソースでのみ使用できます。

k8s.enterprisedb.io/onlineBackup

バックアップがオンラインであるかホットか、Postgres がダウンしているときに取得されたコールドかどうか。このラベルは、VolumeSnapshot リソースでのみ使用できます。

postgresql

非推奨、PostgreSQL インスタンスの名前。代わりに k8s.enterprisedb.io/instanceName を使用してください

k8s.enterprisedb.io/podRole

プーラー展開専用のポッドとデータベースインスタンスに使用されるポッドを区別します。

k8s.enterprisedb.io/poolerName

PgBouncer プーラーの名前。

k8s.enterprisedb.io/pvcRole

PG_DATA や PG_WAL などの PVC の目的。

k8s.enterprisedb.io/reload

ConfigMap および Secret リソースで使用できます。 true に設定されている場合、リソースの変更はオペレーターによって自動的にリロードされます。

k8s.enterprisedb.io/userType

Secret 、superuser Postgres スーパーユーザーアクセスまたは app CloudNativePG クラスター用語のアプリケーションレベルのユーザーに関連付けられている PostgreSQL ユーザーのタイプを指定し、CloudNativePG クラスターの EDB Postgres EDB Postgres によって作成されたデフォルトユーザーに制限されます通常は postgres および app)。

role - **非推奨**

ポッドで実行されているインスタンスのロール primary 、replica 、または unhealthy 。 unhealthy 値は一時的なものです。オペレーターは、フェイルオーバーまたはスイッチオーバー中に古いプライマリでこれを設定し、移行が完了すると自動的にクリアされます。このラベルは非推奨です。代わりに k8s.enterprisedb.io/instanceRole を使用する必要があります。

k8s.enterprisedb.io/scheduled-backup

使用可能な場合、特定の Backup オブジェクトを作成した ScheduledBackup リソースの名前。

k8s.enterprisedb.io/instanceRole

ポッドで実行されているインスタンスのロール primary 、replica 、または unhealthy 。 unhealthy 値は一時的なものです。オペレーターは、フェイルオーバーまたはスイッチオーバー中に古いプライマリでこれを設定し、移行が完了すると自動的にクリアされます。

app.kubernetes.io/managed-by

マネージャーの名前。常に cloudnative-pg になります。CloudNativePG クラスター管理対象リソースのすべての EDB Postgres AI で利用できます。

app.kubernetes.io/name

アプリケーションの名前。常に postgresql になります。ポッド、ジョブ、展開、サービス、持続的ボリュームクレーム、ボリュームスナップショット、podDisruptionBudgets、podMonitors で使用できます。

app.kubernetes.io/component

コンポーネントの名前 database 、pooler 、…。ポッド、ジョブ、展開、サービス、持続的ボリュームクレーム、ボリュームスナップショット、podDisruptionBudgets、podMonitors で使用できます。

app.kubernetes.io/instance

所有する Cluster リソースの名前。ポッド、ジョブ、展開、サービス、volumeSnapshot、podDisruptionBudget、podMonitor で使用できます。

app.kubernetes.io/version

PostgreSQL のメジャーバージョン。ポッド、ジョブ、サービス、volumeSnapshot、podDisruptionBudget、podMonitor で使用できます。

5.21.2 事前定義されたアノテーション

CloudNativePG クラスターの EDB Postgres AI は、次の事前定義されたアノテーションを管理します。

container.apparmor.security.beta.kubernetes.io/*

名前付けたコンテナに適用する AppArmor プロファイルの名前。*AppArmor* を使用してポッドアクセスを制限する を参照

詳細については。

k8s.enterprisedb.io/backupEndTime

バックアップが終了した時刻。このアノテーションは、VolumeSnapshot リソースでのみ使用できます。

k8s.enterprisedb.io/backupEndWAL

バックアップ終了後の WAL。このアノテーションは、VolumeSnapshot リソースでのみ使用できます。

k8s.enterprisedb.io/backupStartTime

バックアップが開始された時刻。

k8s.enterprisedb.io/backupStartWAL

バックアップの開始時の WAL。このアノテーションは、VolumeSnapshot リソースでのみ使用できます。

k8s.enterprisedb.io/coredumpFilter

ビットマスクで表現される Postgres プロセスのコアダンプを制御するフィルター。デフォルトでは、ダンプから共有メモリセグメントを除外するように 0x31 に設定されます。 *PostgreSQL* コアダンプ を参照

詳細については、

k8s.enterprisedb.io/clusterManifest

このリソース PVC などを所有する Cluster のマニフェスト。このラベルは、古い非推奨の k8s.enterprisedb.io/hibernateClusterManifest ラベルを置き換えます。

k8s.enterprisedb.io/fencedInstances

JSON 形式で表現される、フェンスする必要があるインスタンスのリスト。リストに* 要素が含まれる場合、クラスター全体がフェンスされます。

k8s.enterprisedb.io/forceLegacyBackup

テスト目的のみで Cluster リソースに適用され、 --name オプションが使用できなかったバージョン 3.4 2023 年 1 月より前の barman-cloud-backup の動作をシミュレートします。

k8s.enterprisedb.io/hash

リソースのハッシュ値。

k8s.enterprisedb.io/hibernation

Cluster リソースに適用されて、*declarative hibernation feature* を制御します。使用できる値は on および off です。

k8s.enterprisedb.io/managedSecrets

オペレーターが管理するシークレットをプルし、各 Postgres クラスターの ServiceAccount リソースに自動的に設定します。

k8s.enterprisedb.io/nodeSerial

ポッドリソースで、Postgres クラスター内のインスタンスのシリアル番号を識別します。

k8s.enterprisedb.io/operatorVersion

オペレーターのバージョン。

k8s.enterprisedb.io/passwordPassthrough

オペレータースーパーユーザー、アプリケーションユーザー、または管理対象ロールパスワードシークレットが使用するベーシック認証 Secret で enabled に設定されている場合、オペレーターは SCRAM-SHA-256 エンコードの代わりに CREATE/ALTER ROLE ステートメントでパスワード値を逐語転送しますオペレーター側。PostgreSQL は、独自の password_encryption 設定に従って値をエンコードします。 **オペレーターサイドのエンコーディングのオプトアウト** を参照してください。

k8s.enterprisedb.io/pgControldata

pg_controldata コマンドの出力。このアノテーションは、古い非推奨の k8s.enterprisedb.io/hibernatePgControlData アノテーションを置き換えます。

k8s.enterprisedb.io/podEnvHash

k8s.enterprisedb.io/podSpec アノテーションにもポッド環境が含まれるようになったため、非推奨。

k8s.enterprisedb.io/podPatch

アノテーションは Cluster リソースに適用できます。

```
When set to a JSON-patch formatted patch, the patch will be applied to the
instance Pods. The patch can modify any field of the Pod specification,
including security-sensitive fields. The operator validates only that the
patch is syntactically correct and applicable.

As with all Cluster fields that affect Pod specifications, enforcement of
security constraints is delegated to the Kubernetes admission control chain.
See the
[Trust Model and Security Boundaries](security.md#trust-model-and-security-boundaries)
section for details.

 **☒ WARNING:**  This feature may introduce discrepancies between the
operators expectations and Kubernetes behavior. Use with caution and only as a
last resort.

 **IMPORTANT** : adding or changing this annotation wont trigger a rolling deployment
of the generated Pods. The latter can be triggered manually by the user with
`kubectl cnp restart`.
```

k8s.enterprisedb.io/podSpec

オペレーターによって生成されたポッドの spec のスナップショット。このアノテーションは、古い非推奨の k8s.enterprisedb.io/podEnvHash アノテーションを置き換えます。

k8s.enterprisedb.io/poolerSpecHash

プーラーリソースのハッシュ。

k8s.enterprisedb.io/pvcStatus

PVC の現在のステータス initializing 、ready 、または detached 。

k8s.enterprisedb.io/reconcilePodSpec

アノテーションを Cluster または Pooler に適用して、再起動を防止できます。

```
When set to `disabled` on a `Cluster`, the operator prevents instances
from restarting due to changes in the PodSpec. This includes changes to:

  - Topology or affinity
  - Scheduler
  - Volumes or containers

When set to `disabled` on a `Pooler`, the operator restricts any modifications
to the deployment specification, except for changes to `spec.instances`.
```

k8s.enterprisedb.io/reconciliationLoop

Cluster で disabled に設定されている場合、オペレーターは調整ループの実行を防止します。

k8s.enterprisedb.io/reloadedAt

最新のクラスター reload 時間が含まれています。 reload は、プラグインを介してユーザーによってトリガーされます。

k8s.enterprisedb.io/skipEmptyWalArchiveCheck

Cluster リソースで enabled に設定されている場合、オペレーターは、データを書き込む前に WAL アーカイブが空であることを確認するチェックを無効にします。自分の責任で使用してください。

k8s.enterprisedb.io/skipWalArchiving

Cluster リソースで enabled に設定されている場合、オペレーターは WAL アーカイブを無効にします。これにより、archive_mode が off に設定され、すべての PostgreSQL インスタンスの再起動が必要です。自分の責任で使用してください。

k8s.enterprisedb.io/snapshotStartTime

スナップショットが開始された時刻。

k8s.enterprisedb.io/snapshotEndTime

スナップショットが使用する準備ができているとマークされた時間。

k8s.enterprisedb.io/validation

CloudNativePG クラスター管理カスタムリソースの EDB Postgres AI で disabled に設定されている場合、検証 Webhook は制限なしですべての変更を許可します。

```
 **⚠ WARNING:**  Disabling validation may permit unsafe or destructive
operations. Use this setting with caution and at your own risk.
```

k8s.enterprisedb.io/volumeSnapshotDeadline

Backup および ScheduledBackup リソースに適用され、ボリュームのスナップショットのバックアップが失敗したと見なす前に、オペレーターが回復可能なエラーを再試行する時間を制御できます。分単位、デフォルト 10

kubectl.kubernetes.io/restartedAt

利用可能な場合、最後に Postgres クラスターの再起動が要求された時間。

alpha.k8s.enterprisedb.io/unrecoverable

PostgreSQL インスタンスを実行している Pod に適用される実験的なアノテーション。 Pod およびそれに関連するすべての PVC を削除するようにオペレーターに指示します。インスタンスは、構成された参加戦略に従って再作成されます。このアノテーションは、現在のプライマリでも指定されたターゲットプライマリでもないインスタンスでのみ使用できます。

5.21.3 前提条件

デフォルトでは、クラスターのメタデータで定義されたラベルまたはアノテーションは、関連付けられたリソースに継承されません。ラベル/アノテーションの継承を有効にするには、*Operator configuration* で提供されている手順に従ってください。

以下はその例の続きであり、次のことに制限します。

- アノテーション categories
- ラベル app 、environment 、および workload

注釈

注釈とラベルの両方に、コンテキストに最も適した名前を自由に選択してください。名前付けにワイルドカードを使用し、すべてのラベルに`mycompany/*` を使用したり、*mycompany/* で始まるアノテーションを継承するように設定したりするなどの戦略を採用することもできます。

5.21.4 クラスターのメタデータの定義

クラスターを定義するときに、リソースがデプロイされる前に、次のようにメタデータを設定できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
  annotations:
```

(次のページに続く)

(前のページからの続き)

```
    categories: database
  labels:
    environment: production
    workload: database
    app: sso
spec:
  # ... <snip>
```

クラスターが展開されると、たとえば、ポッドにラベルが正しく設定されたことを確認できます。

```
kubectl get pods --show-labels
```

5.21.5 現在の制限

現在、CloudNativePG クラスターの EDB Postgres AI は、ラベルまたは注釈の削除を自動的に伝播しません。したがって、以前に基になるポッドに伝播されたクラスターからアノテーションまたはラベルが削除される場合、オペレーターは、関連するリソースでそれを削除しません。

5.22 Monitoring

5.22.1 モニタリングインスタンス

PostgreSQL インスタンスごとに、オペレーターは、HTTP または HTTPS、ポート 9187 で `metrics` という名前の Prometheus のメトリックのエクスポーターを提供します。オペレーターには、**事前定義されたメトリックセット** に加えて、1 つ以上の `ConfigMap` または `Secret` リソースを介して追加クエリを定義するための高度な構成およびカスタマイズ可能なシステムが付属しています詳細については、以下の **ユーザー定義メトリック** を参照してください。

`postgresql-operator-default-monitoring` という名前の `ConfigMap` で。

> **注釈**
>
> **エクスポートされたメトリックを検査する方法** の指示に従って、エクスポートされたメトリックを検査できます。

以下のセクション。

PostgreSQL で実行されるすべての監視クエリは次のとおりです。

- アトミック クエリごとに 1 つの読み取り専用トランザクション
- `cnp_metrics_exporter` ロール `pg_monitor` のメンバーとして実行
- `application_name` を `cnp_metrics_exporter` に設定して実行

接続は、ポッドローカルの UNIX ソケットでのピア認証を使用します。`session_user` は決してスーパーユーザーではないため、モニタリングセッションは `RESET ROLE` または `RESET SESSION AUTHORIZATION`

を介してエスカレートできません。pg_monitor およびカスタムクエリーに必要なテーブルレベルの付与を超えて、cnp_metrics_exporter に追加の特権またはロールメンバーシップを付与しないでください。追加のメンバーシップは、継承を介してスクレイプセッションにフローし、このプロパティを弱めます。

PostgreSQL の「事前定義されたロール」セクションを参照してください documentation

pg_monitor ロールの詳細は、

デフォルトでは、クエリーは、次のロジックに従って、Cluster リソースの指定された bootstrap メソッドで定義された_main Database_に対して実行されます。

- using initdb クエリーはデフォルトで initdb.database または app で指定されたデータベースに対して実行されます指定しない場合
- using recovery クエリーはデフォルトで recovery.database または postgres で指定されたデータベースに対して実行されます指定しない場合
- using pg_basebackup クエリーはデフォルトで pg_basebackup.database または postgres で指定されたデータベースに対して実行されます指定しない場合

target_databases オプションで 1 つ以上のデータベースのリストを指定することにより、特定のユーザー定義メトリックのデフォルトデータベースを常にオーバーライドできます。

出力キャッシュ

デフォルトでは、モニタリングクエリの出力は 30 秒間キャッシュされます。これは、リソース効率を向上させ、プロメテウスエンドポイントがスクレイピングされるたびに PostgreSQL が監視クエリを実行しないようにするために行われます。

キャッシュ自分自身は、 cache_hits 、cache_misses 、および last_update_timestamp メトリックで観察できます。

cluster.spec.monitoring.metricsQueriesTTL をゼロに設定するとキャッシュが無効になり、その場合、メトリックはすべてのメトリックエンドポイントスクレイピングで実行されます。

Prometheus オペレーターを使用したモニタリング

Prometheus Operator's を使用して、特定の PostgreSQL クラスターをモニタできます。

PodMonitor 。

推奨されるアプローチは、CloudNativePG Cluster クラスターの EDB Postgres AI ごとに PodMonitor を手動で作成および管理することです。この方法では、監視構成とライフサイクルを完全に制御できます。

PodMonitor の作成

クラスターをモニタするには、次のように PodMonitor リソースを定義します。 PodMonitor リソースを検索するように Prometheus Operator が構成されている名前空間に必ずデプロイしてください。

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
```

(次のページに続く)

(前のページからの続き)

```
  name: cluster-example
spec:
  selector:
    matchLabels:
      k8s.enterprisedb.io/cluster: cluster-example
  podMetricsEndpoints:
    - port: metrics
```

- metadata.name PodMonitor に一意の名前を付けます。
- spec.namespaceSelector これを使用して、PostgreSQL クラスターが実行されている名前空間を指定します。
- ***spec.selector.matchLabels k8s.enterprisedb.io/cluster: <cluster-name>* ラベルを使用して、PostgreSQL インスタンスを正しくターゲットとする必要があります。 .. ::**
  #### *PodMonitor* 自動作成の非推奨

現在この機能を使用している場合、 .spec.monitoring.enablePodMonitor を削除または false に設定し、上記のようにクラスターの PodMonitor リソースを手動で作成することを強くお勧めします。この変更により、監視構成の完全な所有権が保証され、オペレーターによって管理または上書きされるのを防ぎます。

メトリックポートで TLS を有効にする

メトリックポートで TLS 通信を有効にするには、.spec.monitoring.tls.enabled 設定を true に構成します。この設定により、メトリックエクスポーターは、PostgreSQL が使用するものと同じサーバー証明書を使用して、ポート 5432 での通信を安全にします。

PodMonitor がオペレーター.spec.monitoring.enablePodMonitor を true に設定して管理されている場合、TLS を介してメトリックにアクセスするために必要な構成が自動的に含まれます。

TLS 経由でメトリックを読み取るのに適した PodMonitor を手動で展開するには、次のように定義し、必要に応じて調整します。

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: cluster-example
spec:
  selector:
    matchLabels:
      "k8s.enterprisedb.io/cluster": cluster-example
  podMetricsEndpoints:
    - port: metrics
      scheme: https
      tlsConfig:
        ca:
          secret:
```

(次のページに続く)

(前のページからの続き)

```
            name: cluster-example-ca
            key: ca.crt
        serverName: cluster-example-rw
```

事前定義されたメトリックセット

すべての PostgreSQL インスタンスエクスポーターは、事前定義されたメトリックのセットを自動的に公開します。これらは 2 つの主要カテゴリに分類できます。

- cnp_collector_* で始まる PostgreSQL 関連メトリックインスタンスを収容する個別のノードの数 s
- 最後に失敗したバックアップと最後に使用可能なバックアップを示すタイムスタンプ、および cluster
- フラグ レプリカクラスターモードが有効または無効かを示します
- フラグ手動スイッチオーバーが必要かどうかを示す b_tran_12 フェンシングが有効かどうかを示す b_tran_12 フラグ有効または無効
- go_* で始まる Go ランタイム関連のメトリック

以下は、インスタンスの localhost:9187/metrics エンドポイントによって返されるメトリックのサンプルです。ご覧のとおり、Prometheus フォーマットは自己文書です。

```
#  HELP cnp_collector_collection_duration_seconds Collection time duration in seconds

#  TYPE cnp_collector_collection_duration_seconds gauge

cnp_collector_collection_duration_seconds{collector="Collect.up"} 0.0031393

#  HELP cnp_collector_collections_total Total number of times PostgreSQL was accessed␣
↪for metrics.

#  TYPE cnp_collector_collections_total counter

cnp_collector_collections_total 2

#  HELP cnp_collector_fencing_on 1 if the instance is fenced, 0 otherwise

#  TYPE cnp_collector_fencing_on gauge

cnp_collector_fencing_on 0

#  HELP cnp_collector_nodes_used NodesUsed represents the count of distinct nodes␣
↪accommodating the instances. A value of -1 suggests that the metric is not␣
↪available. A value of 1 suggests that all instances are hosted on a single node,␣
↪implying the absence of High Availability (HA). Ideally this value should match the␣
↪number of instances in the cluster.
```

(次のページに続く)

(前のページからの続き)

```
#  TYPE cnp_collector_nodes_used gauge

cnp_collector_nodes_used 3

#  HELP cnp_collector_last_collection_error 1 if the last collection ended with error,
 0 otherwise.

#  TYPE cnp_collector_last_collection_error gauge

cnp_collector_last_collection_error 0

#  HELP cnp_collector_manual_switchover_required 1 if a manual switchover is required,
 0 otherwise

#  TYPE cnp_collector_manual_switchover_required gauge

cnp_collector_manual_switchover_required 0

#  HELP cnp_collector_pg_wal Total size in bytes of WAL segments in the /var/lib/
↪postgresql/data/pgdata/pg_wal directory  computed as (wal_segment_size * count)

#  TYPE cnp_collector_pg_wal gauge

cnp_collector_pg_wal{value="count"} 9
cnp_collector_pg_wal{value="slots_max"} NaN
cnp_collector_pg_wal{value="keep"} 32
cnp_collector_pg_wal{value="max"} 64
cnp_collector_pg_wal{value="min"} 5
cnp_collector_pg_wal{value="size"} 1.50994944e+08
cnp_collector_pg_wal{value="volume_max"} 128
cnp_collector_pg_wal{value="volume_size"} 2.147483648e+09

#  HELP cnp_collector_pg_wal_archive_status Number of WAL segments in the /var/lib/
↪postgresql/data/pgdata/pg_wal/archive_status directory (ready, done)

#  TYPE cnp_collector_pg_wal_archive_status gauge

cnp_collector_pg_wal_archive_status{value="done"} 6
cnp_collector_pg_wal_archive_status{value="ready"} 0

#  HELP cnp_collector_replica_mode 1 if the cluster is in replica mode, 0 otherwise

#  TYPE cnp_collector_replica_mode gauge

cnp_collector_replica_mode 0
```

(次のページに続く)

(前のページからの続き)

```
# HELP cnp_collector_sync_replicas Number of requested synchronous replicas␣
↪(synchronous_standby_names)

# TYPE cnp_collector_sync_replicas gauge

cnp_collector_sync_replicas{value="expected"} 0
cnp_collector_sync_replicas{value="max"} 0
cnp_collector_sync_replicas{value="min"} 0
cnp_collector_sync_replicas{value="observed"} 0

# HELP cnp_collector_up 1 if PostgreSQL is up, 0 otherwise.

# TYPE cnp_collector_up gauge

cnp_collector_up{cluster="cluster-example"} 1

# HELP cnp_collector_postgres_version Postgres version

# TYPE cnp_collector_postgres_version gauge

cnp_collector_postgres_version{cluster="cluster-example",full="18.4"} 18.4

# HELP cnp_collector_last_failed_backup_timestamp The last failed backup as a unix␣
↪timestamp (Deprecated)

# TYPE cnp_collector_last_failed_backup_timestamp gauge

cnp_collector_last_failed_backup_timestamp 0

# HELP cnp_collector_last_available_backup_timestamp The last available backup as a␣
↪unix timestamp (Deprecated)

# TYPE cnp_collector_last_available_backup_timestamp gauge

cnp_collector_last_available_backup_timestamp 1.63238406e+09

# HELP cnp_collector_first_recoverability_point The first point of recoverability␣
↪for the cluster as a unix timestamp (Deprecated)

# TYPE cnp_collector_first_recoverability_point gauge

cnp_collector_first_recoverability_point 1.63238406e+09

# HELP cnp_collector_lo_pages Estimated number of pages in the pg_largeobject table
```

(次のページに続く)

(前のページからの続き)

```
#  TYPE cnp_collector_lo_pages gauge

cnp_collector_lo_pages{datname="app"} 0
cnp_collector_lo_pages{datname="postgres"} 78

#  HELP cnp_collector_wal_buffers_full Number of times WAL data was written to disk␣
↪because WAL buffers became full. Only available on PG 14+

#  TYPE cnp_collector_wal_buffers_full gauge

cnp_collector_wal_buffers_full{stats_reset="2023-06-19T10:51:27.473259Z"} 6472

#  HELP cnp_collector_wal_bytes Total amount of WAL generated in bytes. Only␣
↪available on PG 14+

#  TYPE cnp_collector_wal_bytes gauge

cnp_collector_wal_bytes{stats_reset="2023-06-19T10:51:27.473259Z"} 1.0035147e+07

#  HELP cnp_collector_wal_fpi Total number of WAL full page images generated. Only␣
↪available on PG 14+

#  TYPE cnp_collector_wal_fpi gauge

cnp_collector_wal_fpi{stats_reset="2023-06-19T10:51:27.473259Z"} 1474

#  HELP cnp_collector_wal_records Total number of WAL records generated. Only␣
↪available on PG 14+

#  TYPE cnp_collector_wal_records gauge

cnp_collector_wal_records{stats_reset="2023-06-19T10:51:27.473259Z"} 26178

#  HELP cnp_collector_wal_sync Number of times WAL files were synced to disk via␣
↪issue_xlog_fsync request (if fsync is on and wal_sync_method is either fdatasync,␣
↪fsync or fsync_writethrough, otherwise zero). Only available on PG 14+

#  TYPE cnp_collector_wal_sync gauge

cnp_collector_wal_sync{stats_reset="2023-06-19T10:51:27.473259Z"} 37

#  HELP cnp_collector_wal_sync_time Total amount of time spent syncing WAL files to␣
↪disk via issue_xlog_fsync request, in milliseconds (if track_wal_io_timing is␣
↪enabled, fsync is on, and wal_sync_method is either fdatasync, fsync or fsync_
```

(次のページに続く)

(前のページからの続き)

```
↪writethrough, otherwise zero). Only available on PG 14+

#  TYPE cnp_collector_wal_sync_time gauge

cnp_collector_wal_sync_time{stats_reset="2023-06-19T10:51:27.473259Z"} 0

#  HELP cnp_collector_wal_write Number of times WAL buffers were written out to disk␣
↪via XLogWrite request. Only available on PG 14+

#  TYPE cnp_collector_wal_write gauge

cnp_collector_wal_write{stats_reset="2023-06-19T10:51:27.473259Z"} 7243

#  HELP cnp_collector_wal_write_time Total amount of time spent writing WAL buffers␣
↪to disk via XLogWrite request, in milliseconds (if track_wal_io_timing is enabled,␣
↪otherwise zero). This includes the sync time when wal_sync_method is either open_
↪datasync or open_sync. Only available on PG 14+

#  TYPE cnp_collector_wal_write_time gauge

cnp_collector_wal_write_time{stats_reset="2023-06-19T10:51:27.473259Z"} 0

#  HELP cnp_last_error 1 if the last collection ended with error, 0 otherwise.

#  TYPE cnp_last_error gauge

cnp_last_error 0

#  HELP go_gc_duration_seconds A summary of the pause duration of garbage collection␣
↪cycles.

#  TYPE go_gc_duration_seconds summary

go_gc_duration_seconds{quantile="0"} 5.01e-05
go_gc_duration_seconds{quantile="0.25"} 7.27e-05
go_gc_duration_seconds{quantile="0.5"} 0.0001748
go_gc_duration_seconds{quantile="0.75"} 0.0002959
go_gc_duration_seconds{quantile="1"} 0.0012776
go_gc_duration_seconds_sum 0.0035741
go_gc_duration_seconds_count 13

#  HELP go_goroutines Number of goroutines that currently exist.

#  TYPE go_goroutines gauge
```

(次のページに続く)

(前のページからの続き)

```
go_goroutines 25

#  HELP go_info Information about the Go environment.

#  TYPE go_info gauge

go_info{version="go1.20.5"} 1

#  HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.

#  TYPE go_memstats_alloc_bytes gauge

go_memstats_alloc_bytes 4.493744e+06

#  HELP go_memstats_alloc_bytes_total Total number of bytes allocated, even if freed.

#  TYPE go_memstats_alloc_bytes_total counter

go_memstats_alloc_bytes_total 2.1698216e+07

#  HELP go_memstats_buck_hash_sys_bytes Number of bytes used by the profiling bucket␣
↪hash table.

#  TYPE go_memstats_buck_hash_sys_bytes gauge

go_memstats_buck_hash_sys_bytes 1.456234e+06

#  HELP go_memstats_frees_total Total number of frees.

#  TYPE go_memstats_frees_total counter

go_memstats_frees_total 172118

#  HELP go_memstats_gc_cpu_fraction The fraction of this programs available CPU time␣
↪used by the GC since the program started.

#  TYPE go_memstats_gc_cpu_fraction gauge

go_memstats_gc_cpu_fraction 1.0749468700447189e-05

#  HELP go_memstats_gc_sys_bytes Number of bytes used for garbage collection system␣
↪metadata.

#  TYPE go_memstats_gc_sys_bytes gauge
```

(次のページに続く)

(前のページからの続き)

```
go_memstats_gc_sys_bytes 5.530048e+06

#  HELP go_memstats_heap_alloc_bytes Number of heap bytes allocated and still in use.

#  TYPE go_memstats_heap_alloc_bytes gauge

go_memstats_heap_alloc_bytes 4.493744e+06

#  HELP go_memstats_heap_idle_bytes Number of heap bytes waiting to be used.

#  TYPE go_memstats_heap_idle_bytes gauge

go_memstats_heap_idle_bytes 5.8236928e+07

#  HELP go_memstats_heap_inuse_bytes Number of heap bytes that are in use.

#  TYPE go_memstats_heap_inuse_bytes gauge

go_memstats_heap_inuse_bytes 7.528448e+06

#  HELP go_memstats_heap_objects Number of allocated objects.

#  TYPE go_memstats_heap_objects gauge

go_memstats_heap_objects 26306

#  HELP go_memstats_heap_released_bytes Number of heap bytes released to OS.

#  TYPE go_memstats_heap_released_bytes gauge

go_memstats_heap_released_bytes 5.7401344e+07

#  HELP go_memstats_heap_sys_bytes Number of heap bytes obtained from system.

#  TYPE go_memstats_heap_sys_bytes gauge

go_memstats_heap_sys_bytes 6.5765376e+07

#  HELP go_memstats_last_gc_time_seconds Number of seconds since 1970 of last garbage␣
↪collection.

#  TYPE go_memstats_last_gc_time_seconds gauge

go_memstats_last_gc_time_seconds 1.6311727586032727e+09
```

(次のページに続く)

(前のページからの続き)

```
#  HELP go_memstats_lookups_total Total number of pointer lookups.

#  TYPE go_memstats_lookups_total counter

go_memstats_lookups_total 0

#  HELP go_memstats_mallocs_total Total number of mallocs.

#  TYPE go_memstats_mallocs_total counter

go_memstats_mallocs_total 198424

#  HELP go_memstats_mcache_inuse_bytes Number of bytes in use by mcache structures.

#  TYPE go_memstats_mcache_inuse_bytes gauge

go_memstats_mcache_inuse_bytes 14400

#  HELP go_memstats_mcache_sys_bytes Number of bytes used for mcache structures␣
↪obtained from system.

#  TYPE go_memstats_mcache_sys_bytes gauge

go_memstats_mcache_sys_bytes 16384

#  HELP go_memstats_mspan_inuse_bytes Number of bytes in use by mspan structures.

#  TYPE go_memstats_mspan_inuse_bytes gauge

go_memstats_mspan_inuse_bytes 191896

#  HELP go_memstats_mspan_sys_bytes Number of bytes used for mspan structures␣
↪obtained from system.

#  TYPE go_memstats_mspan_sys_bytes gauge

go_memstats_mspan_sys_bytes 212992

#  HELP go_memstats_next_gc_bytes Number of heap bytes when next garbage collection␣
↪will take place.

#  TYPE go_memstats_next_gc_bytes gauge

go_memstats_next_gc_bytes 8.689632e+06
```

(次のページに続く)

(前のページからの続き)

```
#  HELP go_memstats_other_sys_bytes Number of bytes used for other system allocations.

#  TYPE go_memstats_other_sys_bytes gauge

go_memstats_other_sys_bytes 2.566622e+06

#  HELP go_memstats_stack_inuse_bytes Number of bytes in use by the stack allocator.

#  TYPE go_memstats_stack_inuse_bytes gauge

go_memstats_stack_inuse_bytes 1.343488e+06

#  HELP go_memstats_stack_sys_bytes Number of bytes obtained from system for stack␣
↪allocator.

#  TYPE go_memstats_stack_sys_bytes gauge

go_memstats_stack_sys_bytes 1.343488e+06

#  HELP go_memstats_sys_bytes Number of bytes obtained from system.

#  TYPE go_memstats_sys_bytes gauge

go_memstats_sys_bytes 7.6891144e+07

#  HELP go_threads Number of OS threads created.

#  TYPE go_threads gauge

go_threads 18
```

注釈

cnp_collector_postgres_version は、 Postgres PostgreSQL または EPAS の `Major.Minor` バージョンを含む GaugeVec メトリックです。フルセマンティックバージョン `Major.Minor.Patch` は、*full* という名前のラベルフィールドの 1 つ内にあります。

警告

メトリック `cnp_collector_last_failed_backup_timestamp` 、*cnp_collector_last_available_backup_timestamp* 、および `cnp_collector_first_recoverability_point` は、バージョン 1.26 から非推奨になりました。これらのメトリックは、インコア Barman Cloud 非推奨やボリュームスナップショットなどのネイティブバックアップソリューションで引き続き機能します。これらの場合、オブジェクト

ストアへの最初のバックアップが完了するまで、 *cnp_collector_first_recoverability_point* および `cnp_collector_last_available_backup_timestamp` はゼロのままであることに注意してください。これは WAL アーカイブとは別です。

ユーザー定義メトリック

この機能は現在_beta_状態にあり、フォーマットは queries.yaml file (release 0.12)

PostgreSQL Prometheus Exporter の。

カスタムメトリックは、次の例のように、`.spec.monitoring.customQueriesConfigMap` または `customQueriesSecret` セクションの下の `Cluster` 定義で作成した `Configmap` /`Secret` を参照することにより、ユーザーが定義できます。

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
  namespace: test
spec:
  instances: 3

  storage:
    size: 1Gi

  monitoring:
    customQueriesConfigMap:
      - name: example-monitoring
        key: custom-queries
```

`customQueriesConfigMap` /`customQueriesSecret` セクションには、カスタムクエリーが定義されているキーを指定する `ConfigMap` /`Secret` 参照のリストが含まれています。参照されるリソースは、**クラスター**リソースと同じ名前空間に作成する必要があることに注意してください。

注釈

ConfigMap と Secrets をインスタンスによって**自動的に**リロードしたい場合は、キー `k8s.enterprisedb.io/reload` を持つラベルをそれに追加できます。それ以外の場合、 *kubectl cnp reload* サブコマンドを使用してインスタンスをリロードする必要があります。

カスタムクエリーの特権と安全性

警告

カスタムクエリーは、*pg_monitor* を継承する `cnp_metrics_exporter` ロールとして実行されます。*pg_monitor* のスコープ内のクエリーカタログ読み取り、*pg_stat_** ビュー、構成パラメーターは、変更せずに動作します。ユーザー所有テーブルまたはスーパーユーザー専用カタログ`pg_authid`、*pg_subscription* などを読み取るクエリーには、明示的な付与が必要です。テーブルの読み取りには、そのスキーマでの USAGE も必要です。

```
GRANT USAGE ON SCHEMA myschema TO cnp_metrics_exporter;
GRANT SELECT ON TABLE myschema.mytable TO cnp_metrics_exporter;
```

target_databases のすべてのデータベースは、cnp_metrics_exporter ～CONNECT を許可する必要があります。データベースの PUBLIC から CONNECT を取り消したクラスターでは、そのロールに明示的に付与します。

```
GRANT CONNECT ON DATABASE domainapp TO cnp_metrics_exporter;
```

" *" ワイルドカードよりも、信頼できるデータベースたとえば target_databases: ["domainapp"] の明示的なリストを優先します。ワイルドカードは、ロールが接続できるすべてのデータベースをスクレイピングし、残りをサイレントにスキップするため、明示的なリストにより、不足している付与に気づきやすくなります。"* " は、クエリーがクラスター全体でデータベースごとのメトリックを収集することを目的としている場合にのみ使用します。

カタログ参照 pg_catalog.now()、pg_catalog.current_database() をスキーマ修飾して、ユーザー所有オブジェクトによる search_path シャドウイングを防ぎます。

カスタムモニタリングクエリーは、データベースまたはロールに search_path が構成されているかどうかに関係なく、search_path が pg_catalog, public, pg_temp に固定されているトランザクション内で実行されます。したがって、他のユーザー定義スキーマのオブジェクトへの非修飾参照は、解決に失敗します。クエリーが search_path に依存しないように、それらをスキーマ修飾たとえば myschema.mytable 。

ここでは、上記の Cluster の例で参照される、単一のカスタムクエリーを含む ConfigMap の例を示します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: example-monitoring
  namespace: test
  labels:
    k8s.enterprisedb.io/reload: ""
data:
  custom-queries: |
    pg_replication:
      query: "SELECT CASE WHEN NOT pg_catalog.pg_is_in_recovery()
              THEN 0
              ELSE GREATEST (0,
                EXTRACT(EPOCH FROM (pg_catalog.now() OPERATOR(pg_catalog.-) pg_
↪catalog.pg_last_xact_replay_timestamp())))
              END AS lag,
```

(次のページに続く)

(前のページからの続き)

```
            pg_catalog.pg_is_in_recovery() AS in_recovery,
            EXISTS (TABLE pg_catalog.pg_stat_wal_receiver) AS is_wal_receiver_up,
            (SELECT pg_catalog.count(*) FROM pg_catalog.pg_stat_replication) AS␣
↪streaming_replicas"

    metrics:
      - lag:
          usage: "GAUGE"
          description: "Replication lag behind primary in seconds"
      - in_recovery:
          usage: "GAUGE"
          description: "Whether the instance is in recovery"
      - is_wal_receiver_up:
          usage: "GAUGE"
          description: "Whether the instance wal_receiver is up"
      - streaming_replicas:
          usage: "GAUGE"
          description: "Number of streaming replicas connected to the instance"
```

基本的なモニタリングクエリーのリストは、 *default-monitoring.yaml*

default-monitoring.yaml

これは、CloudNativePG クラスター展開の EDB Postgres AI に既にインストールされています **メトリックのデフォルトセット** を参照してください。

述語クエリーを使用したユーザー定義メトリックの例

`predicate_query` オプションを使用すると、ユーザーは `query` を実行して、指定された条件でのみメトリックを収集できます。これを行うには、ユーザーは単一の `boolean` 列で最大 1 つの行を返す述語照会を提供する必要があります。

述語クエリーは、メインクエリーと同じトランザクションで、同じデータベースに対して実行されます。

```
some_query: |
  predicate_query: |
    SELECT
      some_bool as predicate
    FROM some_table
  query: |
    SELECT
     pg_catalog.count(*) as rows
    FROM some_table
  metrics:
    - rows:
        usage: "GAUGE"
        description: "number of rows"
```

複数のデータベースで実行されているユーザー定義メトリックの例

target_databases オプションが複数のデータベースをリストする場合、メトリックは各データベースから収集されます。

target_databases のリストで_shell-like pattern_つまり、* 、? または [] を含むを指定することにより、特定のクエリーに対してデータベース自動検出を有効にできます。提供されている場合、オペレーターは、SELECT datname FROM pg_catalog.pg_database WHERE datallowconn AND NOT datistemplate AND pg_catalog.has_database_privilege(datname, 'CONNECT') の実行によって返されたすべてのデータベースを追加し、 path.Match() ルールに従ってパターンと一致することにより、ターゲットデータベースのリストを展開します。cnp_metrics_exporter に CONNECT 特権が欠如しているデータベースは、サイレントにスキップされます。取り消された PUBLIC アクセスを持つデータベースをスクレイピングする場合は、CONNECT を明示的に付与します上記の「カスタムクエリー特権と安全性」を参照してください。

> **注釈**
>
> * 文字は、yaml に special meaning があるため、このようなパターンが含まれる場合、*target_databases* 値を引用符で `"*"` する必要があります。

返されるラベルには、常にデータベースの名前を含めることをお勧めします。たとえば、次の例のように pg_catalog.current_database() ファンクションを使用します。

```yaml
some_query: |
  query: |
    SELECT
     pg_catalog.current_database() as datname,
     pg_catalog.count(*) as rows
    FROM some_table
  metrics:
    - datname:
        usage: "LABEL"
        description: "Name of current database"
    - rows:
        usage: "GAUGE"
        description: "number of rows"
  target_databases:
    - albert
    - bb
    - freddie
```

これにより、次のメトリックが公開されます。

```
cnp_some_query_rows{datname="albert"} 2
cnp_some_query_rows{datname="bb"} 5
cnp_some_query_rows{datname="freddie"} 10
```

次に、自動検出を有効にしたクエリーの例を示します。これは、 template1 データベースでも実行されますそれ以外の場合、前述のクエリーによって返されません。

```
some_query: |
  query: |
    SELECT
     pg_catalog.current_database() as datname,
     pg_catalog.count(*) as rows
    FROM some_table
  metrics:
    - datname:
        usage: "LABEL"
        description: "Name of current database"
    - rows:
        usage: "GAUGE"
        description: "number of rows"
  target_databases:
    - "*"
    - "template1"
```

上記の例では、次のメトリックを生成しますデータベースが存在する場合。

```
cnp_some_query_rows{datname="albert"} 2
cnp_some_query_rows{datname="bb"} 5
cnp_some_query_rows{datname="freddie"} 10
cnp_some_query_rows{datname="template1"} 7
cnp_some_query_rows{datname="postgres"} 42
```

ユーザー定義メトリックの構造

すべてのカスタムクエリーには、次の基本構造があります。

```
<MetricName>:
  query: "<SQLQuery>"
  metrics:
    - <ColumnName>:
        usage: "<MetricType>"
        description: "<MetricDescription>"
```

次に、使用可能なすべてのフィールドの簡単な説明を示します。

- <MetricName> プロメテウスの名前 metric
- name オーバーライド<MetricName> 、定義されている場合
- query メトリックを生成するためにターゲットデータベースで実行する SQL クエリー
- primary プライマリインスタンスでのみクエリーを実行するかどうか
- edb_fortran_20 Prometheus PostgreSQL エクスポータの互換性-非推奨)

- runonserver クエリを実行する PostgreSQL のバージョンを制限するセマンティックバージョンレンジたとえば">=11.0.0" または">=12.0.0 <=15.0.0"
- target_databases query を実行するデータベースのリスト、またはアシス edb_lk_19

自動検出を有効にします。デフォルトのデータベースが提供されている場合は上書きします。- predicate_query ターゲットデータベースで実行する最大 1 つの行と 1 つの boolean 列を返す SQL クエリー。システムは述語を評価し、true が query . - metrics エクスポートされたすべての列のリストを含むセクション。次のように定義します。- <ColumnName> クエリーによって返された列の名前 - name 列の ColumnName をオーバーライドしますメトリック、定義されている場合 b_tran_17 usage 以下で説明する値のいずれか - description メトリックの説明 - metrics_mapping usage が MAPPEDMETRIC に設定されている場合のオプションの列マッピング

usage の可能な値は次のとおりです。

Column Usage Label	Description
DISCARD	この列は無視する必要があります
LABEL	この列をラベルとして使用します
COUNTER	この列をカウンターとして使用する
GAUGE	この列をゲージとして使用します
MAPPEDMETRIC	指定されたテキスト値のマッピングでこの列を使用します
DURATION	この列をテキストの長さミリ秒単位として使用します
HISTOGRAM	この列をヒストグラムとして使用します

Metric Types にアクセスしてください。

詳細については、Prometheus のドキュメントから 。

ユーザー定義メトリックの出力

カスタム定義メトリックは、次の形式で Prometheus エクスポータエンドポイント:9187/metrics によって返されます。

```
cnp_<MetricName>_<ColumnName>{<LabelColumnName>=<LabelColumnValue> ... } <ColumnValue>
```

注釈

LabelColumnName は、 *usage* が `LABEL` に設定されたメトリックとその `Value`

上記の pg_replication の例を考慮すると、エクスポーターのエンドポイントは、呼び出されると次の出力を結果ます。

```
#  HELP cnp_pg_replication_in_recovery Whether the instance is in recovery

#  TYPE cnp_pg_replication_in_recovery gauge

```

(次のページに続く)

(前のページからの続き)

```
cnp_pg_replication_in_recovery 0

#  HELP cnp_pg_replication_lag Replication lag behind primary in seconds

#  TYPE cnp_pg_replication_lag gauge

cnp_pg_replication_lag 0

#  HELP cnp_pg_replication_streaming_replicas Number of streaming replicas connected␣
↪to the instance

#  TYPE cnp_pg_replication_streaming_replicas gauge

cnp_pg_replication_streaming_replicas 2

#  HELP cnp_pg_replication_is_wal_receiver_up Whether the instance wal_receiver is up

#  TYPE cnp_pg_replication_is_wal_receiver_up gauge

cnp_pg_replication_is_wal_receiver_up 0
```

メトリックのデフォルトセット

オペレーターは、オペレーターの名前空間内の ConfigMap または Secret で定義された一連のモニタリングクエリーをクラスターに自動的に注入するように構成できます。*Operator configuration* の MONITORING_QUERIES_CONFIGMAP または MONITORING_QUERIES_SECRET キーを、それぞれ ConfigMap または Secret の名前に設定する必要があります。オペレーターは queries キーの内容を使用します。

queries コンテンツを変更すると、それを使用するデプロイされたすべてのクラスターにすぐに反映されます。

オペレーターインストールマニフェストには、すべてのクラスターが使用する postgresql-operator-default-monitoring と呼ばれる事前定義された ConfigMap が付属しています。MONITORING_QUERIES_CONFIGMAP は、オペレーター構成でデフォルトで postgresql-operator-default-monitoring に設定されます。

デフォルトのメトリックセットを無効にする場合は、次のことができます。

- オペレーターレベルで無効にする オペレーター ConfigMap で MONITORING_QUERIES_CONFIGMAP / MONITORING_QUERIES_SECRET キーを"" 空の文字列に設定します。オペレーターの ConfigMap を変更するには、オペレーターの再起動が必要です。
- 特定のクラスターで無効にします。クラスターで.spec.monitoring.disableDefaultQueries を true に設定します。

Prometheus Postgres エクスポーターとの違い

CloudNativePG クラスターの EDB Postgres AI は、PostgreSQL Prometheus Exporter からインスピレーションを得ていますが、いくつかの違いがあります。特に、cache_seconds フィールドは、CloudNativePG クラスターエクスポーターの EDB Postgres AI に実装されていません。

メトリックエクスポーターロールの手動作成

オペレーターは、リコンシリエーション中にプライマリで cnp_metrics_exporter PostgreSQL ロールを作成します。次に、ストリーミングレプリケーションを介してスタンバイとレプリカクラスターに伝播します。

ロールが欠落している場合プライマリの前にアップグレードされたレプリカクラスター、ロールより前のバックアップからの復元、誤って削除レプリカクラスター)

```
CREATE ROLE cnp_metrics_exporter WITH LOGIN NOSUPERUSER NOCREATEDB
    NOCREATEROLE NOREPLICATION NOBYPASSRLS INHERIT;
GRANT pg_monitor TO cnp_metrics_exporter;
```

カスタムモニタリングクエリーが pg_monitor のスコープ外のオブジェクトへのアクセスが必要な場合、必要な特権を明示的に付与します。テーブルの SELECT には、スキーマでの USAGE も必要です。

```
GRANT USAGE ON SCHEMA myschema TO cnp_metrics_exporter;
GRANT SELECT ON TABLE myschema.mytable TO cnp_metrics_exporter;
```

5.22.2 CloudNativePG クラスターオペレーターの EDB Postgres AI のモニタリング

オペレーターは、metrics という名前のポート 8080 で HTTP を介して Prometheus メトリックを内部公開します。

> **注釈**
>
> エクスポートされたメトリックを検査する方法 の指示に従って、エクスポートされたメトリックを検査できます。

以下のセクション。

現在、オペレーターはデフォルトの kubebuilder メトリックを公開しています。 kubebuilder documentation を参照

詳細については、

Prometheus でのオペレーターのモニタリング

PodMonitor を定義することにより、 Prometheus Operator を使用してオペレーターをモニタできます。

次のように、オペレーターポッドをポイントしますオペレーターと同じ名前空間に適用されることに注意してください。

```
kubectl -n postgresql-operator-system apply -f - <<EOF
- --
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: postgresql-operator-controller-manager
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: cloud-native-postgresql
  podMetricsEndpoints:
    - port: metrics
EOF
```

オペレーターメトリックの TLS の有効化

デフォルトでは、オペレーターは HTTP を介してポート 8080 でメトリックを公開します。TLS でこのエンドポイントを保護するには、次の手順を実行します。

1. TLS 証明書 tls.crt とプライベートキー tls.key を含む Kubernetes シークレットを作成します。
2. シークレットをオペレーターポッドにマウントします。
3. METRICS_CERT_DIR 環境変数を設定して、証明書がマウントされているディレクトリをポイントします。

Secret 定義の例

```
apiVersion: v1
kind: Secret
metadata:
  name: postgresql-operator-metrics-cert
  namespace: postgresql-operator-system
type: kubernetes.io/tls
data:
  tls.crt: <base64-encoded-certificate>
  tls.key: <base64-encoded-key>
```

次に、オペレーターの展開を更新してシークレットをマウントし、環境変数を構成します。

```
spec:
  template:
    spec:
      containers:
        - name: manager
          env:
            - name: METRICS_CERT_DIR
```

(次のページに続く)

(前のページからの続き)

```
            value: /run/secrets/k8s.enterprisedb.io/metrics
          volumeMounts:
            - mountPath: /run/secrets/k8s.enterprisedb.io/metrics
              name: metrics-certificates
              readOnly: true
      volumes:
        - name: metrics-certificates
          secret:
            secretName: postgresql-operator-metrics-cert
            defaultMode: 420
```

注釈

METRICS_CERT_DIR が設定されている場合、オペレーターはメトリックサーバーの TLS を自動的に有効にします。 *https* スキームを使用するように PodMonitor 構成を更新する必要もあります。

TLS が有効になった PodMonitor 構成の例

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: postgresql-operator-controller-manager
  namespace: postgresql-operator-system
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: cloud-native-postgresql
  podMetricsEndpoints:
    - port: metrics
      scheme: https
      tlsConfig:
        insecureSkipVerify: true # or configure proper CA validation
```

5.22.3 エクスポートされたメトリックを検査する方法

このセクションでは、特定の PostgreSQL インスタンスマネージャープライマリまたはレプリカまたはオペレーターによってエクスポートされたメトリックを検査する方法に関する基本的な手順を提供します。

注釈

以下の例では、デフォルトの名前空間で、*postgresql-operator-system* 名前空間にインストールされたオペレーターを使用して作業していると仮定しています。ユースケースに合わせて調整してください。

ポート転送を使用する

メトリックを検査する最も簡単な方法は、関係するポッドのメトリックポートをポート転送することです。

たとえば、 cluster-example の-1 インスタンスのメトリックを検査するには、9187 ポートをポート転送します。

```
kubectl port-forward cluster-example-1 9187:9187
```

ポート転送がアクティブになると、メトリックは、Web ブラウザーで、構成に応じて HTTP または HTTPS を使用して、アドレス localhost:9187/metrics で簡単に検査できます。

オペレーターポッドは、ポート 8080 でメトリックもエクスポートします。インスタンスと同様に、オペレーター名前空間にあるオペレーターポッドをポート転送します。

```
kubectl -n postgresql-operator-system port-forward pod/<CONTROLLER-MANAGER-POD>␣
↪8080:8080
```

ポート転送がアクティブになると、メトリックはブラウザーで localhost:8080/metrics で簡単に表示できます。

カールを使用する

次のコマンドを使用して curl ポッドを作成します。

```
kubectl apply -f - <<EOF
- --
apiVersion: v1
kind: Pod
metadata:
  name: curl
spec:
  containers:
  - name: curl
    image: curlimages/curl:8.21.0
    command: [sleep, 3600]
EOF
```

インスタンスによってエクスポートされたメトリックを検査するには、ターゲットポッドのポート 9187 に接続する必要があります。ポッドの IP アドレスを知る必要があります。これは、kubectl get pod -o wide を実行すると簡単に見つけられます。次の汎用コマンドは、目的のポッドで curl を実行します。

```
kubectl exec -ti curl -- curl -s <pod_ip>:9187/metrics
```

たとえば、PostgreSQL クラスターが cluster-example と呼ばれ、クラスター内の最初のポッドのエクスポートされたメトリックを取得する場合、次のコマンドを実行してそのポッドの IP をプログラム的に取得できます。

```
POD_IP=$(kubectl get pod cluster-example-1 --template {{.status.podIP}})
```

そして、実行します。

```
kubectl exec -ti curl -- curl -s ${POD_IP}:9187/metrics
```

TLS メトリックを有効にした場合、代わりに実行します。

```
kubectl exec -ti curl -- curl -sk https://${POD_IP}:9187/metrics
```

オペレーターのメトリックにアクセスするには、オペレーターが実行されているポッドをポイントし、ターゲットとして TCP ポート 8080 を使用する必要があります。

メトリックの検査が終了したら、`curl` ポッドを削除することを忘れてください。

```
kubectl delete -f curl.yaml
```

5.22.4 補助リソース

doc/src/samples/monitoring/

ディレクトリには、可観測性のための一連のサンプルファイルがあります。*Part 4 of the quickstart* を参照してください。

コンテキストのセクション

- `kube-stack-config.yaml` kube-stack Helm chart インストールの構成ファイル。これにより、Prometheus がすべての PodMonitor リソースをリッスンすることが保証されます。
- `prometheusrule.yaml` CloudNativePG クラスターの EDB Postgres AI のアラートを含む `PrometheusRule` 。注これには、通知サービスとの相互運用は含まれません。Prometheus documentation を参照してください。
- `podmonitor.yaml` CloudNativePG Cluster Operator 展開の EDB Postgres AI の `PodMonitor` 。

さらに、Prometheus アラートルールの「生」ソースを `alerts.yaml` ファイルで提供します。

CloudNativePG Cluster クラスターとオペレーターの EDB Postgres AI の Grafana ダッシュボードは、専用のリポジトリ cloudnative-pg/grafana-dashboards に保持されます

ダッシュボード JSON 構成として grafana-dashboard.json 。ファイルをダウンロードし、Grafana にインポートできます。メニュー Dashboard > New > Import 。

`kube-prometheus-stack` で使用可能な設定に関する一般的なリファレンスについては、`helm show values prometheus-community/kube-prometheus-stack` を実行できます。 kube-prometheus-stack を参照してください。

詳細ページ。

5.22.5 OpenShift でのモニタリング

Openshift 4.6 以降、Monitoring for user-defined projects と呼ばれる完全なモニタリングスタックがあります

これは、クラスター管理者が有効にできます。 `Cluster` 定義のオプション `spec.monitoring.enablePodMonitor` が `true` に設定されている場合、クラウドネイティブ PostgreSQL は `PodMonitor` オブジェクトを自動的に作成します。

クラスター全体の user-defined モニタリングを有効にするには、最初に次の内容を使用して openshift-monitoring 名前空間/プロジェクトに cluster-monitoring-config という名前で ConfigMap を作成する必要があります。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    enableUserWorkload: true
```

ConfigMap が既に存在する場合、変数 enableUserWorkload: true を追加するだけです。

その後、これに似たものを使用して名前空間/プロジェクトに適切な PodMonitor を作成するだけです。

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: cluster-sample
spec:
  selector:
    matchLabels:
      postgresql: cluster-sample
  podMetricsEndpoints:
    - port: metrics
```

注釈

私たちのサービスはメトリックを指すポートを定義していないため、現在 `ServiceMonitor` は使用しません。メトリックポートを追加すると、機密データが公開される可能性があります。

default-monitoring.yaml

```
- --
apiVersion: v1
kind: ConfigMap
metadata:
  name: default-monitoring
  labels:
    k8s.enterprisedb.io/reload: ""
data:
  queries: |
    backends:
      query: |
```

(次のページに続く)

(前のページからの続き)

```
      SELECT sa.datname
          , sa.usename
          , sa.application_name
          , states.state
          , COALESCE(sa.count, 0) AS total
          , COALESCE(sa.max_tx_secs, 0) AS max_tx_duration_seconds
          FROM ( VALUES (active)
              , (idle)
              , (idle in transaction)
              , (idle in transaction (aborted))
              , (fastpath function call)
              , (disabled)
              ) AS states(state)
          LEFT JOIN (
              SELECT datname
                  , state
                  , usename
                  , COALESCE(application_name, ) AS application_name
                  , pg_catalog.count(*)
                  , COALESCE(EXTRACT (EPOCH FROM (pg_catalog.max(pg_catalog.now()␣
↪OPERATOR(pg_catalog.-) xact_start))), 0) AS max_tx_secs
              FROM pg_catalog.pg_stat_activity
              GROUP BY datname, state, usename, application_name
          ) sa ON states.state OPERATOR(pg_catalog.=) sa.state
          WHERE sa.usename IS NOT NULL
    metrics:
      - datname:
          usage: "LABEL"
          description: "Name of the database"
      - usename:
          usage: "LABEL"
          description: "Name of the user"
      - application_name:
          usage: "LABEL"
          description: "Name of the application"
      - state:
          usage: "LABEL"
          description: "State of the backend"
      - total:
          usage: "GAUGE"
          description: "Number of backends"
      - max_tx_duration_seconds:
          usage: "GAUGE"
          description: "Maximum duration of a transaction in seconds"
```

(次のページに続く)

(前のページからの続き)

```
    backends_waiting:
      query: |
       SELECT pg_catalog.count(*) AS total
       FROM pg_catalog.pg_locks blocked_locks
       JOIN pg_catalog.pg_locks blocking_locks
         ON blocking_locks.locktype OPERATOR(pg_catalog.=) blocked_locks.locktype
         AND blocking_locks.database IS NOT DISTINCT FROM blocked_locks.database
         AND blocking_locks.relation IS NOT DISTINCT FROM blocked_locks.relation
         AND blocking_locks.page IS NOT DISTINCT FROM blocked_locks.page
         AND blocking_locks.tuple IS NOT DISTINCT FROM blocked_locks.tuple
         AND blocking_locks.virtualxid IS NOT DISTINCT FROM blocked_locks.virtualxid
         AND blocking_locks.transactionid IS NOT DISTINCT FROM blocked_locks.
↪transactionid
         AND blocking_locks.classid IS NOT DISTINCT FROM blocked_locks.classid
         AND blocking_locks.objid IS NOT DISTINCT FROM blocked_locks.objid
         AND blocking_locks.objsubid IS NOT DISTINCT FROM blocked_locks.objsubid
         AND blocking_locks.pid OPERATOR(pg_catalog.<>) blocked_locks.pid
       JOIN pg_catalog.pg_stat_activity blocking_activity ON blocking_activity.pid␣
↪OPERATOR(pg_catalog.=) blocking_locks.pid
        WHERE NOT blocked_locks.granted
      metrics:
        - total:
            usage: "GAUGE"
            description: "Total number of backends that are currently waiting on␣
↪other queries"

    pg_database:
      query: |
        SELECT datname
          , pg_catalog.pg_database_size(datname) AS size_bytes
          , pg_catalog.age(datfrozenxid) AS xid_age
          , pg_catalog.mxid_age(datminmxid) AS mxid_age
        FROM pg_catalog.pg_database
        WHERE datallowconn
      metrics:
        - datname:
            usage: "LABEL"
            description: "Name of the database"
        - size_bytes:
            usage: "GAUGE"
            description: "Disk space used by the database"
        - xid_age:
            usage: "GAUGE"
            description: "Number of transactions from the frozen XID to the current␣
↪one"
```

(次のページに続く)

(前のページからの続き)

```
      - mxid_age:
          usage: "GAUGE"
          description: "Number of multiple transactions (Multixact) from the frozen␣
↪XID to the current one"

  pg_postmaster:
    query: |
      SELECT EXTRACT(EPOCH FROM pg_postmaster_start_time) AS start_time
      FROM pg_catalog.pg_postmaster_start_time()
    metrics:
      - start_time:
          usage: "GAUGE"
          description: "Time at which postgres started (based on epoch)"

  pg_replication:
    query: "SELECT CASE WHEN (
              NOT pg_catalog.pg_is_in_recovery()
              OR pg_catalog.pg_last_wal_receive_lsn() OPERATOR(pg_catalog.=) pg_
↪catalog.pg_last_wal_replay_lsn())
            THEN 0
            ELSE GREATEST (0,
              EXTRACT(EPOCH FROM (pg_catalog.now() OPERATOR(pg_catalog.-) pg_
↪catalog.pg_last_xact_replay_timestamp())))
            END AS lag,
            pg_catalog.pg_is_in_recovery() AS in_recovery,
            EXISTS (TABLE pg_catalog.pg_stat_wal_receiver) AS is_wal_receiver_up,
            (SELECT pg_catalog.count(*) FROM pg_catalog.pg_stat_replication) AS␣
↪streaming_replicas"
    metrics:
      - lag:
          usage: "GAUGE"
          description: "Replication lag behind primary in seconds"
      - in_recovery:
          usage: "GAUGE"
          description: "Whether the instance is in recovery"
      - is_wal_receiver_up:
          usage: "GAUGE"
          description: "Whether the instance wal_receiver is up"
      - streaming_replicas:
          usage: "GAUGE"
          description: "Number of streaming replicas connected to the instance"

  pg_replication_slots:
    query: |
      SELECT slot_name,
```

(次のページに続く)

(前のページからの続き)

```
          slot_type,
          database,
          active,
          (CASE pg_catalog.pg_is_in_recovery()
            WHEN TRUE THEN pg_catalog.pg_wal_lsn_diff(pg_catalog.pg_last_wal_receive_
↪lsn(), restart_lsn)
            ELSE pg_catalog.pg_wal_lsn_diff(pg_catalog.pg_current_wal_lsn(), restart_
↪lsn)
          END) as pg_wal_lsn_diff
        FROM pg_catalog.pg_replication_slots
        WHERE NOT temporary
      metrics:
        - slot_name:
            usage: "LABEL"
            description: "Name of the replication slot"
        - slot_type:
            usage: "LABEL"
            description: "Type of the replication slot"
        - database:
            usage: "LABEL"
            description: "Name of the database"
        - active:
            usage: "GAUGE"
            description: "Flag indicating whether the slot is active"
        - pg_wal_lsn_diff:
            usage: "GAUGE"
            description: "Replication lag in bytes"

    pg_stat_archiver:
      query: |
        SELECT archived_count
          , failed_count
          , COALESCE(EXTRACT(EPOCH FROM (pg_catalog.now() OPERATOR(pg_catalog.-) last_
↪archived_time)), -1) AS seconds_since_last_archival
          , COALESCE(EXTRACT(EPOCH FROM (pg_catalog.now() OPERATOR(pg_catalog.-) last_
↪failed_time)), -1) AS seconds_since_last_failure
          , COALESCE(EXTRACT(EPOCH FROM last_archived_time), -1) AS last_archived_time
          , COALESCE(EXTRACT(EPOCH FROM last_failed_time), -1) AS last_failed_time
          , COALESCE(CAST(CAST(x OPERATOR(pg_catalog.||) pg_catalog.right(pg_catalog.
↪split_part(last_archived_wal, ., 1), 16) AS pg_catalog.bit(64)) AS pg_catalog.int8),
 -1) AS last_archived_wal_start_lsn
          , COALESCE(CAST(CAST(x OPERATOR(pg_catalog.||) pg_catalog.right(pg_catalog.
↪split_part(last_failed_wal, ., 1), 16) AS pg_catalog.bit(64)) AS pg_catalog.int8), -
↪1) AS last_failed_wal_start_lsn
          , EXTRACT(EPOCH FROM stats_reset) AS stats_reset_time
```

(次のページに続く)

(前のページからの続き)

```
      FROM pg_catalog.pg_stat_archiver
    predicate_query: |
      SELECT NOT pg_catalog.pg_is_in_recovery()
        OR pg_catalog.current_setting(archive_mode) OPERATOR(pg_catalog.=) always
    metrics:
      - archived_count:
          usage: "COUNTER"
          description: "Number of WAL files that have been successfully archived"
      - failed_count:
          usage: "COUNTER"
          description: "Number of failed attempts for archiving WAL files"
      - seconds_since_last_archival:
          usage: "GAUGE"
          description: "Seconds since the last successful archival operation"
      - seconds_since_last_failure:
          usage: "GAUGE"
          description: "Seconds since the last failed archival operation"
      - last_archived_time:
          usage: "GAUGE"
          description: "Epoch of the last time WAL archiving succeeded"
      - last_failed_time:
          usage: "GAUGE"
          description: "Epoch of the last time WAL archiving failed"
      - last_archived_wal_start_lsn:
          usage: "GAUGE"
          description: "Archived WAL start LSN"
      - last_failed_wal_start_lsn:
          usage: "GAUGE"
          description: "Last failed WAL LSN"
      - stats_reset_time:
          usage: "GAUGE"
          description: "Time at which these statistics were last reset"

  pg_stat_bgwriter:
    runonserver: "<17.0.0"
    query: |
      SELECT checkpoints_timed
        , checkpoints_req
        , checkpoint_write_time
        , checkpoint_sync_time
        , buffers_checkpoint
        , buffers_clean
        , maxwritten_clean
        , buffers_backend
        , buffers_backend_fsync
```

(次のページに続く)

(前のページからの続き)

```
          , buffers_alloc
        FROM pg_catalog.pg_stat_bgwriter
      metrics:
        - checkpoints_timed:
            usage: "COUNTER"
            description: "Number of scheduled checkpoints that have been performed"
        - checkpoints_req:
            usage: "COUNTER"
            description: "Number of requested checkpoints that have been performed"
        - checkpoint_write_time:
            usage: "COUNTER"
            description: "Total amount of time that has been spent in the portion of␣
↪checkpoint processing where files are written to disk, in milliseconds"
        - checkpoint_sync_time:
            usage: "COUNTER"
            description: "Total amount of time that has been spent in the portion of␣
↪checkpoint processing where files are synchronized to disk, in milliseconds"
        - buffers_checkpoint:
            usage: "COUNTER"
            description: "Number of buffers written during checkpoints"
        - buffers_clean:
            usage: "COUNTER"
            description: "Number of buffers written by the background writer"
        - maxwritten_clean:
            usage: "COUNTER"
            description: "Number of times the background writer stopped a cleaning␣
↪scan because it had written too many buffers"
        - buffers_backend:
            usage: "COUNTER"
            description: "Number of buffers written directly by a backend"
        - buffers_backend_fsync:
            usage: "COUNTER"
            description: "Number of times a backend had to execute its own fsync call␣
↪(normally the background writer handles those even when the backend does its own␣
↪write)"
        - buffers_alloc:
            usage: "COUNTER"
            description: "Number of buffers allocated"

    pg_stat_bgwriter_17:
      runonserver: ">=17.0.0"
      name: pg_stat_bgwriter
      query: |
        SELECT buffers_clean
          , maxwritten_clean
```

(次のページに続く)

(前のページからの続き)

```
          , buffers_alloc
          , EXTRACT(EPOCH FROM stats_reset) AS stats_reset_time
        FROM pg_catalog.pg_stat_bgwriter
      metrics:
        - buffers_clean:
            usage: "COUNTER"
            description: "Number of buffers written by the background writer"
        - maxwritten_clean:
            usage: "COUNTER"
            description: "Number of times the background writer stopped a cleaning␣
↪scan because it had written too many buffers"
        - buffers_alloc:
            usage: "COUNTER"
            description: "Number of buffers allocated"
        - stats_reset_time:
            usage: "GAUGE"
            description: "Time at which these statistics were last reset"

    pg_stat_checkpointer:
      runonserver: ">=17.0.0"
      query: |
        SELECT num_timed AS checkpoints_timed
          , num_requested AS checkpoints_req
          , restartpoints_timed
          , restartpoints_req
          , restartpoints_done
          , write_time
          , sync_time
          , buffers_written
          , EXTRACT(EPOCH FROM stats_reset) AS stats_reset_time
        FROM pg_catalog.pg_stat_checkpointer
      metrics:
        - checkpoints_timed:
            usage: "COUNTER"
            description: "Number of scheduled checkpoints that have been performed"
        - checkpoints_req:
            usage: "COUNTER"
            description: "Number of requested checkpoints that have been performed"
        - restartpoints_timed:
            usage: "COUNTER"
            description: "Number of scheduled restartpoints due to timeout or after a␣
↪failed attempt to perform it"
        - restartpoints_req:
            usage: "COUNTER"
            description: "Number of requested restartpoints that have been performed"
```

(次のページに続く)

(前のページからの続き)

```
      - restartpoints_done:
          usage: "COUNTER"
          description: "Number of restartpoints that have been performed"
      - write_time:
          usage: "COUNTER"
          description: "Total amount of time that has been spent in the portion of␣
↪processing checkpoints and restartpoints where files are written to disk, in␣
↪milliseconds"
      - sync_time:
          usage: "COUNTER"
          description: "Total amount of time that has been spent in the portion of␣
↪processing checkpoints and restartpoints where files are synchronized to disk, in␣
↪milliseconds"
      - buffers_written:
          usage: "COUNTER"
          description: "Number of buffers written during checkpoints and␣
↪restartpoints"
      - stats_reset_time:
          usage: "GAUGE"
          description: "Time at which these statistics were last reset"

  pg_stat_database:
    query: |
      SELECT datname
        , xact_commit
        , xact_rollback
        , blks_read
        , blks_hit
        , tup_returned
        , tup_fetched
        , tup_inserted
        , tup_updated
        , tup_deleted
        , conflicts
        , temp_files
        , temp_bytes
        , deadlocks
        , blk_read_time
        , blk_write_time
      FROM pg_catalog.pg_stat_database
    metrics:
      - datname:
          usage: "LABEL"
          description: "Name of this database"
      - xact_commit:
```

(次のページに続く)

(前のページからの続き)

```
            usage: "COUNTER"
            description: "Number of transactions in this database that have been␣
↪committed"
        - xact_rollback:
            usage: "COUNTER"
            description: "Number of transactions in this database that have been␣
↪rolled back"
        - blks_read:
            usage: "COUNTER"
            description: "Number of disk blocks read in this database"
        - blks_hit:
            usage: "COUNTER"
            description: "Number of times disk blocks were found already in the␣
↪buffer cache, so that a read was not necessary (this only includes hits in the␣
↪PostgreSQL buffer cache, not the operating systems file system cache)"
        - tup_returned:
            usage: "COUNTER"
            description: "Number of rows returned by queries in this database"
        - tup_fetched:
            usage: "COUNTER"
            description: "Number of rows fetched by queries in this database"
        - tup_inserted:
            usage: "COUNTER"
            description: "Number of rows inserted by queries in this database"
        - tup_updated:
            usage: "COUNTER"
            description: "Number of rows updated by queries in this database"
        - tup_deleted:
            usage: "COUNTER"
            description: "Number of rows deleted by queries in this database"
        - conflicts:
            usage: "COUNTER"
            description: "Number of queries canceled due to conflicts with recovery␣
↪in this database"
        - temp_files:
            usage: "COUNTER"
            description: "Number of temporary files created by queries in this␣
↪database"
        - temp_bytes:
            usage: "COUNTER"
            description: "Total amount of data written to temporary files by queries␣
↪in this database"
        - deadlocks:
            usage: "COUNTER"
            description: "Number of deadlocks detected in this database"
```

(次のページに続く)

(前のページからの続き)

```
        - blk_read_time:
            usage: "COUNTER"
            description: "Time spent reading data file blocks by backends in this␣
↪database, in milliseconds"
        - blk_write_time:
            usage: "COUNTER"
            description: "Time spent writing data file blocks by backends in this␣
↪database, in milliseconds"

    pg_stat_replication:
      primary: true
      query: |
       SELECT usename
         , COALESCE(application_name, ) AS application_name
         , COALESCE(client_addr::text, ) AS client_addr
         , COALESCE(client_port::text, ) AS client_port
         , EXTRACT(EPOCH FROM backend_start) AS backend_start
         , COALESCE(pg_catalog.age(backend_xmin), 0) AS backend_xmin_age
         , pg_catalog.pg_wal_lsn_diff(pg_catalog.pg_current_wal_lsn(), sent_lsn) AS␣
↪sent_diff_bytes
         , pg_catalog.pg_wal_lsn_diff(pg_catalog.pg_current_wal_lsn(), write_lsn) AS␣
↪write_diff_bytes
         , pg_catalog.pg_wal_lsn_diff(pg_catalog.pg_current_wal_lsn(), flush_lsn) AS␣
↪flush_diff_bytes
         , COALESCE(pg_catalog.pg_wal_lsn_diff(pg_catalog.pg_current_wal_lsn(),␣
↪replay_lsn),0) AS replay_diff_bytes
         , COALESCE((EXTRACT(EPOCH FROM write_lag)),0)::float AS write_lag_seconds
         , COALESCE((EXTRACT(EPOCH FROM flush_lag)),0)::float AS flush_lag_seconds
         , COALESCE((EXTRACT(EPOCH FROM replay_lag)),0)::float AS replay_lag_seconds
       FROM pg_catalog.pg_stat_replication
      metrics:
        - usename:
            usage: "LABEL"
            description: "Name of the replication user"
        - application_name:
            usage: "LABEL"
            description: "Name of the application"
        - client_addr:
            usage: "LABEL"
            description: "Client IP address"
        - client_port:
            usage: "LABEL"
            description: "Client TCP port"
        - backend_start:
            usage: "COUNTER"
```

(次のページに続く)

(前のページからの続き)

```
            description: "Time when this process was started"
        - backend_xmin_age:
            usage: "COUNTER"
            description: "The age of this standbys xmin horizon"
        - sent_diff_bytes:
            usage: "GAUGE"
            description: "Difference in bytes from the last write-ahead log location␣
↪sent on this connection"
        - write_diff_bytes:
            usage: "GAUGE"
            description: "Difference in bytes from the last write-ahead log location␣
↪written to disk by this standby server"
        - flush_diff_bytes:
            usage: "GAUGE"
            description: "Difference in bytes from the last write-ahead log location␣
↪flushed to disk by this standby server"
        - replay_diff_bytes:
            usage: "GAUGE"
            description: "Difference in bytes from the last write-ahead log location␣
↪replayed into the database on this standby server"
        - write_lag_seconds:
            usage: "GAUGE"
            description: "Time elapsed between flushing recent WAL locally and␣
↪receiving notification that this standby server has written it"
        - flush_lag_seconds:
            usage: "GAUGE"
            description: "Time elapsed between flushing recent WAL locally and␣
↪receiving notification that this standby server has written and flushed it"
        - replay_lag_seconds:
            usage: "GAUGE"
            description: "Time elapsed between flushing recent WAL locally and␣
↪receiving notification that this standby server has written, flushed and applied it"

    pg_settings:
      query: |
        SELECT name,
        CASE setting WHEN on THEN 1 WHEN off THEN 0 ELSE setting END AS setting
        FROM pg_catalog.pg_settings
        WHERE vartype IN (integer, real, bool)
        ORDER BY 1
      metrics:
        - name:
            usage: "LABEL"
            description: "Name of the setting"
        - setting:
```

(次のページに続く)

(前のページからの続き)

```
          usage: "GAUGE"
          description: "Setting value"

  pg_extensions:
    query: |
      SELECT
       pg_catalog.current_database() as datname,
       name as extname,
       default_version,
       installed_version,
       CASE
         WHEN default_version OPERATOR(pg_catalog.=) installed_version THEN 0
         ELSE 1
      END AS update_available
      FROM pg_catalog.pg_available_extensions
      WHERE installed_version IS NOT NULL
    metrics:
      - datname:
          usage: "LABEL"
          description: "Name of the database"
      - extname:
          usage: "LABEL"
          description: "Extension name"
      - default_version:
          usage: "LABEL"
          description: "Default version"
      - installed_version:
          usage: "LABEL"
          description: "Installed version"
      - update_available:
          usage: "GAUGE"
          description: "An update is available"
    target_databases:
      - *
```

5.23 Logging

CloudNativePG クラスターの EDB Postgres AI は、セキュリティ上の理由でストレージに永続化せずに、JSON 形式でログを PostgreSQL ログを含む標準出力に直接出力します。このデザインにより、stern のようなコマンドラインのものを含む、ほとんどの Kubernetes 互換ログ管理ツールとのシームレスな統合が促進されます。

Kubernetes Logging Architecture を参照してください。

ドキュメント。

各ログエントリには、次のフィールドが含まれます。

- `level` – ログレベルたとえば `info` 、`notice` 。
- `ts` – タイムスタンプ。
- `logger` – ログのタイプ `postgres` 、`pg_controldata` など。
- `msg` – ログメッセージ、またはメッセージが JSON 形式の場合はキーワード `record`
- `record` – 実際のレコード。`logger` タイプに応じて異なる構造。
- `logging_pod` – ログが生成されたポッドの名前。

> **注釈**
>
> ログ取り込みシステムでカスタムフィールド名が必要な場合、オペレーターコントローラの `log-field-level` および `log-field-timestamp` フラグを使用して、*level* および `ts` フィールドの名前を変更できます。これは、 *cloudnative-pg* オペレーターの `Deployment` 定義を編集することにより構成できます。

5.23.1 クラスターログ

`logLevel` オプションを使用して、クラスター仕様でインスタンスポッドのログレベルを構成できます。使用可能なログレベルは `error` 、`warning` 、`info` デフォルト、`debug` 、および `trace` です。

5.23.2 オペレーターログ

オペレーターポッドによって生成されるログは、インスタンスポッドと同じログレベルで構成できます。`error` 、`warning` 、`info` デフォルト、`debug` 、および `trace` 。

オペレーターのログレベルは、オペレーターの `Deployment` 定義を編集し、`--log-level` コマンドライン引数を目的の値に設定することにより、構成できます。

5.23.3 PostgreSQL ログ

各 PostgreSQL ログエントリは、 `logger` キーが `postgres` に設定された JSON オブジェクトです。ログエントリの構造は次のとおりです。

```
{
  "level": "info",
  "ts": 1619781249.7188137,
  "logger": "postgres",
  "msg": "record",
  "record": {
    "log_time": "2021-04-30 11:14:09.718 UTC",
    "user_name": "",
    "database_name": "",
    "process_id": "25",
    "connection_from": "",
```

(次のページに続く)

(前のページからの続き)


```
    "session_id": "608be681.19",
    "session_line_num": "1",
    "command_tag": "",
    "session_start_time": "2021-04-30 11:14:09 UTC",
    "virtual_transaction_id": "",
    "transaction_id": "0",
    "error_severity": "LOG",
    "sql_state_code": "00000",
    "message": "database system was interrupted; last known up at 2021-04-30 11:14:07␣
↪UTC",
    "detail": "",
    "hint": "",
    "internal_query": "",
    "internal_query_pos": "",
    "context": "",
    "query": "",
    "query_pos": "",
    "location": "",
    "application_name": "",
    "backend_type": "startup"
  },
  "logging_pod": "cluster-example-1"
}
```


注釈

内部的に、オペレーターは PostgreSQL の CSV ログ形式を使用します。詳細については、

PostgreSQL documentation on CSV log format を参照してください。

5.23.4 PG 監査ログ

CloudNativePG クラスターの EDB Postgres AI は、PostgreSQL クラスターで PGAudit のシームレスなネイティブサポートを提供します。

PGAudit を有効にするには、クラスター構成の `postgresql` セクションに必要な `pgaudit` パラメーターを追加します。

さらに、オペレーターは、クラスター内のすべてのデータベースで PGAudit 拡張機能の作成と削除を管理します。

次の例は、PGAudit が有効および構成された PostgreSQL `Cluster` 展開を示しています。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
```

(次のページに続く)

(前のページからの続き)

```
metadata:
  name: cluster-example
spec:
  instances: 3

  postgresql:
    parameters:
      "pgaudit.log": "all, -misc"
      "pgaudit.log_catalog": "off"
      "pgaudit.log_parameter": "on"
      "pgaudit.log_relation": "on"

  storage:
    size: 1Gi
```

PGAudit によって生成された監査 CSV ログエントリは解析され、他のすべてのログと同様に、JSON 形式で標準出力にルーティングされます。

- `.logger` は `pgaudit` に設定されます。
- `.msg` は `record` に設定されます。
- `.record` には、解析されたレコード全体が JSON オブジェクトとして含まれています。この構造は、JSON オブジェクトとしてフォーマットされた PGAudit CSV メッセージを含む`.record.audit` を除き `logging_collector` ログの構造に似ています。

この例は、サンプルログエントリを示しています。

```
{
  "level": "info",
  "ts": 1627394507.8814096,
  "logger": "pgaudit",
  "msg": "record",
  "record": {
    "log_time": "2021-07-27 14:01:47.881 UTC",
    "user_name": "postgres",
    "database_name": "postgres",
    "process_id": "203",
    "connection_from": "[local]",
    "session_id": "610011cb.cb",
    "session_line_num": "1",
    "command_tag": "SELECT",
    "session_start_time": "2021-07-27 14:01:47 UTC",
    "virtual_transaction_id": "3/336",
    "transaction_id": "0",
    "error_severity": "LOG",
    "sql_state_code": "00000",
```

(次のページに続く)

(前のページからの続き)


```
    "backend_type": "client backend",
    "audit": {
      "audit_type": "SESSION",
      "statement_id": "1",
      "substatement_id": "1",
      "class": "READ",
      "command": "SELECT FOR KEY SHARE",
      "statement": "SELECT pg_current_wal_lsn()",
      "parameter": "<none>"
    }
  },
  "logging_pod": "cluster-example-1"
}
```


PGAudit documentation を参照してください。

レコードの各フィールドの詳細については、

5.23.5 EDB Audit log

EDB Postgres Advanced Server で実行されているクラスターは、次のように EDB Audit を有効にできます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3
  imageName: docker.enterprisedb.com/k8s/edb-postgres-advanced:18-standard-ubi9

  postgresql:
    epas:
      audit: true

  storage:
    size: 1Gi
```

`.spec.postgresql.epas.audit: true` を設定すると、次のパラメーターが適用されます。

```
edb_audit = csv
edb_audit_destination = file
edb_audit_directory = /controller/log
edb_audit_filename = edb_audit
edb_audit_rotation_day = none
edb_audit_rotation_seconds = 0
edb_audit_rotation_size = 0
```

(次のページに続く)

(前のページからの続き)

```
edb_audit_tag =
edb_log_every_bulk_value = false
```

他のパラメーターは、通常のように.spec.postgresql.parameters を介して渡すことができます。

監査 CSV ログは、残りのすべてのログと同様に、解析され、JSON 形式で stdout にルーティングされます。

- .logger を edb_audit に設定
- .msg を record に設定
- 解析されたレコード全体を JSON オブジェクトとして含む.record

以下の例を参照してください。


```
{
  "level": "info",
  "ts": 1624629110.7641866,
  "logger": "edb_audit",
  "msg": "record",
  "record": {
    "log_time": "2021-06-25 13:51:50.763 UTC",
    "user_name": "postgres",
    "database_name": "postgres",
    "process_id": "68",
    "connection_from": "[local]",
    "session_id": "60d5df76.44",
    "session_line_num": "5",
    "process_status": "idle in transaction",
    "session_start_time": "2021-06-25 13:51:50 UTC",
    "virtual_transaction_id": "3/93",
    "transaction_id": "1183",
    "error_severity": "AUDIT",
    "sql_state_code": "00000",
    "message": "statement: GRANT EXECUTE ON function pg_catalog.pg_read_binary_
↪file(text) TO \"streaming_replica\"",
    "detail": "",
    "hint": "",
    "internal_query": "",
    "internal_query_pos": "",
    "context": "",
    "query": "",
    "query_pos": "",
    "location": "",
    "application_name": "",
    "backend_type": "client backend",
    "command_tag": "GRANT",
```


(次のページに続く)

(前のページからの続き)

```
    "audit_tag": "",
    "type": "grant"
  },
  "logging_pod": "cluster-example-1"
}
```

EDB Audit file を参照してください

レコードのフィールドの詳細については、

5.23.6 その他のログ

オペレーターとそのインスタンスによって生成されたすべてのログは JSON 形式であり、`logger` フィールドは、それらを生成したプロセスを示します。使用可能な `logger` 値は次のとおりです。

- `barman-cloud-wal-archive` `barman-cloud-wal-archive` からのログ
- `barman-cloud-wal-restore` `barman-cloud-wal-restore` からのログ
- `edb_audit` EDB Audit 拡張機能から
- `initdb` `initdb` の実行からのログ
- `pg_basebackup` `pg_basebackup` の実行からのログ
- `pg_controldata` `pg_controldata` の実行からのログ
- `pg_ctl` `pg_ctl` サブコマンドの実行からのログ
- `pg_rewind` `pg_rewind` の実行からのログ
- `pgaudit` PGAudit 拡張機能からのログ
- `postgres` `postgres` インスタンスからのログ `msg` は `record` とは異なります
- `wal-archive` インスタンスマネージャーの `wal-archive` サブコマンドからのログ
- `wal-restore` インスタンスマネージャーの `wal-restore` サブコマンドからのログ
- `instance-manager` *PostgreSQL instance manager* から

特定の構造に従う `postgres` および `edb_audit` を除き、他のすべての `logger` 値には、ログに記録されるエスケープメッセージを含む `msg` フィールドが含まれます。

5.24 Certificates

CloudNativePG クラスターの EDB Postgres AI は、TLS 証明書をネイティブにサポートするように設計されました。クラスターをセットアップするには、オペレーターに以下が必要です。

- サーバー認証局 CA 証明書
- サーバー CA によって署名されたサーバー TLS 証明書
- クライアント CA 証明書

- クライアント CA によって生成されたストリーミングレプリケーションクライアント証明書

> **注釈**
>
> クラスターが使用するすべてのシークレットとその有効期限は、クラスターのステータスで見つけることができます。

CloudNativePG クラスターの EDB Postgres AI は、TLS 証明書に関して非常に柔軟です。主に 2 つのモードで動作します。

1. Operator-Managed Mode –証明書は完全に自動化された方法でオペレーターによって内部管理され、CloudNativePG クラスター用に EDB Postgres AI によって作成された CA を使用して署名されます。
2. **ユーザー提供の証明書モード** – 証明書はオペレーターの外部で生成され、シークレットとしてクラスター定義にインポートされます。CloudNativePG クラスターの EDB Postgres AI は cert-manager と自分自身を統合します

Cert-manager **の例** を参照してください。

証明書の一部のみが CNP 外部で生成されるハイブリッドアプローチを選択することもできます。

> **注釈**
>
> オペレーターとインスタンスは、DNS 名を無視して、CA のみに対してサーバー証明書を検証します。このアプローチは、クラスター内の通信に使用される `<cluster>-rw` サービスのユーザー指定証明書に DNS 名が一般的に欠如しているためです。

> **注釈**
>
> 上記にリストされている CA 管理証明書を超えて、オペレーターはメモリ内に生成された自己署名クライアント証明書を使用して、インスタンスマネージャーのステータスポートに対して認証します。この証明書はクライアント CA によって署名されていません。インスタンスマネージャーは、公開キーフィンガープリントを固定することによりそれを信頼します。 **オペレーターからインスタンスへの認証** を参照してください。

デフォルトでは、オペレーターは単一の認証局 CA を自動的に生成して、クライアントとサーバーの証明書の両方を発行します。これらの証明書は事業者によって継続的に管理され、有効期限の 7 日前に自動更新されます 90 日の有効期間内。

> **注釈**
>
> *CERTIFICATE_DURATION* および `EXPIRING_CHECK_THRESHOLD` 環境変数を構成することにより、このデフォルトの動作を調整できます。詳細なガイダンスについては、 *Operator configuration* を参照してください。

証明書を生成するとき、オペレーターは、Kubernetes クラスターの DNS ゾーンがデフォルトで `cluster.`

local に設定されていることを想定しています。この動作は、KUBERNETES_CLUSTER_DOMAIN 環境変数を設定することによりカスタマイズできます。便利な別の方法は、*operator's configuration capability* を使用することです。

5.24.1 サーバー証明書

5.24.2 サーバー CA シークレット

オペレーターは、自己署名 CA を生成し、次のキーを含む汎用シークレットに保存します。

- ca.crt – サーバー証明書の検証に使用される CA 証明書。クライアントの接続文字列で sslrootcert として使用されます。
- ca.key – サーバー SSL 証明書に自動的に署名するために使用されるキー。

5.24.3 サーバー TLS シークレット

オペレーターは、生成された自己署名 CA を使用して、サーバー TLS 証明書に署名します。これは kubernetes.io/tls タイプのシークレットに保存され、インスタンスによって ssl_cert_file および ssl_key_file として使用されるように構成されます。このアプローチにより、クライアントは自分の ID を確認し、安全に接続できます。

5.24.4 サーバー代替 DNS 名

デフォルトのものに加えて、生成されたサーバー TLS シークレットの一部として DNS サーバーの代替名を指定できます。

5.24.5 クライアント証明書

5.24.6 クライアント CA シークレット

デフォルトでは、サーバー CA と同じ自己署名 CA が使用されます。パブリック部分は ssl_ca_file としてすべてのインスタンスに渡されるため、署名したクライアント証明書を検証できます。プライベートキーは同じシークレットに保存され、kubectl cnp プラグインによって生成されたクライアント証明書の署名に使用されます。

5.24.7 クライアント streaming_replica 証明書

オペレーターは、生成された自己署名 CA を使用して、ユーザー streaming_replica のクライアント証明書に署名し、kubernetes.io/tls タイプのシークレットに保存します。プライマリインスタンスへの安全な接続を許可するために、この証明書はレプリカの接続文字列で sslcert および sslkey として渡されます。

ユーザー提供の証明書モード

5.24.8 サーバー証明書

必要に応じて、 2 つのサーバー証明書を提供し、cert-manager などの別のコンポーネントを使用して生成することもできます。クラスターにカスタムサーバー TLS 証明書を使用するには、次のパラメーターを指定する必要があります。

- `serverTLSSecret` – サーバー TLS 証明書を含む `kubernetes.io/tls` タイプのシークレットの名前。これには、標準の `tls.crt` および `tls.key` キーの両方が含まれている必要があります。
- `serverCASecret` – `ca.crt` キーを含むシークレットの名前。

> **注釈**
>
> オペレーターは、クライアント証明書に関連する 2 つのシークレットを作成および管理します。

> **注釈**
>
> オペレーターとインスタンスは、DNS 名を無視して、CA のみに対してサーバー証明書を検証します。このアプローチは、クラスター内の通信に使用される `<cluster>-rw` サービスのユーザー指定証明書に DNS 名が一般的に欠如しているためです。

> **注釈**
>
> ConfigMap とシークレットをインスタンスによってリロードする場合は、キー *k8s.enterprisedb.io/reload* のラベルを追加できます。それ以外の場合は、 *kubectl cnp reload* サブコマンドを使用してインスタンスをリロードする必要があります。

5.24.9 例

次のファイルがあるとします。

- `server-ca.crt` – サーバー TLS 証明書に署名した CA の証明書。
- `server.crt` – サーバー TLS 証明書の証明書。
- `server.key` –サーバー TLS 証明書のプライベートキー。

CA 証明書を含むシークレットを作成します。

```
kubectl create secret generic my-postgresql-server-ca \
  --from-file=ca.crt=./server-ca.crt
```

TLS 証明書を使用してシークレットを作成します。

```
kubectl create secret tls my-postgresql-server \
  --cert=./server.crt --key=./server.key
```

これらのシークレットを参照する PostgreSQL クラスターを作成します。

```
kubectl apply -f - <<EOF
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3
  certificates:
    serverCASecret: my-postgresql-server-ca
    serverTLSSecret: my-postgresql-server
  storage:
    storageClass: standard
    size: 1Gi
EOF
```

新しいクラスターは、TLS 接続に提供されたサーバー証明書を使用します。

5.24.10 Cert-manager の例

この簡単な例は、 cert-manager の使用方法を示しています

自己署名 CA をセットアップし、必要な TLS サーバー証明書を生成するには。

```
- --
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: selfsigned-issuer
spec:
  selfSigned: {}
- --
apiVersion: v1
kind: Secret
metadata:
  name: my-postgres-server-cert
  labels:
    k8s.enterprisedb.io/reload: ""
- --
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: my-postgres-server-cert
spec:
  secretName: my-postgres-server-cert
  usages:
```

(次のページに続く)

(前のページからの続き)

```
    - server auth
  dnsNames:
    - cluster-example-lb.internal.mydomain.net
    - cluster-example-rw
    - cluster-example-rw.default
    - cluster-example-rw.default.svc
    - cluster-example-r
    - cluster-example-r.default
    - cluster-example-r.default.svc
    - cluster-example-ro
    - cluster-example-ro.default
    - cluster-example-ro.default.svc
  issuerRef:
    name: selfsigned-issuer
    kind: Issuer
    group: cert-manager.io
```

Cert-manager は、`my-postgres-server-cert` という名前のシークレットを作成します。これには、必要なすべてのファイルが含まれており、次のようにクラスターから参照できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3
  certificates:
    serverTLSSecret: my-postgres-server-cert
    serverCASecret: my-postgres-server-cert
  storage:
    size: 1Gi
```

cert-manager を使用して、サーバーとクライアントの両方の CA と証明書を管理する完全な例は、cluster-example-cert-manager.yaml

展開マニフェスト。

5.24.11 クライアント証明書

必要に応じて、cert-manager や HashiCorp vault などの別のコンポーネントを使用して生成し、2 つのクライアント証明書を提供することもできます。カスタム CA を使用してクラスターのクライアント証明書を検証するには、次のパラメーターを指定する必要があります。

- `replicationTLSSecret` – ユーザー `streaming_replica` のクライアント証明書を含む `kubernetes.io/tls` タイプのシークレットの名前 `streaming_replica`。これには、標準の `tls.crt` および `tls.key` キーの両方が含まれている必要があります。

- clientCASecret – クライアント証明書の検証に使用する CA の ca.crt キーを含むシークレットの名前。

> **注釈**
>
> オペレーターは、サーバー証明書に関連する 2 つのシークレットを作成および管理します。

> **注釈**
>
> クラスターはクライアント CA 秘密キーを制御していないため、 *kubectl cnp certificate* を使用してクライアント証明書を生成できません。

> **注釈**
>
> ConfigMap とシークレットをインスタンスによって自動的にリロードする場合は、キー *k8s.enterprisedb.io/reload* のラベルを追加できます。それ以外の場合、 *kubectl cnp reload* サブコマンドを使用してインスタンスをリロードする必要があります。

5.24.12 streaming_replica クライアント証明書のカスタマイズ

一部の環境では、会社のポリシーまたは複数のクラスターで共有される CA などのセキュリティ上の問題のため、共通名 streaming_replica の証明書を生成できない場合があります。このような場合、ユーザーマッピング機能を使用して、異なる一般名を含む証明書を使用して streaming_replica ユーザーとして認証を許可できます。

この設定を構成するには、 cnp_streaming_replica という名前の事前定義マップの pg_ident.conf エントリーを追加します。

たとえば、共通名 streaming-replica.cnp.svc.cluster.local の証明書を使用して streaming_replica 認証を有効にするには、次をクラスター定義に追加します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  postgresql:
    pg_ident:
      - cnp_streaming_replica streaming-replica.cnp.svc.cluster.local streaming_
↪replica
```

オペレーターによる pg_ident.conf の管理方法の詳細については、ドキュメントの *PostgreSQL Configuration* を参照してください。

5.24.13 Cert-manager の例

この簡単な例は、 cert-manager の使用方法を示しています

自己署名 CA をセットアップし、必要な TLS サーバー証明書を生成するには。

```
- --
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: selfsigned-issuer
spec:
  selfSigned: {}
- --
apiVersion: v1
kind: Secret
metadata:
  name: my-postgres-client-cert
  labels:
    k8s.enterprisedb.io/reload: ""
- --
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: my-postgres-client-cert
spec:
  secretName: my-postgres-client-cert
  usages:
    - client auth
  commonName: streaming_replica
  issuerRef:
    name: selfsigned-issuer
    kind: Issuer
    group: cert-manager.io
```

Cert-manager は、必要なすべてのファイルを含む `my-postgres-client-cert` という名前のシークレットを作成します。次のようにクラスターから参照できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3
  certificates:
    clientCASecret: my-postgres-client-cert
    replicationTLSSecret: my-postgres-client-cert
```

(次のページに続く)

(前のページからの続き)

```
  storage:
    size: 1Gi
```

cert-manager を使用して、サーバーとクライアントの両方の CA と証明書を管理する完全な例は、cluster-example-cert-manager.yaml

展開マニフェスト。

5.24.14 cluster-example-cert-manager.yaml

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: selfsigned-issuer
spec:
  selfSigned: {}
- --
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: server-ca
spec:
  isCA: true
  commonName: my-selfsigned-server-ca
  secretName: server-ca-key-pair
  privateKey:
    algorithm: ECDSA
    size: 256
  issuerRef:
    name: selfsigned-issuer
    kind: Issuer
    group: cert-manager.io
- --
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: server-ca-issuer
spec:
  ca:
    secretName: server-ca-key-pair
- --
apiVersion: v1
kind: Secret
metadata:
  name: my-postgres-server-cert
```

(次のページに続く)

(前のページからの続き)

```
  labels:
    k8s.enterprisedb.io/reload: ""
- --
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: my-postgres-server-cert
spec:
  secretName: my-postgres-server-cert
  usages:
    - server auth
  dnsNames:
    - cluster-example-lb.internal.mydomain.net
    - cluster-example-rw
    - cluster-example-rw.default
    - cluster-example-rw.default.svc
    - cluster-example-r
    - cluster-example-r.default
    - cluster-example-r.default.svc
    - cluster-example-ro
    - cluster-example-ro.default
    - cluster-example-ro.default.svc
  issuerRef:
    name: server-ca-issuer
    kind: Issuer
    group: cert-manager.io
- --
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: client-ca
spec:
  isCA: true
  commonName: my-selfsigned-client-ca
  secretName: client-ca-key-pair
  privateKey:
    algorithm: ECDSA
    size: 256
  issuerRef:
    name: selfsigned-issuer
    kind: Issuer
    group: cert-manager.io
- --
apiVersion: cert-manager.io/v1
kind: Issuer
```

(次のページに続く)

(前のページからの続き)

```
metadata:
  name: client-ca-issuer
spec:
  ca:
    secretName: client-ca-key-pair
- --
apiVersion: v1
kind: Secret
metadata:
  name: my-postgres-client-cert
  labels:
    k8s.enterprisedb.io/reload: ""
- --
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: my-postgres-client-cert
spec:
  secretName: my-postgres-client-cert
  usages:
    - client auth
  commonName: streaming_replica
  issuerRef:
    name: client-ca-issuer
    kind: Issuer
    group: cert-manager.io
- --
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3
  certificates:
    serverTLSSecret: my-postgres-server-cert
    serverCASecret: my-postgres-server-cert
    clientCASecret: my-postgres-client-cert
    replicationTLSSecret: my-postgres-client-cert
  storage:
    size: 1Gi
```

5.25 Client TLS/SSL connections

CloudNativePG クラスターの EDB Postgres AI が TLS 証明書をサポートする方法の詳細については、こちらをご覧ください。

CloudNativePG クラスターオペレーターの EDB Postgres AI は、転送中の暗号化とサーバー側とクライアント側での認証の両方で TLS / SSL と連携するように設計されました。CNP オペレーターを使用して作成されたクラスターには、TLS クライアント証明書を作成および署名するための認証局 CA が付属しています。kubectl の cnp プラグインを使用して、パスワードを使用する代わりにユーザーを認証するための新しい TLS クライアント証明書を発行できます。

TLS / SSL 証明書を使用して認証するためのこれらの手順は、cluster-example-pg-hba.yaml マニフェストを使用してクラスターをインストールしていることを前提としています。構成より規約パラダイムによると、そのファイルは、app と呼ばれるユーザーが所有する app データベースを作成します。bootstrap セクションの initdb 構成を介して、この規則を変更できます。

5.25.1 新しい証明書の発行

kubectl のプラグインの使用方法の詳細については、コンテンツをご覧ください。

次のようにして、cluster-example PostgreSQL クラスターにアプリユーザーの証明書を作成できます。

```
kubectl cnp certificate cluster-app \
  --cnp-cluster cluster-example \
  --cnp-user app
```

これで、証明書を検証できるようになりました。

```
kubectl get secret cluster-app \
  -o jsonpath="{.data[tls\.crt]}" \
  | base64 -d | openssl x509 -text -noout \
  | head -n 11
```

出力

```
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      5d:e1:72:8a:39:9f:ce:51:19:9d:21:ff:1e:4b:24:5d
    Signature Algorithm: ecdsa-with-SHA256
    Issuer: OU = default, CN = cluster-example
    Validity
      Not Before: Mar 22 10:22:14 2021 GMT
      Not After : Mar 22 10:22:14 2022 GMT
    Subject: CN = app
```

ご覧のとおり、デフォルトの TLS クライアント証明書は、90 日間の有効期間と、PostgreSQL のユーザー名に対応する単純な CN で作成されます。EXPIRE_CHECK_THRESHOLD および CERTIFICATE_DURATION パ

ラメーターを使用して、有効性としきい値を指定できます。これは、pg_hba.conf の hostssl エントリの cert 認証方法を活用するために必要です。

5.25.2 TLS 証明書を介した接続のテスト

次に、CloudNativePG Cluster クラスターの EDB Postgres AI に接続するデモクライアントアプリケーションを構成することにより、このクライアント証明書をテストします。

cert-test.yaml と呼ばれる次のマニフェストは、データベースクラスターが実行されているのと同じ名前空間にテストアプリケーションを使用してデモポッドを作成します。

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cert-test
spec:
  replicas: 1
  selector:
    matchLabels:
      app: webtest
  template:
    metadata:
      labels:
        app: webtest
    spec:
      containers:
        - image: ghcr.io/cloudnative-pg/webtest:1.7.0
          name: cert-test
          volumeMounts:
            - name: secret-volume-root-ca
              mountPath: /etc/secrets/ca
            - name: secret-volume-app
              mountPath: /etc/secrets/app
          ports:
            - containerPort: 8080
          env:
            - name: DATABASE_URL
              value: >
                sslkey=/etc/secrets/app/tls.key
                sslcert=/etc/secrets/app/tls.crt
                sslrootcert=/etc/secrets/ca/ca.crt
                host=cluster-example-rw.default.svc
                dbname=app
                user=app
                sslmode=verify-full
            - name: SQL_QUERY
              value: SELECT 1
```

(次のページに続く)

(前のページからの続き)

```
        readinessProbe:
          httpGet:
            port: 8080
            path: /tx
    volumes:
      - name: secret-volume-root-ca
        secret:
          secretName: cluster-example-ca
          defaultMode: 0600
      - name: secret-volume-app
        secret:
          secretName: cluster-app
          defaultMode: 0600
```

このポッドは、CloudNativePG Cluster オペレーターの EDB Postgres AI によって管理されるシークレットをマウントします。

- `sslcert` – TLS クライアントのパブリック証明書。
- `sslkey` – TLS クライアント証明書のプライベートキー。
- `sslrootcert` – インスタンスの ID の確認に使用するサーバー上の証明書に署名した TLS CA 証明書。

これらは、psql および pgbench のような他の libpq ベースのアプリケーションが Postgres データベースへの TLS 暗号化された接続を確立するために必要なデフォルトリソースを作成するために使用されます。

デフォルトでは、psql は現在のユーザーの`~/.postgresql` ディレクトリで証明書を検索しますが、`sslkey` 、`sslcert` 、および `sslrootcert` オプションを使用して、libpq を暗号化マテリアルの実際の場所を指すことができます。これらのファイルのコンテンツは、kubectl の cnp プラグインを使用して以前に作成したシークレットから収集されます。

アプリケーションを展開します。

```
kubectl create -f cert-test.yaml
```

次に、作成したポッドを PostgreSQL クライアントとして使用して、作成した TLS 証明書を使用して SSL 接続と認証を検証します。

準備状況プローブは、データベースサーバーに到達できるときにアプリケーションの準備ができていることを確認するように構成されました。

接続が機能していることを確認できます。これを行うには、ポッドのコンテナ内でインタラクティブな `bash` を実行し、必要なオプションを使用して psql を実行します。PostgreSQL サーバーは、読み取り/書き込みの Kubernetes サービスを介して公開されます。psql コマンドを指定してこのサービスに接続します。

```
kubectl exec -it cert-test -- bash -c "psql
sslkey=/etc/secrets/app/tls.key sslcert=/etc/secrets/app/tls.crt
sslrootcert=/etc/secrets/ca/ca.crt host=cluster-example-rw.default.svc dbname=app
user=app sslmode=verify-full -c select version();"
```

出力

```
                                        version
- ------------------------------------------------------------------------------------
↪-
- -----------------
PostgreSQL 18.4 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 8.3.1 20191121 (Red Hat
8. 3.1-5), 64-bit
(1 row)
```

5.25.3 TLS プロトコルバージョンについて

デフォルトでは、オペレーターは両方の ssl_min_protocol_version を設定します

および ssl_max_protocol_version

TLSv1.3 に。

これは、PostgreSQL オペランドイメージに、TLSv1.3 バージョンをサポートする OpenSSL ライブラリが含まれていることを前提としています。そうでない場合、またはクライアントアプリケーションがより低いバージョン番号を必要とする場合、他の Postgres GUC と同様に、PostgreSQL 構成で手動で構成する必要があります。

cluster-example-pg-hba.yaml

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3
  postgresql:
    pg_hba:
      - hostssl app all all cert

  storage:
    size: 1Gi
```

5.26 Connecting from an application

アプリケーションは、同じ Kubernetes クラスター内の CloudNativePG クラスター用に EDB Postgres AI によって作成されたサービスで動作することになっています。

サービスとその管理方法の詳細については、*Service management* セクションを参照してください。

これらのサービスは、次の方法でアプリケーションで使用できます。

- DNS 解決

- 環境変数

PostgreSQL に接続するための資格情報には、オペレーターによって生成されたシークレットを使用できます。

5.26.1 DNS 解決

Kubernetes DNS サービスを使用して、特定のサーバーをポイントできます。オペレーターには Kubernetes DNS サービスが必要です。アプリケーションが PostgreSQL クラスターと同じ名前空間に展開されている場合、サービスの名前を使用してこれを行うことができます。PostgreSQL クラスターが別の名前空間にある場合、完全な修飾子 `service-name.namespace-name` を使用できます。

DNS が優先および推奨される検出方法です。

5.26.2 環境変数

PostgreSQL クラスターが含まれる同じ名前空間にアプリケーションを展開する場合、環境変数を使用してデータベースに接続することもできます。

たとえば、PostgreSQL クラスターの名前が `pg-database` であるとすると、アプリケーションで次の環境変数を使用できます。

- `PG_DATABASE_R_SERVICE_HOST` 読み取り専用ワークロードのすべての PostgreSQL インスタンスを指すサービスの IP アドレス
- `PG_DATABASE_RO_SERVICE_HOST` クラスターのすべてのホットスタンバイレプリカをポイントするサービスの IP アドレス
- `PG_DATABASE_RW_SERVICE_HOST` クラスターの_primary_インスタンスを指すサービスの IP アドレス

5.26.3 シークレット

PostgreSQL オペレーターは、展開する PostgreSQL クラスターごとに最大 2 つの `basic-auth` タイプシークレットを生成します。

- `[cluster name]-app` `.spec.bootstrap.initdb.secret.name` を介して既存のシークレットを提供していない場合
- `[cluster name]-superuser` `.spec.enableSuperuserAccess` が `true` に設定されており、`.spec.superuserSecret` を使用して別のシークレットを指定していない場合

各シークレットには次のものが含まれます。

- ユーザー名
- パスワード
- RW サービスへのホスト名
- ポート番号
- データベース名
- 作業用 .pgpass file

- uri
- jdbc-uri
- fqdn-uri
- fqdn-jdbc-uri

URI で使用される FQDN は、`KUBERNETES_CLUSTER_DOMAIN` 構成パラメーターで指定された Kubernetes クラスタードメインを使用して計算されます。詳細については、*the operator configuration documentation* を参照してください。

-app 資格情報は、PostgreSQL クラスターに接続するアプリケーションが使用するものであり、データベースを_owning_するユーザーに対応します。

-superuser のものは、管理目的でのみ使用されることになっており、`postgres` ユーザーに対応します。

5.27 Connection Pooling

CloudNativePG クラスターの EDB Postgres AI は、`Pooler` カスタムリソース定義 CRD を介して、PostgreSQL で最も人気のあるオープンソース接続プーラーの 1 つである PgBouncer との接続プーリングのネイティブサポートを提供します。

簡単に言うと、CloudNativePG クラスターの EDB Postgres AI のプーラーは、アプリケーションと PostgreSQL サービスたとえば `rw` サービスの間にある PgBouncer ポッドの展開です。別個の、スケーラブル、構成可能、高可用性のデータベースアクセスレイヤーを作成します。

> **警告**
>
> CloudNativePG クラスターの EDB Postgres AI には、PgBouncer の `auth_dbname` 機能が必要です。必ず PgBouncer コンテナイメージバージョン**1.19 以降**を使用してください。

5.27.1 アーキテクチャ

次の図は、PgBouncer に基づくデータベースアクセスレイヤーの導入が、CloudNativePG クラスターの EDB Postgres AI のアーキテクチャをどのように変更するかを強調しています。PostgreSQL プライマリサービスに直接接続する代わりに、アプリケーションは PgBouncer の同等のサービスに接続できます。この機能により、既存の接続を再利用できるため、PostgreSQL 側でのパフォーマンスとリソース管理が向上します。

5.27.2 クイックスタート

この例は、CloudNativePG クラスターの EDB Postgres AI が PgBouncer プーラーを実装する方法を示すのに役立ちます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Pooler
metadata:
  name: pooler-example-rw
```

(次のページに続く)

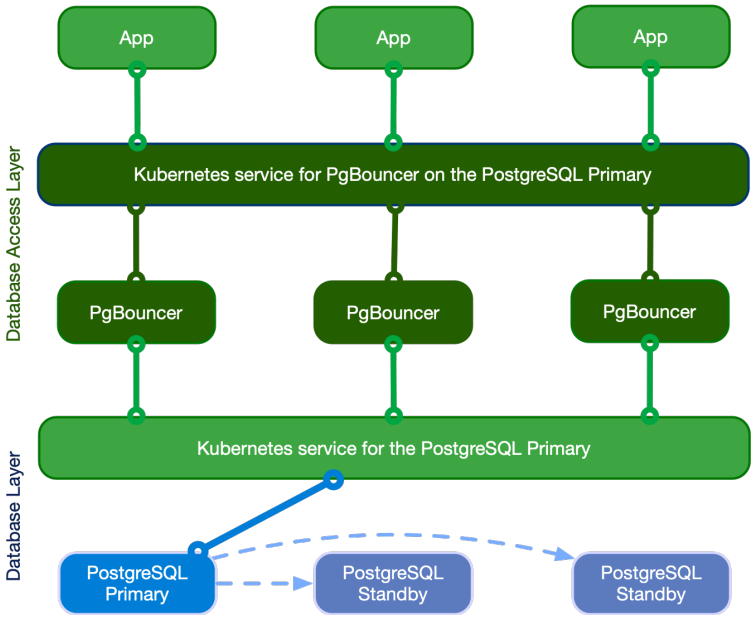


図 3: Applications writing to the single primary via PgBouncer

(前のページからの続き)

```
spec:
  cluster:
    name: cluster-example

  instances: 3
  type: rw
  pgbouncer:
    poolMode: session
    parameters:
      max_client_conn: "1000"
      default_pool_size: "10"
```

警告

spec.cluster フィールドは、作成後は不変です。プーラーを別の `Cluster` でポイントするには、既存のリソースを更新する代わりに、新しい `Pooler` リソースを作成します。

`Pooler` リソースは、Postgres クラスターと同じ名前空間に存在する必要があります。これは、session で構成された latest stable image of PgBouncer を実行する 3 つのポッドの Kubernetes デプロイメントで構成されています

それぞれ最大 1000 の接続を受け入れます。デフォルトのプールサイズは、PostgreSQL に対する 10 ユーザー/データベースペアです。

CloudNativePG クラスターの EDB Postgres AI は、PgBouncer で使用される構成ファイルを含むプーラーと同じ名前のシークレットも作成します。

API リファレンスで。

5.27.3 リソースのライフサイクルのプール化

`Pooler` リソースは、オペレーターによって自動的に管理されません。必要に応じて手動で作成し、同じ PostgreSQL クラスターに複数のプーラーを展開できます。

理解する重要なポイントは、`Cluster` および `Pooler` リソースのライフサイクルが独立していることです。クラスターを削除してもプーラーは自動的に削除されず、プーラーを削除してもクラスターには影響しません。

> **注釈**
>
> プーラーの動作を理解すると、アーキテクチャを完全に柔軟に設計できます。プーラーなしのクラスター、単一のプーラーのあるクラスター、または複数のプーラーたとえば、アプリケーションごとに 1 つを持つクラスターを実行できます。

5.27.4 セキュリティ

PgBouncer プーラーは、プールのクライアントアプリケーションとサーバー PostgreSQL の両方で、TLS 接続を介した転送中の暗号化の CloudNativePG クラスターサポートのために EDB Postgres AI と透過的に統合されています。

コンテナは `pgbouncer` システムユーザーとして実行され、 `pgbouncer` 管理データベースへのアクセスは、ピア認証を介したローカル接続を介してのみ許可されます。

証明書

デフォルトでは、PgBouncer プーラーは PostgreSQL クラスターと同じ証明書を再利用します。TLS クライアント証明書認証に依存して、PostgreSQL サーバーに接続し、クライアントのパスワード認証に使用される `auth_query` を実行します **認証を使用する** を参照してください。

独自のシークレットを指定すると、ビルトイン統合が無効になります。その時点から、認証を管理するための完全な制御と責任を取得します。サポートされているシークレット形式は次のとおりです。

1. ベーシック認証
2. TLS
3. 不透明

不透明なシークレットの場合、 `Pooler` リソースは次のキーを想定しています。

- `tls.crt`
- `tls.key`

実際、これは、不透明なシークレットを TLS シークレットとして、同じ構造から開始できることを意味します。

5.27.5 認証

デフォルトの認証方法

デフォルトでは、CloudNativePG クラスターの EDB Postgres AI は、PostgreSQL データベースに接続する PgBouncer クライアントのパスワードベースの認証をネイティブにサポートしています。

このビルトインメカニズムは、PgBouncer の auth_dbname バージョン 1.19 で導入、auth_user および auth_query オプションとともに利用します。

ビルトイン統合は、次のタスクを実行します。

- PostgreSQL サーバーに cnp_pooler_pgbouncer と呼ばれる専用ユーザーを作成します
- postgres データベースにルックアップファンクションを作成し、実行特権を cnp_pooler_pgbouncer に付与します PoLA 原則に従って
- このユーザーの TLS 証明書を発行します
- auth_user として cnp_pooler_pgbouncer を使用し、auth_dbname として postgres を使用するように PgBouncer を構成します
- 発行された TLS 証明書を使用して PostgreSQL に対して cnp_pooler_pgbouncer を認証するように PgBouncer を構成します
- クラスターに関連付けられているプーラーがない場合、上記のすべてを自動的にクリーンアップします

SQL 命令

ビルトイン統合の一部として、CloudNativePG クラスターの EDB Postgres AI は、リコンシリエーション中に一連の SQL ステートメントを自動的に実行します。これらのステートメントは、インスタンスマネージャーによって postgres データベースに対して postgres ユーザーを使用して実行されます。

ロールの作成

```
CREATE ROLE cnp_pooler_pgbouncer WITH LOGIN;
```

postgres データベースへのアクセスを付与します。

```
GRANT CONNECT ON DATABASE postgres TO cnp_pooler_pgbouncer;
```

パスワード検証のためのルックアップファンクションを作成します。このファンクションは、SECURITY DEFINER 特権で postgres データベースに作成され、PgBouncer の auth_query オプションで使用されます。ファンクション所有者として実行されるため、その search_path は pg_catalog, pg_temp に固定され、ファンクション本体が呼び出し元またはテナント制御の search_path を介して演算子またはオブジェクトを解決できません。

```
CREATE OR REPLACE FUNCTION public.user_search(uname TEXT)
  RETURNS TABLE (usename name, passwd text)
  LANGUAGE sql SECURITY DEFINER
```

(次のページに続く)

(前のページからの続き)

```
  SET search_path = pg_catalog, pg_temp AS
  SELECT usename, passwd FROM pg_catalog.pg_shadow WHERE usename=$1;;
```

> **注釈**
>
> CloudNativePG クラスターの EDB Postgres AI の以前のバージョンで作成されたクラスターは、固定された `search_path` なしの `user_search` ファンクションを実行します。オペレーターは、クラスターがアップグレードされたときのリコンシリエーション中に、 *SET search_path* 句を使用してファンクションを自動的に再作成します。

```
REVOKE ALL ON FUNCTION public.user_search(text)
  FROM public;

GRANT EXECUTE ON FUNCTION public.user_search(text)
  TO cnp_pooler_pgbouncer;
```

カスタム認証方法

独自の証明書シークレットを提供すると、ビルトイン統合が無効になります。

これにより、認証プロセスを自分自身で管理する柔軟性と責任が得られます。上記の手順に従って、デフォルトの設定と同様の動作を複製できます。

5.27.6 ポッドテンプレート

`Pooler` リソースを使用すると、 `template` セクションを介して基になるポッドをカスタマイズできます。これにより、スケジュール制約、カスタムセキュリティコンテキスト、リソースオーバーライドなどの高度な構成のための Kubernetes `PodSpec` へのフルアクセスが提供されます。

サポートされているフィールドの完全なリストについては、 *PoolerSpec* API リファレンスを参照してください。

主要な要件

- **``pgbouncer`` コンテナ名**コンテナ設定画像やリソースなどをオーバーライドする場合、コンテナの名前は `pgbouncer` に設定する必要があります**。オペレーターは、この特定の名前を探して PgBouncer プロセスを管理します。
- **必須 ``containers`` フィールド** `template` は標準の Kubernetes `PodSpec` スキーマに従っているため、`containers` フィールドは必須です。
- コンテナレベルの設定を変更しない場合は、空の配列 `containers: []` に設定する必要があります。
- `containers` フィールドが見つからない場合、API サーバーは `ValidationError` をスローします。

例

ポッドアンチアフィニティを使用した高可用性

この構成では、 podAntiAffinity を使用して PgBouncer ポッドがさまざまなノードに分散されるようにし、単一のノードの障害がプール全体をダウンさせるのを防ぎます。

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Pooler
metadata:
  name: pooler-example-rw
spec:
  cluster:
    name: cluster-example
  instances: 3
  type: rw

  template:
    metadata:
      labels:
        app: pooler
    spec:
      containers: []
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            - labelSelector:
                matchExpressions:
                  - key: app
                    operator: In
                    values:
                      - pooler
              topologyKey: "kubernetes.io/hostname"
```

リソース制限

pgbouncer という名前のコンテナを template セクションに追加することにより、リソース要求と制限を定義できます。

```yaml
#  ...

template:
  metadata:
    # ...
  spec:
    containers:
      # This name MUST be "pgbouncer"
```

(次のページに続く)

(前のページからの続き)

```
    - name: pgbouncer
      resources:
        requests:
          cpu: "0.1"
          memory: 100Mi
        limits:
          cpu: "0.5"
          memory: 500Mi
```

5.27.7 PgBouncer 画像

デフォルトでは、CloudNativePG クラスターの EDB Postgres AI は latest stable PgBouncer image を展開します

演算子はそれに対して構築されました。そのデフォルトは 3 つの方法でオーバーライドできます。複数の場合、ソースはトップダウンで評価されます。以下のリストで最初に一致したものが使用されます。一致するものがない場合、演算子のビルトインデフォルトが適用されます。

1. `spec.template.spec.containers` 内の `pgbouncer` コンテナに設定された明示的なイメージエスケープハッチ-以下の **ポッドテンプレートのオーバーライド** を参照してください。
2. `spec.pgbouncer.image` — `Pooler` に直接設定されたイメージリファレンス。
3. `spec.pgbouncer.imageCatalogRef` — `ImageCatalog` または `ClusterImageCatalog` のエントリーへの参照。

`spec.pgbouncer.image` および `spec.pgbouncer.imageCatalogRef` は相互排他的です。最大 1 つ設定します。

`spec.pgbouncer.image` を使用して、特定の PgBouncer バージョンを固定するか、プライベートレジストリからプルします。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Pooler
metadata:
  name: pooler-example-rw
spec:
  cluster:
    name: cluster-example
  instances: 3
  type: rw
  pgbouncer:
    poolMode: session
    image: docker.enterprisedb.com/k8s/pgbouncer:1.25.1-ubi9
```

イメージカタログを使用する

Pooler リソースは、 Cluster リソースと同じパターンに従って、ImageCatalog または ClusterImageCatalog を介して PgBouncer コンテナイメージを一元管理できますイメージカタログを使用する を参照してください。カタログエントリーは、カタログの componentImages リストで定義された key によって選択されます。

spec.pgbouncer.imageCatalogRef でカタログエントリーを参照します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Pooler
metadata:
  name: pooler-example-rw
spec:
  cluster:
    name: cluster-example
  instances: 3
  type: rw
  pgbouncer:
    poolMode: session
    imageCatalogRef:
      apiGroup: postgresql.k8s.enterprisedb.io
      kind: ImageCatalog
      name: my-catalog
      key: pgbouncer
```

代わりにクラスター全体のカタログを使用するには、kind: ClusterImageCatalog を設定し、対応するリソースで name をポイントします。仕様の残りの部分は同じです。

カタログエントリが更新されると、オペレーターはそれを参照するすべてのプーラーを自動的にリコンサイルし、Pooler 仕様を変更せずに新しいイメージをロールアウトします。

ポッドテンプレートのオーバーライド

ポッドテンプレートは、 pgbouncer コンテナで image を伝えることもできます。その場合、他のすべてのソース spec.pgbouncer.image および spec.pgbouncer.imageCatalogRef を含むをオーバーライドします。これを **エスケープハッチ**として扱います。他のコンテナレベルの設定リソース、環境、セキュリティコンテキストをカスタマイズする必要があり、たまたま同じ場所に画像をピン付けしたい場合にのみ使用します。日常的なイメージ変更の場合、spec.pgbouncer.image またはイメージカタログを優先します。これらのフィールドは検証され、相互排他的で、PgBouncer イメージをゲートするアドミッションポリシーに表示されます。より広範なテンプレートの仕組みについては、ポッドテンプレート を参照してください。

```
#  ...

template:
  spec:
    containers:
```

(次のページに続く)

(前のページからの続き)

```
      - name: pgbouncer
        image: my-pgbouncer:latest
```

解決された画像のモニタリング

オペレーターは、解決された画像を status.image に保存し、結果を status.phase 、 active 、paused 、inactive 、または failed のいずれかに反映します。failed では、 status.phaseReason は原因を説明しますたとえば、カタログまたはキーが存在しない場合。次の方法で現在の状態を検査できます。

```
kubectl get pooler pooler-example-rw -o jsonpath={.status.image}
```

API リファレンスで。

場合によっては、プーラーには別のラベル、アノテーション、または別のサービスタイプが必要になる場合があります。serviceTemplate フィールドを使用して、これを実現できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Pooler
metadata:
  name: pooler-example-rw
spec:
  cluster:
    name: cluster-example
  instances: 3
  type: rw
  serviceTemplate:
    metadata:
      labels:
        app: pooler
    spec:
      type: LoadBalancer
  pgbouncer:
    poolMode: session
    parameters:
      max_client_conn: "1000"
      default_pool_size: "10"
```

デフォルトで、オペレーターは次のデータを使用して ServicePort を追加します。

```
ports:
- name: pgbouncer
  port: 5432
  protocol: TCP
  targetPort: pgbouncer
```

> **警告**
>
> 名前 `pgbouncer` またはポート `5432` で `ServicePort` を指定すると、デフォルトの `ServicePort` が追加されません。これは、同じ `name` または `port` と同じ `ServicePort` エントリが Kubernetes では許可されず、エラーが発生するためです。

5.27.8 高可用性 HA

Kubernetes の展開のため、単一のインスタンスまたは複数のポッドで実行するようにプーラーを構成できます。公開されたサービスにより、クライアントが PgBouncer を実行している利用可能なポッドにランダムに分散されるようになり、基になるサーバー `rw` サービスを使用する場合またはサーバー複数のレプリカで `ro` サービスを使用する場合への接続を管理および再利用します。

> **警告**
>
> インフラストラクチャが複数のアベイラビリティーゾーンにまたがり、それらの間のレイテンシーが高い場合、ネットワークホップに注意してください。たとえば、ゾーン 2 で実行され、ゾーン 3 で実行されている PgBouncer に接続し、ゾーン 1 の PostgreSQL プライマリをポイントしている場合を検討します。

5.27.9 PgBouncer 構成オプション

オペレーターは configuration options for PgBouncer の大部分を管理し、それらのサブセットのみを変更できます。

> **警告**
>
> オペレーターは、これらの設定を検証なしで PgBouncer に直接渡します。構成エラーまたはクラッシュループを防ぐには、各パラメーターが特定の PgBouncer イメージバージョンでサポートされていることを確認します。

- auth_type
- application_name_add_host
- autodb_idle_timeout
- cancel_wait_timeout
- client_idle_timeout
- client_login_timeout
- client_tls_ciphers
- client_tls_sslmode
- client_tls13_ciphers 1.25+

- default_pool_size
- disable_pqexec
- dns_max_ttl
- dns_nxdomain_ttl
- idle_transaction_timeout
- ignore_startup_parameters `extra_float_digits,options` に追加します-CloudNativePG クラスターの EDB Postgres AI が必要
- listen_backlog
- log_connections
- log_disconnections
- log_pooler_errors
- log_stats デフォルトで無効 `0` 、以下の 展開元 hostssl postgres Monitor ${podselector:monitoring} scram-sha-256 セクションで説明するように、Prometheus エクスポートによって統計が既に収集されていることを考えると、
- max_client_conn
- max_db_connections
- max_packet_size
- max_prepared_statements
- max_user_connections
- min_pool_size
- pkt_buf
- query_timeout
- query_wait_timeout
- reserve_pool_size
- reserve_pool_timeout
- sbuf_loopcnt
- server_check_delay
- server_check_query
- server_connect_timeout
- server_fast_close
- server_idle_timeout
- server_lifetime

- server_login_retry
- server_reset_query
- server_reset_query_always
- server_round_robin
- server_tls_ciphers
- server_tls13_ciphers 1.25+
- server_tls_protocols
- server_tls_sslmode
- stats_period
- suspend_timeout
- tcp_defer_accept
- tcp_keepalive
- tcp_keepcnt
- tcp_keepidle
- tcp_keepintvl
- tcp_user_timeout
- tcp_socket_buffer
- track_extra_parameters
- verbose

PgBouncer 構成のカスタマイズは、`.spec.pgbouncer.parameters` マップに宣言的に書き込まれます。

オペレーターはプーラー仕様の変更に対応し、すべての PgBouncer インスタンスは、サービスを中断せずに更新された構成をリロードします。

> **警告**
>
> すべての PgBouncer ポッドは同じ構成で、仕様のパラメーターに合わせて調整されています。これらのパラメーターを間違えると、プーラー全体の操作性を混乱させる可能性があります。演算子はオプションの値を検証しません。

5.27.10 モニタリング

`Pooler` の PgBouncer 実装には、デフォルトの Prometheus エクスポーターが付属しています。次のコマンドを実行して、`cnp_pgbouncer_` 接頭辞を持ついくつかのメトリックを利用できるようにします。

- `SHOW LISTS` 接頭辞 `cnp_pgbouncer_lists`
- `SHOW POOLS` 接頭辞 `cnp_pgbouncer_pools`

- SHOW STATS 接頭辞 cnp_pgbouncer_stats

CloudNativePG クラスターインスタンスの EDB Postgres AI と同様に、エクスポーターは、PgBouncer を実行している各ポッドのポート 9127 で実行され、Go ランタイム接頭辞 go_* に関連するメトリックも提供します。

注釈

PgBouncer を実行しているポッドでエクスポートされたメトリックを検査できます。手順については、**エクスポートされたメトリックを検査する方法** を参照してください。正しい IP と `9127` ポートを使用していることを確認します。

この例は、cnp_pgbouncer メトリックの出力を示しています。

```
#  HELP cnp_pgbouncer_collection_duration_seconds Collection time duration in seconds

#  TYPE cnp_pgbouncer_collection_duration_seconds gauge

cnp_pgbouncer_collection_duration_seconds{collector="Collect.up"} 0.002338805

#  HELP cnp_pgbouncer_collection_errors_total Total errors occurred accessing␣
↪PostgreSQL for metrics.

#  TYPE cnp_pgbouncer_collection_errors_total counter

cnp_pgbouncer_collection_errors_total{collector="sql: Scan error on column index 16,␣
↪name \"load_balance_hosts\": converting NULL to int is unsupported"} 5

#  HELP cnp_pgbouncer_collections_total Total number of times PostgreSQL was accessed␣
↪for metrics.

#  TYPE cnp_pgbouncer_collections_total counter

cnp_pgbouncer_collections_total 5

#  HELP cnp_pgbouncer_last_collection_error 1 if the last collection ended with error,
 0 otherwise.

#  TYPE cnp_pgbouncer_last_collection_error gauge

cnp_pgbouncer_last_collection_error 0

#  HELP cnp_pgbouncer_lists_databases Count of databases.

#  TYPE cnp_pgbouncer_lists_databases gauge
```

(次のページに続く)

(前のページからの続き)

```
cnp_pgbouncer_lists_databases 1

#  HELP cnp_pgbouncer_lists_dns_names Count of DNS names in the cache.

#  TYPE cnp_pgbouncer_lists_dns_names gauge

cnp_pgbouncer_lists_dns_names 0

#  HELP cnp_pgbouncer_lists_dns_pending Not used.

#  TYPE cnp_pgbouncer_lists_dns_pending gauge

cnp_pgbouncer_lists_dns_pending 0

#  HELP cnp_pgbouncer_lists_dns_queries Count of in-flight DNS queries.

#  TYPE cnp_pgbouncer_lists_dns_queries gauge

cnp_pgbouncer_lists_dns_queries 0

#  HELP cnp_pgbouncer_lists_dns_zones Count of DNS zones in the cache.

#  TYPE cnp_pgbouncer_lists_dns_zones gauge

cnp_pgbouncer_lists_dns_zones 0

#  HELP cnp_pgbouncer_lists_free_clients Count of free clients.

#  TYPE cnp_pgbouncer_lists_free_clients gauge

cnp_pgbouncer_lists_free_clients 49

#  HELP cnp_pgbouncer_lists_free_servers Count of free servers.

#  TYPE cnp_pgbouncer_lists_free_servers gauge

cnp_pgbouncer_lists_free_servers 0

#  HELP cnp_pgbouncer_lists_login_clients Count of clients in login state.

#  TYPE cnp_pgbouncer_lists_login_clients gauge

cnp_pgbouncer_lists_login_clients 0

#  HELP cnp_pgbouncer_lists_pools Count of pools.
```

(次のページに続く)

(前のページからの続き)

```
#  TYPE cnp_pgbouncer_lists_pools gauge

cnp_pgbouncer_lists_pools 1

#  HELP cnp_pgbouncer_lists_used_clients Count of used clients.

#  TYPE cnp_pgbouncer_lists_used_clients gauge

cnp_pgbouncer_lists_used_clients 1

#  HELP cnp_pgbouncer_lists_used_servers Count of used servers.

#  TYPE cnp_pgbouncer_lists_used_servers gauge

cnp_pgbouncer_lists_used_servers 0

#  HELP cnp_pgbouncer_lists_users Count of users.

#  TYPE cnp_pgbouncer_lists_users gauge

cnp_pgbouncer_lists_users 2

#  HELP cnp_pgbouncer_pools_cl_active Client connections that are linked to server␣
↪connection and can process queries.

#  TYPE cnp_pgbouncer_pools_cl_active gauge

cnp_pgbouncer_pools_cl_active{database="pgbouncer",user="pgbouncer"} 1

#  HELP cnp_pgbouncer_pools_cl_active_cancel_req Client connections that have␣
↪forwarded query cancellations to the server and are waiting for the server response.

#  TYPE cnp_pgbouncer_pools_cl_active_cancel_req gauge

cnp_pgbouncer_pools_cl_active_cancel_req{database="pgbouncer",user="pgbouncer"} 0

#  HELP cnp_pgbouncer_pools_cl_cancel_req Client connections that have not forwarded␣
↪query cancellations to the server yet.

#  TYPE cnp_pgbouncer_pools_cl_cancel_req gauge

cnp_pgbouncer_pools_cl_cancel_req{database="pgbouncer",user="pgbouncer"} 0

#  HELP cnp_pgbouncer_pools_cl_waiting Client connections that have sent queries but␣
```

(次のページに続く)

(前のページからの続き)

```
↪have not yet got a server connection.

#  TYPE cnp_pgbouncer_pools_cl_waiting gauge

cnp_pgbouncer_pools_cl_waiting{database="pgbouncer",user="pgbouncer"} 0

#  HELP cnp_pgbouncer_pools_cl_waiting_cancel_req Client connections that have not␣
↪forwarded query cancellations to the server yet.

#  TYPE cnp_pgbouncer_pools_cl_waiting_cancel_req gauge

cnp_pgbouncer_pools_cl_waiting_cancel_req{database="pgbouncer",user="pgbouncer"} 0

#  HELP cnp_pgbouncer_pools_load_balance_hosts Number of hosts not load balancing␣
↪between hosts

#  TYPE cnp_pgbouncer_pools_load_balance_hosts gauge

cnp_pgbouncer_pools_load_balance_hosts{database="pgbouncer",user="pgbouncer"} 0

#  HELP cnp_pgbouncer_pools_maxwait How long the first (oldest) client in the queue␣
↪has waited, in seconds. If this starts increasing, then the current pool of servers␣
↪does not handle requests quickly enough. The reason may be either an overloaded␣
↪server or just too small of a pool_size setting.

#  TYPE cnp_pgbouncer_pools_maxwait gauge

cnp_pgbouncer_pools_maxwait{database="pgbouncer",user="pgbouncer"} 0

#  HELP cnp_pgbouncer_pools_maxwait_us Microsecond part of the maximum waiting time.

#  TYPE cnp_pgbouncer_pools_maxwait_us gauge

cnp_pgbouncer_pools_maxwait_us{database="pgbouncer",user="pgbouncer"} 0

#  HELP cnp_pgbouncer_pools_pool_mode The pooling mode in use. 1 for session, 2 for␣
↪transaction, 3 for statement, -1 if unknown

#  TYPE cnp_pgbouncer_pools_pool_mode gauge

cnp_pgbouncer_pools_pool_mode{database="pgbouncer",user="pgbouncer"} 3

#  HELP cnp_pgbouncer_pools_sv_active Server connections that are linked to a client.

#  TYPE cnp_pgbouncer_pools_sv_active gauge
```

(次のページに続く)

(前のページからの続き)

```
cnp_pgbouncer_pools_sv_active{database="pgbouncer",user="pgbouncer"} 0

#  HELP cnp_pgbouncer_pools_sv_active_cancel Server connections that are currently␣
↪forwarding a cancel request

#  TYPE cnp_pgbouncer_pools_sv_active_cancel gauge

cnp_pgbouncer_pools_sv_active_cancel{database="pgbouncer",user="pgbouncer"} 0

#  HELP cnp_pgbouncer_pools_sv_idle Server connections that are unused and␣
↪immediately usable for client queries.

#  TYPE cnp_pgbouncer_pools_sv_idle gauge

cnp_pgbouncer_pools_sv_idle{database="pgbouncer",user="pgbouncer"} 0

#  HELP cnp_pgbouncer_pools_sv_login Server connections currently in the process of␣
↪logging in.

#  TYPE cnp_pgbouncer_pools_sv_login gauge

cnp_pgbouncer_pools_sv_login{database="pgbouncer",user="pgbouncer"} 0

#  HELP cnp_pgbouncer_pools_sv_tested Server connections that are currently running␣
↪either server_reset_query or server_check_query.

#  TYPE cnp_pgbouncer_pools_sv_tested gauge

cnp_pgbouncer_pools_sv_tested{database="pgbouncer",user="pgbouncer"} 0

#  HELP cnp_pgbouncer_pools_sv_used Server connections that have been idle for more␣
↪than server_check_delay, so they need server_check_query to run on them before they␣
↪can be used again.

#  TYPE cnp_pgbouncer_pools_sv_used gauge

cnp_pgbouncer_pools_sv_used{database="pgbouncer",user="pgbouncer"} 0

#  HELP cnp_pgbouncer_pools_sv_wait_cancels Servers that normally could become idle,␣
↪but are waiting to do so until all in-flight cancel requests have completed that␣
↪were sent to cancel a query on this server.

#  TYPE cnp_pgbouncer_pools_sv_wait_cancels gauge
```

(次のページに続く)

(前のページからの続き)

```
cnp_pgbouncer_pools_sv_wait_cancels{database="pgbouncer",user="pgbouncer"} 0

#  HELP cnp_pgbouncer_stats_avg_bind_count Average number of prepared statements␣
↪readied for execution by clients and forwarded to PostgreSQL by pgbouncer.

#  TYPE cnp_pgbouncer_stats_avg_bind_count gauge

cnp_pgbouncer_stats_avg_bind_count{database="pgbouncer"} 0

#  HELP cnp_pgbouncer_stats_avg_client_parse_count Average number of prepared␣
↪statements created by clients.

#  TYPE cnp_pgbouncer_stats_avg_client_parse_count gauge

cnp_pgbouncer_stats_avg_client_parse_count{database="pgbouncer"} 0

#  HELP cnp_pgbouncer_stats_avg_query_count Average queries per second in last stat␣
↪period.

#  TYPE cnp_pgbouncer_stats_avg_query_count gauge

cnp_pgbouncer_stats_avg_query_count{database="pgbouncer"} 0

#  HELP cnp_pgbouncer_stats_avg_query_time Average query duration, in microseconds.

#  TYPE cnp_pgbouncer_stats_avg_query_time gauge

cnp_pgbouncer_stats_avg_query_time{database="pgbouncer"} 0

#  HELP cnp_pgbouncer_stats_avg_recv Average received (from clients) bytes per second.

#  TYPE cnp_pgbouncer_stats_avg_recv gauge

cnp_pgbouncer_stats_avg_recv{database="pgbouncer"} 0

#  HELP cnp_pgbouncer_stats_avg_sent Average sent (to clients) bytes per second.

#  TYPE cnp_pgbouncer_stats_avg_sent gauge

cnp_pgbouncer_stats_avg_sent{database="pgbouncer"} 0

#  HELP cnp_pgbouncer_stats_avg_server_parse_count Average number of prepared␣
↪statements created by pgbouncer on a server.

#  TYPE cnp_pgbouncer_stats_avg_server_parse_count gauge
```

(次のページに続く)

(前のページからの続き)

```
cnp_pgbouncer_stats_avg_server_parse_count{database="pgbouncer"} 0

#  HELP cnp_pgbouncer_stats_avg_wait_time Time spent by clients waiting for a server,␣
↪in microseconds (average per second).

#  TYPE cnp_pgbouncer_stats_avg_wait_time gauge

cnp_pgbouncer_stats_avg_wait_time{database="pgbouncer"} 0

#  HELP cnp_pgbouncer_stats_avg_xact_count Average transactions per second in last␣
↪stat period.

#  TYPE cnp_pgbouncer_stats_avg_xact_count gauge

cnp_pgbouncer_stats_avg_xact_count{database="pgbouncer"} 0

#  HELP cnp_pgbouncer_stats_avg_xact_time Average transaction duration, in␣
↪microseconds.

#  TYPE cnp_pgbouncer_stats_avg_xact_time gauge

cnp_pgbouncer_stats_avg_xact_time{database="pgbouncer"} 0

#  HELP cnp_pgbouncer_stats_total_bind_count Total number of prepared statements␣
↪readied for execution by clients and forwarded to PostgreSQL by pgbouncer

#  TYPE cnp_pgbouncer_stats_total_bind_count gauge

cnp_pgbouncer_stats_total_bind_count{database="pgbouncer"} 0

#  HELP cnp_pgbouncer_stats_total_client_parse_count Total number of prepared␣
↪statements created by clients.

#  TYPE cnp_pgbouncer_stats_total_client_parse_count gauge

cnp_pgbouncer_stats_total_client_parse_count{database="pgbouncer"} 0

#  HELP cnp_pgbouncer_stats_total_query_count Total number of SQL queries pooled by␣
↪pgbouncer.

#  TYPE cnp_pgbouncer_stats_total_query_count gauge

cnp_pgbouncer_stats_total_query_count{database="pgbouncer"} 15
```

(次のページに続く)

(前のページからの続き)

```
#  HELP cnp_pgbouncer_stats_total_query_time Total number of microseconds spent by␣
↪pgbouncer when actively connected to PostgreSQL, executing queries.

#  TYPE cnp_pgbouncer_stats_total_query_time gauge

cnp_pgbouncer_stats_total_query_time{database="pgbouncer"} 0

#  HELP cnp_pgbouncer_stats_total_received Total volume in bytes of network traffic␣
↪received by pgbouncer.

#  TYPE cnp_pgbouncer_stats_total_received gauge

cnp_pgbouncer_stats_total_received{database="pgbouncer"} 0

#  HELP cnp_pgbouncer_stats_total_sent Total volume in bytes of network traffic sent␣
↪by pgbouncer.

#  TYPE cnp_pgbouncer_stats_total_sent gauge

cnp_pgbouncer_stats_total_sent{database="pgbouncer"} 0

#  HELP cnp_pgbouncer_stats_total_server_parse_count Total number of prepared␣
↪statements created by pgbouncer on a server.

#  TYPE cnp_pgbouncer_stats_total_server_parse_count gauge

cnp_pgbouncer_stats_total_server_parse_count{database="pgbouncer"} 0

#  HELP cnp_pgbouncer_stats_total_wait_time Time spent by clients waiting for a␣
↪server, in microseconds.

#  TYPE cnp_pgbouncer_stats_total_wait_time gauge

cnp_pgbouncer_stats_total_wait_time{database="pgbouncer"} 0

#  HELP cnp_pgbouncer_stats_total_xact_count Total number of SQL transactions pooled␣
↪by pgbouncer.

#  TYPE cnp_pgbouncer_stats_total_xact_count gauge

cnp_pgbouncer_stats_total_xact_count{database="pgbouncer"} 15

#  HELP cnp_pgbouncer_stats_total_xact_time Total number of microseconds spent by␣
↪pgbouncer when connected to PostgreSQL in a transaction, either idle in transaction␣
↪or executing queries.
```

(次のページに続く)

(前のページからの続き)

```
# TYPE cnp_pgbouncer_stats_total_xact_time gauge

cnp_pgbouncer_stats_total_xact_time{database="pgbouncer"} 0
```

> **注釈**
>
> メトリックをより深く理解するには、PgBouncer のドキュメントを参照してください。

クラスターに関しては、Prometheus operator's を使用して特定のプーラーを監視できます

PodMonitor 。

次の基本的な例を使用して、特定のプーラーに PodMonitor を展開し、必要に応じて変更できます。

```yaml
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: <POOLER_NAME>
spec:
  selector:
    matchLabels:
      k8s.enterprisedb.io/poolerName: <POOLER_NAME>
  podMetricsEndpoints:
    - port: metrics
```

メトリックエンドポイントの TLS

HTTPS でメトリックエンドポイントを提供するように.spec.monitoring.tls.enabled: true を設定します。デフォルトでは、クラスターのサーバー証明書が使用されます。証明書は TLS ハンドシェイクごとにリロードされるため、ポッドを再起動せずにローテーションが選択されます。

```yaml
spec:
  monitoring:
    tls:
      enabled: true
```

.spec.pgbouncer.clientTLSSecret が設定されている場合、メトリックサーバーは代わりにその証明書を提示します。

```yaml
spec:
  pgbouncer:
    clientTLSSecret:
      name: <CLIENT_TLS_SECRET>
  monitoring:
```

(次のページに続く)

(前のページからの続き)

```
    tls:
      enabled: true
```

Prometheus は IP でポッドをスクレイピングし、証明書の SAN は一般にポッド IP をカバーしないため、生成された PodMonitor は insecureSkipVerify=true でスクレイピングします。

厳密な検証が必要な場合は、.spec.monitoring.enablePodMonitor: false を設定し、PodMonitor を自分で管理します。オペレーター生成されたものは insecureSkipVerify=true にハードコードされ、リコンサイルのたびに仕様を上書きするため、生成された PodMonitor の手動パッチは残りません。

PodMonitor 自動作成の非推奨

現在この機能を使用している場合、 .spec.monitoring.enablePodMonitor を削除または false に設定し、上記のようにプーラーの PodMonitor リソースを手動で作成することを強くお勧めします。この変更により、監視構成の完全な所有権が保証され、オペレーターによって管理または上書きされるのを防ぎます。

5.27.11 ロギング

ログは、次の例のように、JSON 形式で標準出力に直接送信されます。

```
{
  "level": "info",
  "ts": SECONDS.MICROSECONDS,
  "msg": "record",
  "pipe": "stderr",
  "record": {
    "timestamp": "YYYY-MM-DD HH:MM:SS.MS UTC",
    "pid": "<PID>",
    "level": "LOG",
    "msg": "kernel file descriptor limit: 1048576 (hard: 1048576); max_client_conn:␣
↪100, max expected fd use: 112"
  }
}
```

5.27.12 接続の一時停止

Pooler 仕様では、宣言的構成のみを使用して、PgBouncer の PAUSE および RESUME コマンドを利用できます。これは、 paused オプションを使用して行うことができます。デフォルトでは false に設定されています。 true に設定すると、オペレーターは PgBouncer の PAUSE コマンドを内部的に呼び出します。

1. クエリが完了するのを待った後、PostgreSQL サーバーへのアクティブな接続をすべて閉じます。
2. クライアントからの新しい接続を一時停止します

paused オプションが false にリセットされると、オペレーターは PgBouncer で RESUME コマンドを呼び出し、Pooler リソースで定義された PostgreSQL サービスへのタップを再度開きます。

5.27.13 制限事項

単一の PostgreSQL クラスター

プーラーの現在の実装は、CloudNativePG Cluster クラスターサービスの特定の EDB Postgres AI の一部として動作するように設計されています。現在、複数のクラスターにまたがるプーラーを作成することはできません。

制御された構成可能性

CloudNativePG クラスターの EDB Postgres AI は、PgBouncer レイヤーが PostgreSQL と通信するために使用されるいくつかの構成オプションを透過的に管理します。このようなオプションは外部から構成できず、TLS 証明書、認証設定、`databases` セクション、および `users` セクションが含まれます。また、単一の PostgreSQL クラスターの特定のユースケースを考慮して、採用される基準は、ユーザーが構成できるオプションを明示的にリストすることです。

> **注釈**
>
> 採用されたソリューションは、大部分のユースケースに対応する可能性があります。これにより、PgBouncer の別のオペレーターを将来的に実装して、より高度なカスタマイズされたシナリオでガンマを完了する余地が残されています。

5.28 Replica clusters

レプリカクラスターは、CloudNativePG クラスターの EDB Postgres AI、別の PostgreSQL インスタンスからデータをレプリケートするように設計された `Cluster` リソースであり、理想的には CloudNativePG クラスターの EDB Postgres AI によって管理されます。

通常、レプリカクラスターは、別のリージョンの別の Kubernetes クラスターに展開されます。これらのクラスターは、カスケードレプリケーションを実行するように構成でき、以下で詳しく説明するように、ソースからのデータレプリケーションのオブジェクトストアに依存できます。

レプリカクラスターには、主に 2 つのユースケースがあります。

1. **ディザスターリカバリーと高可用性**ディザスターリカバリーと、通常は異なるリージョンにあるさまざまな Kubernetes クラスターにわたる CloudNativePG クラスタークラスターの EDB Postgres AI の高可用性を強化します。 CloudNativePG Cluster 用語の EDB Postgres AI では、これは **分散トポロジ**と呼ばれます。

2. **読み取り専用ワークロード**レポートやオンライン分析処理 OLAP などの目的で、PostgreSQL クラスターのスタンドアロンレプリカを作成します。これらのレプリカは、主に読み取り専用ワークロード用です。CloudNativePG Cluster 用語の EDB Postgres AI では、これは **スタンドアロンレプリカクラスター**と呼ばれます。

例、以下の図 — *Architecture* から取得

—は、主にディザスターリカバリーの目的で機能する対称レプリカクラスターを使用した、2 つの Kubernetes クラスターにまたがる分散 PostgreSQL トポロジを示しています。

5.28.1 基本概念

CloudNativePG クラスターの EDB Postgres AI は、PostgreSQL レプリケーションフレームワークをベースに構築されており、このセクションで説明するレプリカクラスター機能を使用して、既存のソースクラスターから PostgreSQL クラスターを作成および同期できます。ソースは、プライマリクラスターまたは別のレプリカクラスターカスケードレプリケーションです。

PostgreSQL のロールについて

レプリカクラスターは継続的リカバリモードで動作します。つまり、カタログを含むデータベースへの変更、およびロールやデータベースのようなグローバルオブジェクトは許可されません。これらの変更は、`Cluster` がプライマリに移行するまで遅延されます。このフェーズでは、ロールなどのグローバルオブジェクトは、ソースクラスターで定義されたとおりに残ります。CloudNativePG クラスターの EDB Postgres AI は、クラスターが昇格すると、ローカル再定義を適用します。

クラスターを昇格する予定がない場合読み取り専用ワークロードの場合、またはレプリカクラスターが昇格したらソースクラスターから完全に切断する予定がある場合、アクションを実行する必要はありません。これは通常、**スタンドアロンレプリカクラスター** の場合です。

ある時点でクラスターを昇格する予定がある場合、CloudNativePG クラスターの EDB Postgres AI は、レプリカクラスターからプライマリに移行するときに次のロールとパスワードを管理します。

- アプリケーションユーザー
- スーパーユーザー使用している場合
- *declarative interface* を使用して定義されたロール

上記のロールとパスワードが変更されないことをシームレスに保証することが目的である場合、各 `Cluster` で上記に必要なシークレットを定義する必要があります。これは通常、**分散トポロジ** の場合です。

レプリカクラスターのブートストラップ

最初のステップは、次のいずれかの方法を使用してレプリカクラスターをブートストラップすることです。

- `pg_basebackup` 経由の **ストリーミングレプリケーション**
- **ボリュームスナップショットからのリカバリ**
- オブジェクトストアの **Barman Cloud バックアップからのリカバリ**

`pg_basebackup` ストリーミングまたはリカバリーボリュームスナップショットまたはオブジェクトストアを使用して PostgreSQL サーバーのクローンを作成する手順の詳細については、*BootstrapConfiguration* を参照してください。

レプリケーションの構成

レプリカクラスターのベースバックアップが利用できるようになったら、PostgreSQL 継続的リカバリーを使用してオリジンから変更をレプリケートする方法を定義する必要があります。3 つの主なオプションがあります。

1. **ストリーミングレプリケーション**レプリカクラスターとソース間のストリーミングレプリケーションを設定します。この方法では、ネットワーク接続を構成し、適切な管理およびセキュリティ対策を実装して、シームレスなデータ転送を保証する必要があります。

2. **WAL アーカイブ**オブジェクトストアに保存されている WAL 書き込み先行ログアーカイブを使用します。WAL ファイルはソースクラスターからオブジェクトストアに定期的に転送され、そこから Barman Cloud のような CNPG-I プラグイン

`restore_command` を介してレプリカクラスターのそれらを取得します。

3. **ハイブリッドアプローチ**ストリーミングレプリケーションと WAL アーカイブ方法の両方を組み合わせます。PostgreSQL は、必要に応じてこれら 2 つのアプローチを管理および切り替えて、データの一貫性と可用性を保証できます。

外部クラスターの定義

外部クラスターを構成する場合、次のオプションがあります。

- **``plugin`` セクション**

- CNPG-I を使用したレプリカクラスターのブートストラップを有効にします

restore_job をサポートするプラグイン

および wal プロトコル. - CloudNativePG クラスターの EDB Postgres AI は、Barman Cloud Plugin をサポート

オブジェクトストアからレプリカクラスターをブートストラップできるようにします。

- **``connectionParameters`` セクション**

- `pg_basebackup` セクションを使用したストリーミングレプリケーションを介したレプリカクラスターのブートストラップを有効にします。

- CloudNativePG クラスターの EDB Postgres AI は、指定されたプライマリインスタンスに `primary_conninfo` オプションを自動的に設定し、ソースに接続するための WAL レシーバープロセスを開始しますクラスタ化してデータを受信します。

非推奨ですが、**``barmanObjectStore`` セクション**に引き続きアクセスできます。

- CloudNativePG クラスターの EDB Postgres AI が、指定されたプライマリインスタンスで `restore_command` を自動的に設定することで、WAL アーカイブの使用を有効にします。

- ボリュームスナップショットが実現できない場合、`recovery` セクションを使用してオブジェクトストアからレプリカクラスターをブートストラップできます。

バックアップおよびシンメトリックアーキテクチャ

レプリカクラスターは、指定されたプライマリから予約されたオブジェクトストアへのバックアップを実行でき、分散環境での対称アーキテクチャをサポートします。このアーキテクチャの選択は、制御されたデータセンターのスイッチオーバーまたは予期しないイベントに続くフェイルオーバー中にクラスターがプロモーションの準備ができるようにするため、重要です。

分散アーキテクチャの柔軟性

PostgreSQL データベースの優先分散アーキテクチャを柔軟に設計でき、以下から選択できます。

- **プライベートクラウド**異なるデータセンターにある複数の Kubernetes クラスターにまたがります。
- **パブリッククラウド**さまざまなリージョンにある複数の Kubernetes クラスターにまたがります。
- **ハイブリッドクラウド**プライベートクラウドとパブリッククラウドを組み合わせます。
- **マルチクラウド**さまざまなリージョンおよびクラウドサービスプロバイダーにまたがる複数の Kubernetes クラスターにまたがります。

5.28.2 レプリカクラスターのセットアップ

ソースクラスターからレプリカクラスターをセットアップするには、次の手順に従ってクラスター YAML ファイルを作成し、それに応じて構成します。

1. **外部クラスターの定義**

- `externalClusters` セクションで、レプリカクラスターを指定します。
- ディザスターリカバリー DR と高可用性 HA を目的とした分散 PostgreSQL トポロジの場合、このセクションはすべての PostgreSQL クラスターに対して定義する必要があります分散データベースで。

2. **レプリカクラスターのブートストラップ**

- **ストリーミングブートストラップ**ストリーミングレプリケーションを介したブートストラップには `pg_basebackup` セクションを使用します。
- **スナップショット/オブジェクトストアブートストラップ** `recovery` セクションを使用してボリュームスナップショットからブートストラップまたはオブジェクトストア。

3. **継続的リカバリ戦略** `.spec.replica` スタンザで定義します。

- **分散トポロジ** `externalClusters` で定義された分散トポロジとともに、 `primary` 、 `source` 、および `self` フィールドを使用して構成します。これにより、CloudNativePG クラスターの EDB Postgres AI は、プライマリクラスターの降格と、その後のプロモーショントークンを使用したレプリカクラスターのプロモーションを宣言的に制御できます。
- **スタンドアロンレプリカクラスター** `enabled` オプションを使用して継続的リカバリーを有効にし、`source` を設定しますフィールドを `externalClusters` 名前を指すように設定します。この構成は、主に読み取り専用ワークロードを目的としたレプリカの作成に適しています。

継続的リカバリーのための分散トポロジーとスタンドアロンレプリカクラスター戦略の両方を以下で徹底的に説明します。

5.28.3 分散トポロジ

分散 PostgreSQL データベースの計画

ドワイト・アイゼンハワーの有名な言葉「計画がすべてです」。これは、Kubernetes での PostgreSQL アーキテクチャの設計にも当てはまります。

最初に、分散トポロジを紙の上で概念化し、それを CloudNativePG クラスター API 構成の EDB Postgres AI に変換します。この構成には、主に次のものが含まれます。

- `externalClusters` セクション。分散 PostgreSQL セットアップ内のすべての `Cluster` 定義に含める必要があります。
- `.spec.replica` スタンザ、具体的には `primary` 、`source` 、およびオプションの `self` フィールド。

たとえば、南ヨーロッパと中央ヨーロッパにある 2 つの Kubernetes クラスターに分散された PostgreSQL クラスターを展開するとします。

このシナリオでは、CloudNativePG クラスターの EDB Postgres AI が南ヨーロッパの Kubernetes クラスターにインストールされており、プライマリとして機能する `cluster-eu-south` という名前の PostgreSQL `Cluster` が存在すると仮定します。このクラスターには、ローカルオブジェクトストアで構成された継続的バックアップがあります。このオブジェクトストアには、中央ヨーロッパの Kubernetes クラスターにインストールされている `cluster-eu-central` という名前の PostgreSQL `Cluster` からもアクセスできます。最初に、`cluster-eu-central` はレプリカクラスターとして機能します。対称的なアプローチに従い、継続的バックアップ用のローカルオブジェクトストアも持っています。これは、`cluster-eu-south` によって読み取られる必要があります。

この例では、復旧は WAL シッピングのみを介して実行され、2 つのクラスター間のストリーミングレプリケーションは行われません。ただし、 “*Configuring replication*” で説明しているように、ストリーミングレプリケーションを単独で使用するか、ハイブリッドアプローチつまりフォールバックとして WAL シッピングを使用したストリーミングレプリケーションを採用するようにセットアップを構成できます。

セクション。

次に、オブジェクトストアの Barman Cloud プラグインに依存して、両方の `Cluster` リソースの `externalClusters` セクションを構成する方法を示します。

```
#  Distributed topology configuration

externalClusters:
  - name: cluster-eu-south
    plugin:
      name: barman-cloud.cloudnative-pg.io
      parameters:
        barmanObjectName: cluster-eu-south
        serverName: cluster-eu-south
  - name: cluster-eu-central
    plugin:
      name: barman-cloud.cloudnative-pg.io
      parameters:
        barmanObjectName: cluster-eu-central
        serverName: cluster-eu-central
```

`cluster-eu-south` PostgreSQL プライマリ `Cluster` の`.spec.replica` スタンザは次のように構成する必要があります。

```
replica:
  primary: cluster-eu-south
  source: cluster-eu-central
```

一方、 cluster-eu-central PostgreSQL レプリカ Cluster の.spec.replica スタンザは次のように構成する必要があります。

```
replica:
  primary: cluster-eu-south
  source: cluster-eu-south
```

この構成では、primary フィールドが Cluster リソースまたは別のものが使用される場合は.spec.replica.self の名前と一致する場合、現在のクラスターは分散トポロジのプライマリと見なされます。それ以外の場合、source この場合、Barman オブジェクトストアを使用してからのレプリカとして設定されます。

この設定により、複数の Kubernetes クラスターにわたる分散 PostgreSQL アーキテクチャを効率的に管理でき、宣言的構成を使用したプライマリ PostgreSQL クラスターの制御されたスイッチオーバーを介して、高可用性とディザスターリカバリーの両方を保証します。

分散トポロジにおける制御されたスイッチオーバーは、以下を含む 2 段階のプロセスです。

- プライマリクラスターのレプリカクラスターへの降格
- レプリカクラスターのプライマリクラスターへのプロモーション

これらのプロセスについては、次のセクションで説明します。

上記であり、分散トポロジに参加しているすべての Cluster オブジェクトでシークレットを含む同一のロール定義を使用します。

プライマリのレプリカクラスターへの降格

CloudNativePG クラスターの EDB Postgres AI は、プライマリクラスターをレプリカクラスターに降格する機能を提供します。このアクションは、通常、プライマリロールをあるデータセンターから別のデータセンターに移行するときに計画されます。このプロセスには、現在のプライマリクラスター cluster-eu-south などをレプリカクラスターに降格し、その後、完全に同期したときに指定されたレプリカクラスター cluster-eu-central などをプライマリに昇格させることが含まれます。

新しいプライマリになるように選択されたレプリカクラスターをポイントする現在のプライマリ Cluster リソースで外部クラスターを定義している場合、次のように primary フィールドを変更するだけです。

```
replica:
  primary: cluster-eu-central
  source: cluster-eu-central
```

プライマリ PostgreSQL クラスターが降格されると、書き込み操作は可能でなくなります。CloudNativePG クラスターの EDB Postgres AI の場合

1. シャットダウンチェックポイントを含む WAL ファイルを WAL アーカイブの.partial ファイルとしてアーカイブします。

2. ステータスに demotionToken 、システム識別子、タイムスタンプ、タイムライン ID、REDO ロケーション、最新のチェックポイントの REDO WAL ファイルなど、pg_controldata からの関連情報を含む base64 エンコードされた JSON 構造を生成します。

最初のステップは、WAL アーカイブのみを使用して継続的リカバリプロセスにフィードを与えるために降格/プロモートする必要がありますストリーミングレプリケーションなしで。

2 番目のステップ .status.demotionToken の生成は、データの損失や元のプライマリをリビルドすることなく、スムーズな降格/プロモーションプロセスを保証します。

この段階で、元のプライマリはレプリカクラスターに移行し、新しいグローバルプライマリ cluster-eu-central からの WAL データを待機しています。

他のクラスターの昇格を続行するには、次のコマンドを使用して cluster-eu-south から demotionToken を取得する必要があります。

```
kubectl get cluster cluster-eu-south \
  -o jsonpath={.status.demotionToken}
```

cnp プラグインを使用して、クラスターのステータスを確認して demotionToken を取得できます。トークンは Demotion token セクションの下にリストされています。

> **注釈**
>
> *cluster-eu-south* から取得した `demotionToken` は、*cluster-eu-central* の `promotionToken` になります。

cnp プラグインを使用して、クラスターのステータスを確認して、ロールの変更を確認できます。

```
kubectl cnp status cluster-eu-south
```

レプリカのプライマリクラスターへの昇格

PostgreSQL レプリカクラスター cluster-eu-central などをプライマリクラスターに昇格させ、指定されたプライマリを実際のプライマリインスタンスにするには、次の手順を同時に実行する必要があります。

1. .spec.replica.primary を、昇格する現在のレプリカクラスターの名前に設定します cluster-eu-central など。
2. .spec.replica.promotionToken を、以前のプライマリクラスターから取得した値で設定しますプライマリのレプリカクラスターへの降格 を参照してください。

cluster-eu-central の仕様の更新された replica セクションは次のようになります。

```
replica:
  primary: cluster-eu-central
  promotionToken: <PROMOTION_TOKEN>
  source: cluster-eu-south
```

> **警告**
>
> 変更を `primary` および `promotionToken` フィールドに同時に適用することが重要です。プロモーショントークンを省略すると、フェールオーバーがトリガーされ、以前のプライマリのリビルドが必要になります。

これらの調整を行った後、CloudNativePG クラスターの EDB Postgres AI は、レプリカクラスターのプライマリクラスターへのプロモーションを開始します。最初に、CloudNativePG クラスターの EDB Postgres AI は、指定されたプライマリクラスターが、トークンに含まれる指定されたログシーケンス番号 LSN までのすべての書き込み先行ロギング WAL 情報をレプリケートするのを待機します。この目標を達成すると、プロモーションプロセスが開始されます。新しいプライマリクラスターは、タイムラインを切り替え、履歴ファイルと新しい WAL をアーカイブすることにより、レプリカとして動作する `cluster-eu-south` クラスターのレプリケーションプロセスのブロックを解除します。

ロールの変更を確認するには、 `cnp` プラグインを使用して、クラスターのステータスを確認します。

```
kubectl cnp status cluster-eu-central
```

このコマンドは、`cluster-eu-central` の現在のステータスを提供し、プライマリへの昇格を確認します。

これらの手順に従うことで、スムーズで制御されたプロモーションプロセスを保証し、中断を最小限に抑え、PostgreSQL クラスター全体でデータの整合性を維持できます。

5.28.4 スタンドアロンレプリカクラスター

CloudNativePG クラスターの EDB Postgres AI では、スタンドアロンレプリカクラスターは、次の構成で継続的リカバリーの PostgreSQL クラスターです。

- `.spec.replica.enabled` を `true` に設定
- `.spec.replica.source` フィールドを介して定義された物理レプリケーションソース、`externalClusters` 名前を指す

`.spec.replica.enabled` が `false` に設定されている場合、レプリカクラスターは継続的リカバリモードを終了し、プライマリクラスターになり、元のソースから完全に分離されます。

> **警告**
>
> レプリケーションの無効化は、**不可逆的な**オペレーションです。レプリケーションが無効になり、指定されたプライマリがプライマリに昇格すると、レプリカクラスターとソースクラスターは決定的に 2 つの独立したクラスターになります。

分散トポロジとの主な違い

スタンドアロンレプリカクラスターは、ディザスターリカバリーの目的で使用できますが、いくつかの重要な点で「分散トポロジ」戦略とは異なります。

- **分散データベース概念の欠如**スタンドアロンレプリカクラスターは、単純な形式 2 つのクラスターまたはより複雑な構成たとえば、円形トポロジの 3 つのクラスターなど、分散データベースの概念をサポートしていません。
- **グローバルプライマリクラスターなし**スタンドアロンレプリカクラスターには、グローバルプライマリクラスターの概念がありません。
- **制御されたスイッチオーバーなし**スタンドアロンレプリカクラスターは、プライマリにのみ昇格できます。制御されたスイッチオーバーは不可能であるため、以前のプライマリクラスターは再クローンを作成する必要があります。

フェールオーバーは両方の戦略で同じであるため、元のプライマリが復旧した場合は再クローンを作成する必要があります。

pg_basebackup を使用したスタンドアロンレプリカクラスターの例

この **最初の例**は、ブートストラップと継続的リカバリーの両方でストリーミングレプリケーションを使用して、スタンドアロンレプリカクラスターを定義します。レプリカクラスターは、TLS 認証を使用してソースクラスターに接続します。

sample YAML

`samples/` サブディレクトリ。

ソースクラスターを指している `bootstrap` および `replica` セクションに注意してください。

```
bootstrap:
  pg_basebackup:
    source: cluster-example

replica:
  enabled: true
  source: cluster-example
```

前述の構成では、アプリケーションデータベースとその所有ユーザーがデフォルトの `app` に設定されていることを前提としています。リストアする PostgreSQL クラスターが別の名前を使用している場合、**アプリケーションデータベースの構成** の文書に従ってそれらを指定する必要があります。また、元のクラスターからアプリケーションユーザーシークレットをコピーし、ソースとの同期を維持することを検討する必要があります。詳細は、*PostgreSQL のロールについて* を参照してください。

`externalClusters` セクションでは、`connectionParameters` サブセクションのホストに対応した適切な名前空間を使用することに注意してください。レプリカクラスターが別の名前空間にある場合、`-replication` および`-ca` シークレットは、必要に応じてコピーしておく必要があります。

```
externalClusters:
  - name: <MAIN-CLUSTER>
    connectionParameters:
      host: <MAIN-CLUSTER>-rw.<NAMESPACE>.svc
      user: streaming_replica
      sslmode: verify-full
```

(次のページに続く)

(前のページからの続き)

```
      dbname: postgres
    sslKey:
      name: <MAIN-CLUSTER>-replication
      key: tls.key
    sslCert:
      name: <MAIN-CLUSTER>-replication
      key: tls.crt
    sslRootCert:
      name: <MAIN-CLUSTER>-ca
      key: ca.crt
```

オブジェクトストアからのスタンドアロンレプリカクラスターの例

2 番目の例 は、 recovery セクションと継続的リカバリーを使用してオブジェクトストアからブートストラップするレプリカクラスターを定義します。ストリーミングレプリケーションと特定のオブジェクトストアの両方を使用します。ストリーミングレプリケーションの場合、レプリカクラスターは基本認証を使用してソースクラスターに接続します。

sample YAML

samples/ サブディレクトリにあります。

ソースクラスターを指している bootstrap および replica セクションに注意してください。

```
bootstrap:
  recovery:
    source: cluster-example

replica:
  enabled: true
  source: cluster-example
```

前述の構成では、アプリケーションデータベースとその所有ユーザーがデフォルトの app に設定されていることを前提としています。リストアする PostgreSQL クラスターが別の名前を使用している場合、**アプリケーションデータベースの構成** の文書に従ってそれらを指定する必要があります。また、元のクラスターからアプリケーションユーザーシークレットをコピーし、ソースとの同期を維持することを検討する必要があります。詳細については、*PostgreSQL のロールについて* を参照してください。

externalClusters セクションでは、 endpointURL および connectionParameters.host で適切な名前空間を使用するように注意してください。また、必要に応じて必要なシークレットがコピーされていること、およびソースクラスターのバックアップが既に作成されていることを確認してください。

```
externalClusters:
- name: <MAIN-CLUSTER>
  # Example with Barman Cloud Plugin
  plugin:
    name: barman-cloud.cloudnative-pg.io
```

(次のページに続く)

(前のページからの続き)

```
    parameters:
      barmanObjectName: <MAIN-CLUSTER>
      serverName: <MAIN-CLUSTER>
      ...
  connectionParameters:
    host: <MAIN-CLUSTER>-rw.default.svc
    user: postgres
    dbname: postgres
  password:
    name: <MAIN-CLUSTER>-superuser
    key: password
```

注釈

ソースクラスターとレプリカクラスター間でストリーミングレプリケーションを使用するには、2 つのクラスター間のネットワーク接続があることを確認し、パスワードまたは証明書を保持する必要なシークレットがすべて事前に適切に作成されていることを確認する必要があります。

ボリュームスナップショットを使用した例

ボリュームスナップショットを使用し、ストレージクラスがスナップショットクラスター間の可用性を提供する場合、それを活用して、ソースクラスターのボリュームスナップショットを介してレプリカクラスターをブートストラップできます。

3 番目の例 は、 `recovery` セクションを使用してボリュームスナップショットからブートストラップするレプリカクラスターを定義します。ストリーミングレプリケーション基本認証を介して、オブジェクトストアを使用して WAL ファイルを取得します。

sample YAML

`samples/` サブディレクトリにあります。

この例では、アプリケーションデータベースとその所有ユーザーがデフォルトの `app` に設定されていることを前提としています。リストアする PostgreSQL クラスターが別の名前を使用している場合、*Configure theapplication database* の文書に従ってそれらを指定する必要があります。また、元のクラスターからアプリケーションユーザーシークレットをコピーし、ソースとの同期を維持することを検討する必要があります。詳細については、*PostgreSQL* **のロールについて** を参照してください。

5.28.5 遅延レプリカ

CloudNativePG クラスターの EDB Postgres AI は、PostgreSQL の recovery_min_apply_delay を活用して、*.spec.replica.minApplyDelay* を介して**遅延レプリカ** の作成をサポートします。

遅延レプリカは、指定された時間プライマリデータベースより意図的に遅延するように設計されています。この遅延は、PostgreSQL の基になる `recovery_min_apply_delay` パラメーターにマップする `.spec.replica.minApplyDelay` オプションを使用して構成できます。

遅延レプリカの主な目的は、プライマリデータベースでの意図しない SQL ステートメントの実行の影響を軽減することです。これは、UPDATE や DELETE などの操作が適切な WHERE 句なしで実行されるシナリオで特に役立ちます。

レプリカクラスターで遅延を構成するには、 .spec.replica.minApplyDelay オプションを調整します。このパラメーターは、レプリカがプライマリからどれくらいの時間を決定します。例

```
#  ...

replica:
  enabled: true
  source: cluster-example
  # Enforce a delay of 8 hours
  minApplyDelay: "8h"

#  ...
```

上記の例では、8 時間のバッファ期間を提供して、問題がレプリカに伝播する前に検出および修正することにより、誤ったデータの変更に対する安全対策に役立ちます。

目標復旧時間と意図しないプライマリデータベース操作の潜在的な影響に基づいて、必要に応じて遅延を監視および調整します。

遅延レプリカの主な使用ケースは次のように要約できます。

1. 人的エラーの軽減プライマリデータベースでの意図しない SQL 操作に起因するデータの破損または損失のリスクを削減します。
2. 復旧時間の最適化変更が他のレプリカに適用される前に問題を特定および修正できる遅延レプリカを使用することにより、意図しない変更からの迅速な復旧を促進します。
3. データ保護の強化不要な変更の伝播を介入および防止する機会を提供する時間バッファを導入することにより、重要なデータを保護します。

> **警告**
>
> 遅延レプリカの `minApplyDelay` オプションは、*promotionToken* と組み合わせて使用することはできません。

遅延レプリカをレプリケーション戦略に統合することにより、PostgreSQL 環境の復元力とデータ保護機能を強化できます。特定のニーズとデータの重要度に基づいて、遅延期間を調整します。

5.29 Kubernetes upgrade and maintenance

最新の Kubernetes クラスターを維持することは、最適なパフォーマンスとセキュリティを保証するために重要であり、特に自己管理クラスター、特にベアメタルインフラストラクチャで実行されているクラスターの場合。定期的な更新は、メンテナンスのためにクラスターからノードを一時的に削除することに関連するダウンタイムの制御にもかかわらず、技術的負債に対処し、ビジネスリスクを軽減するのに役立ちます。オペレーションでのリスクの受け入れに関する詳細な洞察については、Embracing Risk を参照してください。

Site Reliability Engineering の本からの章。

5.29.1 定期的なアップデートの重要性

Kubernetes の更新には、基になる Linux サーバーへのセキュリティ更新の適用、故障したハードウェアコンポーネントの交換、クラスターの最新の Kubernetes バージョンへのアップグレードなどのメンテナンスタスクの計画と実行が含まれます。これらのアクティビティは、堅牢で安全なインフラストラクチャを維持するために不可欠です。

5.29.2 クラスター内のメンテナンス操作

通常、メンテナンス操作は、structured process に従って、一度に 1 つのノードで実行されます。

1. ワークロードのエビクション `drain` ワークロードは更新するノードから正常に移動され、スムーズな移行が保証されます。
2. 操作の実行システムの更新やハードウェアの交換など、実際のメンテナンス操作が実行されます。
3. ノードのクラスター `uncordon` への再参加更新されたノードはクラスターに再統合され、その責任を再開する準備が整います。

このプロセスでは、アップグレード期間全体でワークロードを停止するか、クラスター内の他のノードに移行する必要があります。

5.29.3 一時的な PostgreSQL クラスターの低下

標準のアプローチはサービスの信頼性を保証し、Kubernetes の自己修復機能を活用しますが、一時的に機能が低下したクラスターでの動作が受け入れられるシナリオがあります。これは、**ノードローカルストレージ**に依存する PostgreSQL クラスターに特に関連します。ここで、ストレージは PostgreSQL データベースを実行している Kubernetes ワーカーノードにローカルです。ノードローカルストレージまたは単に_local storage_は、パフォーマンスを向上させるために使用されます。

> **注釈**
>
> データベースファイルがネットワーク経由でアクセスできる共有ストレージにある場合、オペレーターのデフォルトの自己修復動作は、ドレイン操作の後に別のノードのポッドによってボリュームが再利用されるシナリオを効率的に処理できます。このような場合、このドキュメントの残りのセクションをスキップできます。

5.29.4 ポッドの中断予算

デフォルトでは、CloudNativePG クラスターの EDB Postgres AI は、Postgres クラスター操作を保護します。ノードがドレインされ、クラスターのプライマリインスタンスが含まれている場合、スイッチオーバーはドレインの前に発生します。ノードのインスタンスがレプリカにダウングレードされると、ドレインを再開できます。単一インスタンスクラスターの場合、スイッチオーバーは不可能であるため、CloudNativePG クラスターの EDB Postgres AI は、インスタンスが収容されているノードのドレインを防ぎます。さらに、3 つ以上のインスタンスがあるクラスターでは、CloudNativePG クラスターの EDB Postgres AI は、ドレイン操作中に一度に 1 つのレプリカのみが正常にシャットダウンされることを保証します。

各 PostgreSQL `Cluster` には、関連する 2 つの `PodDisruptionBudget` リソースが装備されています。`kubectl get pdb` コマンドで簡単に確認できます。

すべての運用 Postgres クラスターでポッドの中断バジェットを有効にしたままにすることをお勧めします。これは、*API Reference - v1.30.0* で詳細に説明しているように、 `.spec.enablePDB` オプションを切り替えることにより簡単に管理できます。

5.29.5 開発またはテストに使用される PostgreSQL クラスター

開発目的で使用される PostgreSQL クラスターの場合、多くの場合単一のインスタンスで構成され、ポッドの中断バジェットを無効にすることが重要です。これを行わないと、そのクラスターをホストするノードはドレインされません。

次の例は、1 インスタンスの開発クラスターのポッド中断バジェットを無効にする方法を示しています。

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: dev
spec:
  instances: 1
  enablePDB: false

  storage:
    size: 1Gi
```

この構成により、開発アクティビティ中にノードのドレインを制限せずに、よりスムーズなメンテナンス手順が保証されます。

5.29.6 ノードのメンテナンスウィンドウ

リリース 1.23 より前には、CloudNativePG クラスターの EDB Postgres AI には、ローカルストレージを扱う際に Kubernetes のアップグレードを管理する宣言的メカニズムが 1 つだけありました。標準の自己修復を回避するには、`nodeMaintenanceWindow` オプションを介してクラスターを一時的に **メンテナンスモード**にする必要がありましたたとえば、物理ノードのパーティションを拡大したり、ノード自分自身を更新したりしながら、キックインするプロシージャ。

> **警告**
>
> メンテナンスウィンドウの期間を可能な限り短い時間に制限します。このフェーズでは、Kubernetes の予想される動作の一部が無効になっているか、自己修復、ローリング更新、ポッドの中断バジェットなどの制限付きで実行されています。

クラスターの `nodeMaintenanceWindow` オプションには、さらに 2 つの設定があります。

`inProgress` ノードのメンテナンスウィンドウが現在進行中かどうかを示すブール値。デフォルトでは、`off` に設定されています。メンテナンスウィンドウ中に、以下の `reusePVC` オプションがオペレーターによって評価されます。

reusePVC メンテナンス操作中に既存の PVC を再利用するかどうかを定義するブール値。デフォルトでは、on に設定されています。 **有効になっている**、Kubernetes はノードが再び起動するのを待機して、既存の PVC を再利用します。PodDisruptionBudget ポリシーは一時的に削除されます。**無効になっている**、Kubernetes は、PostgreSQL の物理的ストリーミングレプリケーションに依存して、新しい PVC を使用して別のノードでポッドの再作成を強制し、ポッドとともに古い PVC を破棄します。このシナリオは、データベースのサイズが小さく、新しい PostgreSQL インスタンスの再クローンの作成にかかる時間が待機するよりも短い場合を除き、一般的にお勧めできません。この動作は、インスタンスが 1 つだけで再利用 PVC が無効になっているクラスターには適用されません。以下のセクションを参照してください。

注釈

kubectl drain コマンドを実行するときは、*--delete-emptydir-data* オプションを追加する必要があります。恐れないでください。これは、PostgreSQL データディレクトリではなく、オペレーターが内部的に使用する別のボリュームを参照します。

reusePVC が false に設定された単一インスタンスクラスター

reusePVC が false に設定されている単一インスタンスクラスター内の唯一の PostgreSQL インスタンスを削除することは、すべてのデータが失われることを意味します。したがって、メンテナンスモードでも、ユーザーがそのようなインスタンスが実行されているノードをドレインできないようにします。

ただし、このようなノードのメンテナンスが必要な場合、2 つのオプションがあります。

1. reusePVC を有効にし、ダウンタイムを受け入れます
2. 別のノードでインスタンスを複製し、プライマリをスイッチオーバーします

データベースサービスのダウンタイムがご使用の環境で許容される限り、ノードのドレインは、nodeMaintenanceWindow を inProgress: true および reusePVC: true に設定するだけで簡単です。これにより、元の PVC が利用可能になるとすぐにインスタンスを削除して再作成できますたとえば、ノードローカルストレージを使用して、ノードがバックアップされたらすぐに。

それ以外の場合は、クラスターをスケールアップし、別のノードで新しいインスタンスを作成し、新しいインスタンスをプライマリに昇格させて、メンテナンス中のノードで元のインスタンスをシャットダウンする必要があります。この場合の唯一のダウンタイムは、スイッチオーバーの期間です。

考えられるアプローチは次のとおりです。

1. 現在のインスタンスが実行されているノードを遮断します。
2. クラスターを 2 つのインスタンスにスケールアップします。データベースのサイズによっては、時間がかかる場合があります。
3. 現在のプライマリが遮断されたノードで実行されている場合、新しいインスタンスが実行されると、オペレーターは自動的にスイッチオーバーを実行します。
4. クラスターを単一のインスタンスにスケールダウンします。これにより、古いインスタンスが削除されます
5. 新しいノードで新しいプライマリを実行したまま、古いプライマリのノードを正常にドレインできるようになりました。

5.30 PostgreSQL upgrades

PostgreSQL のアップグレードは、2 つのカテゴリに分類されます。

- マイナーバージョンのアップグレード 例 18.0 から 18.1
- メジャーバージョンのアップグレード 例、16.x から 18.0

5.30.1 マイナーバージョンのアップグレード

PostgreSQL バージョン番号は `major.minor` 形式に従います。例、バージョン 18.1 では

- 18 はメジャーバージョンです
- 1 はマイナーバージョンです

マイナーリリースは、同じメジャーバージョンの以前および後のマイナーリリースと完全な互換性があります。これらにはバグ修正とセキュリティ更新が含まれていますが、内部ストレージ形式への変更は導入されません。

CloudNativePG クラスターの EDB Postgres AI のマイナーバージョンのアップグレード

新しいマイナーバージョンにアップグレードするには、クラスター定義の PostgreSQL コンテナイメージ参照を直接またはイメージカタログを介して更新するだけです。CloudNativePG クラスターの EDB Postgres AI は *rolling update of the cluster* をトリガーし、各インスタンスをレプリカから 1 つずつ置き換えます。すべてのレプリカが更新されると、プライマリのスイッチオーバーまたは再起動が実行されてプロセスが完了します。

5.30.2 メジャーバージョンのアップグレード

PostgreSQL のメジャーリリースでは、内部データストレージ形式の変更が導入され、より構造化されたアップグレードプロセスが必要です。

CloudNativePG クラスターの EDB Postgres AI は、メジャーアップグレードを実行する 3 つの方法をサポートしています。

1. *Logical dump/restore* – ブルー/グリーン展開、オフライン。
2. *Native logical replication* – ブルー/グリーン展開、オンライン。
3. `pg_upgrade` を使用した物理的 – インプレースアップグレード、オフライン以下の *PostgreSQL* のオフラインインプレースメジャーアップグレード でカバーされています。

各方法には、ダウンタイム、複雑さ、データ量の処理の点でトレードオフがあります。最適なアプローチは、アップグレード戦略と操作の制約によって異なります。

5.30.3 オフラインインプレースメジャーアップグレード

CloudNativePG クラスターの EDB Postgres AI は、より高い PostgreSQL メジャーバージョンを使用した新しいオペランドコンテナイメージがクラスターで宣言的に要求された場合、**オフラインインプレースメジャーアップグレード** を実行します。

警告

PostgreSQL 17.0〜17.5 にはバグがあり、*max_slot_wal_keep_size* パラメーターが `-1` 以外の値に設定されている場合、アップグレードの成功を妨げます。アップグレードプロセスは、レプリケーションスロット構成に関連するエラーで失敗します。この号は fixed in PostgreSQL 17.6 and 18 or later versions になりました。 PostgreSQL 17.0〜17.5 を使用している場合は、メジャーアップグレードを試みる前に少なくとも PostgreSQL 17.6 にアップグレードするか、クラスター構成で `max_slot_wal_keep_size` パラメーターを `-1` に一時的に設定してください。

次の 2 つの方法のいずれかでアップグレードをトリガーできます。

- `.spec.imageName` オプションを介してイメージタグのメジャーバージョンを更新することにより。
- **イメージカタログを使用する** を使用してバージョンの変更を管理する。

サポートされているイメージタグの詳細については、**イメージタグの要件** を参照してください。

警告

CloudNativePG クラスターの EDB Postgres AI は、PostgreSQL 拡張機能を担当していません。ソース PostgreSQL イメージの拡張機能がターゲットイメージの拡張機能と互換性があり、アップグレードパスがサポートされていることを確認する必要があります。予期しない問題を回避するために、事前にアップグレードプロセスを徹底的にテストしてください。 *extensions management feature*

拡張機能のアップグレードを宣言的に管理するのに役立ちます。

アップグレードプロセス

1. すべてのクラスターポッドをシャットダウンして、データの一貫性を保証します。
2. 以前の PostgreSQL バージョンとイメージを `.status.pgDataImageInfo` の下のクラスターのステータスに記録します。
3. 新しいアップグレードジョブを開始します。
    - イメージのバイナリとデータファイルがメジャーアップグレード要求と一致することを確認します。
    - `PGDATA` 、および該当する場合は WAL ファイルとテーブルスペースの新しいディレクトリを作成します。
    - 次を使用してアップグレードを実行します。 `pg_upgrade --link` オプションを使用した場合
    - 正常に完了すると、元のディレクトリをアップグレードされた対応するディレクトリに置き換えます。

警告

アップグレードプロセス中、レプリカを含む PostgreSQL クラスター全体は、アプリケーションで使用できません。続行する前に、システムがこのダウンタイムを許容できることを確認してください。

警告

インプレースアップグレードの実行は、固有のリスクを伴う例外的な操作です。アップグレードプロセスを開始する前に、クラスターの完全なバックアップを作成することを強くお勧めします。

注釈

pg_upgrade の詳細なガイダンスについては、公式 PostgreSQL documentation を参照してください。

アップグレード後のアクション

アップグレードが成功した場合、CloudNativePG クラスターの EDB Postgres AI

- レプリカの PVC を破壊します利用可能な場合。
- 必要に応じて、レプリカをスケールアップします。

警告

レプリカの再クローン作成は、特に非常に大規模なデータベースの場合、時間がかかる場合があります。潜在的な遅延に対応できるように、それに応じて計画を立てます。アップグレードが完了したら、できるだけ早く新しいベースバックアップを作成します。アップグレード前のバックアップと WAL ファイルは、メジャーバージョンの境界を越えたポイントインタイムリカバリー PITR に使用できません。*Backup and WAL Archive Considerations* を参照

詳細については、

警告

pg_upgrade は、オプティマイザー統計を転送しません。アップグレードの後、データベースで`ANALYZE`を実行して更新することができます。

アップグレードが失敗した場合は、クラスターの構成のイメージを以前のメジャーバージョンに戻します。オペレーターはロールバックを検出し、失敗したアップグレードジョブを自動的に削除するため、クラスターを元のバージョンで再起動できます。

メジャーアップグレード中の拡張機能

クラスターがイメージボリューム拡張を宣言すると、アップグレード Job は、ソースバージョンとターゲットバージョンの両方の拡張イメージをサイドバイサイドでマウントします。ソースバージョンの拡張機能が必要であるため、古いサーバーは、既に配置されている既存の共有ライブラリで起動できます。pg_upgrade が新しいメジャーへのアップグレードを適切に実行できるように、ターゲットバージョンの拡張機能新しい PostgreSQL メジャー用に構築されたものも存在する必要があります。

- ソースバージョンの拡張機能は/extensions/<name> 定常状態パスにマウントされているため、古い PGDATA の dynamic_library_path および extension_control_path GUC は既存の値を保持します。これにより、メジャーアップグレードが失敗した場合に、古いメジャーバージョンへの復帰がシームレスに機能します。
- ターゲットバージョンの拡張機能は、アップグレードジョブ中に/new-extensions/<name> に一時的にマウントされ、dynamic_library_path および extension_control_path は、新しいメジャーの PGDATA に応じて構成されます。

LD_LIBRARY_PATH および PATH は両方のセットで拡張されているため、どちらのバージョンのバイナリと共有オブジェクトも到達可能です。extensions[].env を介して宣言された環境変数は、*Precedence and Conflict Resolution* で説明されているルールに従います。

> **注釈**
>
> ターゲットバージョンのエントリは、ソースバージョンのエントリの**後**適用されます。両方に同じ変数が定義されている場合、ターゲットバージョンの値が優先されます。

pg_upgrade は、ターゲットクラスターに default_version がインストールされたままのバージョン pg_upgrade より新しい拡張機能が含まれている場合、PGDATA 内の update_extensions.sql スクリプトを発行します。そのスクリプトが適用されるまで、pg_catalog の SQL レベルの拡張メタデータは、実行中のサーバーによって実際にロードされた共有ライブラリよりも遅延します。

スクリプトは、プライマリポッドの PGDATA 内にあります。データベーススーパーユーザー postgres を使用して実行して、すべてのデータベースのこれらの拡張機能を更新できます。

```
PRIMARY=$(kubectl get cluster <name> -o jsonpath={.status.currentPrimary})
SCRIPT=/var/lib/postgresql/data/pgdata/update_extensions.sql

kubectl exec -i "$PRIMARY" -- psql -f "$SCRIPT"
```

> **注釈**
>
> 上記のスクリプトパスは、CloudNativePG クラスターオペランドイメージの EDB Postgres AI が使用するデフォルトの `PGDATA` の場所を想定しています。オペレーターは、*pg_upgrade* がスクリプトを発行するときに解決されたパスもログに記録するため、オペランドイメージが別の `PGDATA` を使用しているかどうかをオペレーターログに確認します。

メジャーアップグレードを実行すると、pg_upgrade は新しい_System ID_で新しいデータベースシステムを作成し、PostgreSQL タイムラインを 1 にリセットします。これは、バックアップと WAL アーカイブに

影響を与えます。

- **タイムラインファイルの競合**新しいタイムライン 1 ファイルは、元のクラスターのタイムライン 1 ファイルを上書きする場合があります。
- **混合バージョンアーカイブ**介入がなければ、アーカイブには WAL ファイルと両方の PostgreSQL バージョンのバックアップが含まれます。

> **警告**
>
> ポイントインタイムリカバリー PITR は、メジャー PostgreSQL バージョンの境界を超えてサポートされていません。アップグレード前のバックアップを使用して、アップグレード後の特定の時点にリカバリすることはできません。アップグレード後できるだけ早く新しいベースバックアップを取得して、新しいメジャーバージョンのリカバリベースラインを確立します。

バックアップシステムがメジャーアップグレードを処理する方法は、プラグイン実装によって異なります。一部のプラグインは、アップグレード中にアーカイブの分離を自動的に管理する場合がありますが、メジャーバージョンごとに異なるアーカイブパスを使用するには手動構成が必要なプラグインもあります。メジャーアップグレード中の特定の動作については、バックアッププラグインのドキュメントを参照してください。

例 Barman Cloud プラグインを使用した手動アーカイブパスの分離

Barman Cloud プラグインは、メジャーアップグレード中にアーカイブを自動的に分離しません。アップグレード前のバックアップを保存し、アーカイブをクリーンな状態に保つには、アップグレードをトリガーするときに `serverName` パラメーターを変更します。

アップグレード前 PostgreSQL 16

```
spec:
  imageName: docker.enterprisedb.com/k8s/postgresql:16-minimal-ubi9
  plugins:
    - name: plugin-barman-cloud
      enabled: true
      parameters:
        destinationPath: s3://my-bucket/
        serverName: cluster-example-pg16
```

アップグレードをトリガーするには、 `imageName` と `serverName` の両方を一緒に変更します。

```
spec:
  imageName: docker.enterprisedb.com/k8s/postgresql:18-minimal-ubi9
  plugins:
    - name: plugin-barman-cloud
      enabled: true
      parameters:
        destinationPath: s3://my-bucket/
        serverName: cluster-example-pg18
```

この構成では、 cluster-example-pg16 の古いアーカイブはアップグレード前リカバリーのためにそのまま残りますが、アップグレードされたクラスターは cluster-example-pg18 に書き込みます。

> **注釈**
>
> 非推奨のツリー内 `barmanObjectStore` 実装では、メジャーアップグレード中に別のアーカイブを手動で `serverName` 変更する必要があります。

例メジャーアップグレードの実行

バージョン 16 を実行している次の PostgreSQL クラスターを検討します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  imageName: docker.enterprisedb.com/k8s/postgresql:16-minimal-ubi9
  instances: 3
  storage:
    size: 1Gi
```

次のコマンドを使用して、現在の PostgreSQL バージョンを確認できます。

```
kubectl cnp psql cluster-example -- -qAt -c SELECT version()
```

これにより、次のような出力が返されます。

```
PostgreSQL 16.x ...
```

クラスターをバージョン 18 にアップグレードするには、メジャーバージョンタグを 16 から 18 に変更して、imageName フィールドを更新します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  imageName: docker.enterprisedb.com/k8s/postgresql:18-minimal-ubi9
  instances: 3
  storage:
    size: 1Gi
```

アップグレードプロセス

1. クラスターのシャットダウン – 一貫したアップグレードを保証するために、すべてのクラスターポッドが終了します。
2. アップグレードジョブの実行 – 新しいジョブは、プライマリポッドの名前にサフィックス `-major-upgrade` を追加した名前で作成されます。このジョブは、プライマリの永続ボリュームグループで `pg_upgrade` を実行します。
3. アップグレード後の手順
   - レプリカの PVC グループ `cluster-example-2` および `cluster-example-3` が削除されます。
   - プライマリポッドが再起動されます。
   - 2 つの新しいレプリカ `cluster-example-2` および `cluster-example-3` が、アップグレードされたプライマリから再クローンされます。

アップグレードが完了すると、同じコマンドを実行して新しいメジャーバージョンを確認できます。

```
kubectl cnp psql cluster-example -- -qAt -c SELECT version()
```

これにより、次のような出力が返されます。

```
PostgreSQL 18.x ...
```

`app` データベースで `ANALYZE` を実行することにより、統計を更新できるようになりました。

```
kubectl cnp psql cluster-example -- app -c ANALYZE
```

5.31 EDB Postgres AI for CloudNativePG Cluster Plugin

CloudNativePG クラスターの EDB Postgres AI は、Kubernetes でクラスターを管理するための `kubectl` のプラグインを提供します。プラグインは、OpenShift 環境での `oc` でも動作します。

5.31.1 インストール

さまざまな方法を使用して `cnp` プラグインをインストールできます。

> **注釈**
>
> エアギャップシステムの場合、以前にダウンロードしたファイルを使用して、パッケージマネージャーを介したインストールが良いオプションである場合があります。

インストールスクリプトを介して

```
curl -sSfL \
  https://github.com/EnterpriseDB/kubectl-cnp/raw/main/install.sh | \
  sudo sh -s -- -b /usr/local/bin
```

Debian または RedHat パッケージを使用する

releases section of the GitHub repository では、関心のあるリリースに移動できます CloudNativePG クラスターオペレーターの EDB Postgres AI と同じまたは新しいリリースを選択します。その中には **Assets** セクションがあります。そのセクションには、さまざまなシステム用の事前ビルドパッケージが含まれています。その結果、標準の手順と手順に従って、システムにインストールできます。

Debian パッケージ

たとえば、Intel ベースの 64 ビットサーバーの場合、プラグインの 1.30.0 リリースをインストールしてみましょう。最初に、適切な.deb ファイルをダウンロードします。

```
wget https://github.com/EnterpriseDB/kubectl-cnp/releases/download/v1.30.0/kubectl-
↪cnp_1.30.0_linux_x86_64.deb \
  --output-document kube-plugin.deb
```

次に、スーパーユーザー権限で、 dpkg を使用してローカルファイルからインストールします。

```
sudo dpkg -i kube-plugin.deb
```

```
Selecting previously unselected package cnp.
(Reading database ... 6688 files and directories currently installed.)
Preparing to unpack kube-plugin.deb ...
Unpacking kubectl-cnp (1.30.0) ...
Setting up kubectl-cnp (1.30.0) ...
```

RPM パッケージ

.rpm パッケージの例と同様に、Intel 64 ビットマシンの 1.30.0 リリースをインストールしましょう。ファイル名を提供する--output フラグに注意してください。

```
curl -L https://github.com/EnterpriseDB/kubectl-cnp/releases/download/v1.30.0/kubectl-
↪cnp_1.30.0_linux_x86_64.rpm \
  --output kube-plugin.rpm
```

次に、スーパーユーザー権限を使用して、yum でインストールすると、使用する準備が整います。

```
sudo yum --disablerepo=* localinstall kube-plugin.rpm
```

```
Failed to set locale, defaulting to C.UTF-8
Dependencies resolved.
================================================================================
 Package            Architecture         Version                  Repository
         Size
================================================================================
Installing:
```

(次のページに続く)

(前のページからの続き)

```
 cnp                x86_64               1.30.0-1                   @commandline
         20 M

Transaction Summary
================================================================================
Install  1 Package

Total size: 20 M
Installed size: 78 M
Is this ok [y/N]: y
```

サポートされているアーキテクチャ

CloudNativePG クラスタープラグインの EDB Postgres AI は、現在次のオペレーティングシステムとアーキテクチャ用にビルドされています。

- Linux
  - amd64
  - arm 5/6/7
  - arm64
  - s390x
  - ppc64le
- macOS
  - amd64
  - arm64
- Windows
  - 386
  - amd64
  - arm 5/6/7
  - arm64

オートコンプリートの設定

プラグインのオートコンプリートを構成するには、ヘルパーシェルスクリプトを現在の PATH にインストールする必要があります。後者に`/usr/local/bin`が含まれていると仮定すると、これは次のコマンドで実行できます。

```
cat > kubectl_complete-cnp <<EOF
# !/usr/bin/env sh

#  Call the __complete command passing it all arguments

kubectl cnp __complete "\$@"
EOF

chmod +x kubectl_complete-cnp

#  Important: the following command may require superuser permission

sudo mv kubectl_complete-cnp /usr/local/bin
```

5.31.2 使用する

プラグインがインストールおよび展開されたら、次のように使用を開始できます。

```
kubectl cnp <command> <args...>
```

> **注釈**
>
> プラグインは、標準出力チャンネルが端末に接続されているかどうかを自動的に検出します。このような場合、ANSI カラーをコマンド出力に追加する場合があります。色を無効にするには、コマンドで `--color=never` オプションを使用します。

インストールマニフェストの生成

cnp プラグインを使用して、オペレーターのインストールのための YAML マニフェストを生成できます。このオプションは通常、レプリカの数、インストール名前空間、監視する名前空間などのデフォルト構成をオーバーライドする場合に使用されます。

詳細および使用可能なオプションについては、次を実行します。

```
kubectl cnp install generate --help
```

主なオプションは次のとおりです。

- `-n` オペレーターデフォルト `postgresql-operator-system` をインストールする名前空間を指定します。
- `--control-plane` true に設定されている場合、オペレーターの展開には `node-role.kubernetes.io/control-plane` の許容範囲とアフィニティが含まれます。
- `--replicas` 展開内のレプリカの数を設定します。
- `--watch-namespace` 監視する名前空間のコンマ区切りリストを指定しますデフォルトすべての名前空間。

- --version 1.23 など、インストールするオペレーターのマイナーバージョンを定義します。マイナーバージョンが指定された場合、プラグインはそのマイナーバージョンの最新のパッチバージョンをインストールします。バージョンが指定されない場合、プラグインはオペレーターの最新の MAJOR.MINOR.PATCH バージョンをインストールします。

オペレーターをインストールする YAML マニフェストを生成する generate コマンドの例は、次のとおりです。

```
kubectl cnp install generate \
  -n king \
  --version 1.23 \
  --replicas 3 \
  --watch-namespace "albert, bb, freddie" \
  > operator.yaml
```

上記のコマンドのフラグは、次の意味を持っています。

- -n king {{name.abbr}}演算子を king 名前空間にインストールします
- --version 1.23 マイナーバージョン 1.23 の最新パッチバージョンをインストールします
- --replicas 3 3 つのレプリカを使用してオペレーターをインストールします
- --watch-namespace "albert, bb, freddie" は、オペレーターに albert 、bb 、および freddie 名前空間の変更のみを監視させます

ステータス

status コマンドは、クラスターの現在のステータスの概要を提供します。

- **一般情報**クラスターの名前、PostgreSQL のシステム ID、インスタンス数、現在のタイムライン、WAL 内の位置
- **backup** リカバリー可能ポイント、およびプライマリまたはレプリカクラスターの場合は指定されたプライマリから pg_stat_archiver ビューによって返される WAL アーカイブステータス
- **ストリーミングレプリケーション**プライマリインスタンスの pg_stat_replication ビューから直接取得した情報
- **instances** 各 Postgres インスタンスに関する情報、各インスタンスマネージャーによって直接取得されます。スタンバイの場合、Current LSN フィールドは、リカバリー中に再生された最新のログ先行書き込みの場所リプレイ LSN に対応します。

```
kubectl cnp status sandbox
```

```
Cluster Summary
Name:                default/sandbox
System ID:           7423474350493388827
PostgreSQL Image:    docker.enterprisedb.com/k8s/edb-postgres-extended:18.1-standard-
↪ubi9
Primary instance:    sandbox-1
```

(次のページに続く)

(前のページからの続き)

```
Primary start time:  2024-10-08 18:31:57 +0000 UTC (uptime 1m14s)
Status:              Cluster in healthy state
Instances:           3
Ready instances:     3
Size:                126M
Current Write LSN:   0/604DE38 (Timeline: 1 - WAL File: 000000010000000000000006)

Continuous Backup status
Not configured

Streaming Replication status
Replication Slots Enabled
Name       Sent LSN   Write LSN  Flush LSN  Replay LSN  Write Lag  Flush Lag  Replay␣
↪Lag  State      Sync State  Sync Priority  Replication Slot
- ---       --------   ---------  ---------  ----------  ---------  ---------  -------
↪---  -----      ----------  -------------  ----------------
sandbox-2  0/604DE38  0/604DE38  0/604DE38  0/604DE38   00:00:00   00:00:00  ␣
↪00:00:00    streaming  async       0              active
sandbox-3  0/604DE38  0/604DE38  0/604DE38  0/604DE38   00:00:00   00:00:00  ␣
↪00:00:00    streaming  async       0              active

Instances status
Name       Current LSN  Replication role  Status  QoS         Manager Version  Node
- ---       -----------  ----------------  ------  ---         ---------------  ----
sandbox-1  0/604DE38    Primary           OK      BestEffort  1.30.0           k8s-eu-
↪worker
sandbox-2  0/604DE38    Standby (async)   OK      BestEffort  1.30.0           k8s-eu-
↪worker2
sandbox-3  0/604DE38    Standby (async)   OK      BestEffort  1.30.0           k8s-eu-
↪worker
```

より詳細なステータス情報が必要な場合は、 `--verbose` オプションまたは略して`-v` を使用します。フラグが繰り返されるたびに、詳細レベルが増加します。

```
kubectl cnp status sandbox --verbose
```

```
Cluster Summary
Name:                default/sandbox
System ID:           7423474350493388827
PostgreSQL Image:    docker.enterprisedb.com/k8s/edb-postgres-extended:18.1-standard-
↪ubi8
Primary instance:    sandbox-1
Primary start time:  2024-10-08 18:31:57 +0000 UTC (uptime 2m4s)
Status:              Cluster in healthy state
Instances:           3
```

(次のページに続く)

(前のページからの続き)

```
Ready instances:     3
Size:                126M
Current Write LSN:   0/6053720 (Timeline: 1 - WAL File: 000000010000000000000006)

Continuous Backup status
Not configured

Physical backups
No running physical backups found

Streaming Replication status
Replication Slots Enabled
Name       Sent LSN   Write LSN  Flush LSN  Replay LSN  Write Lag  Flush Lag  Replay␣
↪Lag  State      Sync State  Sync Priority  Replication Slot  Slot Restart LSN  Slot␣
↪WAL Status  Slot Safe WAL Size
- ---       --------   ---------  ---------  ----------  ---------  ---------  -------
↪---  -----      ----------  -------------  ----------------  ----------------  -----
↪----------  ------------------
sandbox-2  0/6053720  0/6053720  0/6053720  0/6053720   00:00:00   00:00:00
 00:00:00    streaming  async       0              active            0/6053720
 reserved         NULL
sandbox-3  0/6053720  0/6053720  0/6053720  0/6053720   00:00:00   00:00:00
 00:00:00    streaming  async       0              active            0/6053720
 reserved         NULL

Unmanaged Replication Slot Status
No unmanaged replication slots found

Managed roles status
No roles managed

Tablespaces status
No managed tablespaces

Pod Disruption Budgets status
Name             Role     Expected Pods  Current Healthy  Minimum Desired Healthy ␣
↪Disruptions Allowed
- ---             ----     -------------  ---------------  ----------------------  --
↪------------------
sandbox          replica  2              2                1                       1
sandbox-primary  primary  1              1                1                       0

Instances status
Name       Current LSN  Replication role  Status  QoS         Manager Version  Node
- ---       -----------  ----------------  ------  ---         ---------------  ----
```

(次のページに続く)

(前のページからの続き)

```
sandbox-1  0/6053720    Primary           OK       BestEffort  1.30.0          k8s-eu-
↪worker
sandbox-2  0/6053720    Standby (async)   OK       BestEffort  1.30.0          k8s-eu-
↪worker2
sandbox-3  0/6053720    Standby (async)   OK       BestEffort  1.30.0          k8s-eu-
↪worker
```

追加の-v `kubectl cnp status sandbox -v -v` などを使用すると、PostgreSQL 構成、HBA 設定、および証明書を表示することもできます。

このコマンドは、 `yaml` および `json` 形式での出力もサポートしています。

プロモート

このコマンドの意味は、クラスター内のポッドをプライマリに `promote` することにより、メンテナンス作業を開始したり、クラスターでのスイッチオーバー状況をテストしたりできます

```
kubectl cnp promote cluster-example cluster-example-2
```

または、インスタンスノード番号を使用してプロモートせることができます

```
kubectl cnp promote cluster-example 2
```

証明書

EDB Postgres AI for CloudNativePG クラスターオペレーターを使用して作成されたクラスターは、CA と連携して TLS 認証証明書に署名します。

証明書を取得するには、資格情報を保存するシークレットの名前、クラスター名、およびこの証明書のユーザーを提供する必要があります

```
kubectl cnp certificate cluster-cert --cnp-cluster cluster-example --cnp-user appuser
```

シークレットが作成された後、 `kubectl` を使用して取得できます

```
kubectl get secret cluster-cert
```

また、次のコマンドを使用してプレーンテキストで同じ内容を取得します。

```
kubectl get secret cluster-cert -o json | jq -r .data | map(@base64d) | .[]
```

再起動

`kubectl cnp restart` コマンドは、次の 2 つの場合に使用できます。

- 特定のクラスターのロールアウト再起動を調整するようにオペレーターに要求します。これは、カスタム監視クエリを含む ConfigMap などのクラスター依存オブジェクトに構成変更を適用する場合に役立ちます。

- 単一インスタンスの再起動を要求します。インスタンスがクラスターのプライマリの場合はインプレース、またはレプリカの場合はポッドの削除と再作成。

```
#  this command will restart a whole cluster in a rollout fashion

kubectl cnp restart [clusterName]

#  this command will restart a single instance, according to the policy above

kubectl cnp restart [clusterName] [pod]
```

インプレース・リスタートが要求されたが、スイッチオーバーなしでは変更を適用できない場合、スイッチオーバーはインプレース・リスタートより優先されます。この一般的なケースは、PostgreSQL イメージのマイナーアップグレードです。

> **注釈**
>
> ConfigMaps と Secrets をインスタンスによって**自動的に**リロードしたい場合は、キー `k8s.enterprisedb.io/reload` を持つラベルをそれに追加できます。

リロード

`kubectl cnp reload` コマンドは、オペレーターに、特定のクラスターの調整ループをトリガーするように要求します。これは、カスタム監視クエリを含む ConfigMap などのクラスター依存オブジェクトに構成変更を適用する場合に役立ちます。

次のコマンドは、特定のクラスターのすべての構成をリロードします。

```
kubectl cnp reload [cluster_name]
```

メンテナンス

`kubectl cnp maintenance` コマンドは、名前空間全体で 1 つ以上のクラスターを変更し、メンテナンスウィンドウ値を設定するのに役立ちます。次のフィールドを変更します。

- .spec.nodeMaintenanceWindow.inProgress
- .spec.nodeMaintenanceWindow.reusePVC

これを使用して `set` および `unset` を引数として受け入れ、`set` の場合は `inProgress` を `true` に設定し、`unset` の場合は `false` に設定します。

デフォルトでは、 `--reusePVC` フラグが渡されない限り、 `reusePVC` は常に `false` に設定されます。

プラグインは、変更するクラスターのリストとその新しい値の確認を求めます。これが受け入れられる場合、このアクションはリスト内のすべてのクラスターに適用されます。

Kubernetes クラスター内のすべての PostgreSQL をメンテナンスに設定する場合は、次のコマンドを記述するだけです。

```
kubectl cnp maintenance set --all-namespaces
```

更新するすべてのクラスターのリストが得られます

```
The following are the new values for the clusters
Namespace  Cluster Name     Maintenance  reusePVC
- --------  ------------     -----------  --------
default    cluster-example  true         false
default    pg-backup        true         false
test       cluster-example  true         false
Do you want to proceed? [y/n]: y
```

レポート

`kubectl cnp report` コマンドは、さまざまな情報を ZIP ファイルにバンドルします。実稼働環境のクラスターの問題をデバッグするために必要なコンテキストを提供することを目的としています。

`operator` および `cluster` の 2 つのサブコマンドがあります。

レポート演算子

`operator` サブコマンドは、オペレーターの展開、構成、およびイベントに関する情報の提供をオペレーターに要求します。

> **注釈**
>
> デフォルトでは、オペレーターログは収集されませんが、 *--logs* フラグを使用してオペレーターログ収集を有効にできます

- **展開情報** オペレーター展開とオペレーターポッド
- **configuration** オペレーター名前空間のシークレットと ConfigMap
- **events** オペレーター名前空間のイベント
- **webhook configuration** Webhook 構成の変更および検証
- **webhook service** Webhook サービス
- **logs** JSON 行形式のオペレーターポッドのログオプショナル、デフォルトではオフ

このコマンドは、YAML 形式のさまざまなマニフェストを含む ZIP ファイルを生成しますデフォルトでは、`-o` フラグを使用して JSON に設定可能です。 `-f` フラグを使用して、結果ファイルに明示的に名前を付けます。 `-f` フラグを使用しない場合、zip ファイルにデフォルトのタイムスタンプ付きファイル名が作成されます。

> **注釈**
>
> Report プラグインは `kubectl` 規約に従い、名前空間によって制限されたオブジェクトを検索します。{{name.short}}オペレーターは、通常、クラスターと同じ名前空間にインストールされません。例デフォルトのインストール名前空間は postgresql-operator-system です

```
kubectl cnp report operator -n <namespace>
```

での結果

```
Successfully written report to "report_operator_<TIMESTAMP>.zip" (format: "yaml")
```

-f フラグを設定した場合

```
kubectl cnp report operator -n <namespace> -f reportRedacted.zip
```

ファイルを解凍すると、タイムスタンプ付きのトップレベルフォルダーが生成され、ディレクトリを整理します。

```
unzip reportRedacted.zip
```

結果は次のとおりです。

```
Archive:  reportRedacted.zip
   creating: report_operator_<TIMESTAMP>/
   creating: report_operator_<TIMESTAMP>/manifests/
  inflating: report_operator_<TIMESTAMP>/manifests/deployment.yaml
  inflating: report_operator_<TIMESTAMP>/manifests/operator-pod.yaml
  inflating: report_operator_<TIMESTAMP>/manifests/events.yaml
  inflating: report_operator_<TIMESTAMP>/manifests/validating-webhook-configuration.
↪yaml
  inflating: report_operator_<TIMESTAMP>/manifests/mutating-webhook-configuration.yaml
  inflating: report_operator_<TIMESTAMP>/manifests/webhook-service.yaml
  inflating: report_operator_<TIMESTAMP>/manifests/postgresql-operator-ca-secret.yaml
  inflating: report_operator_<TIMESTAMP>/manifests/postgresql-operator-webhook-cert.
↪yaml
```

--logs オプションをアクティブにすると、追加のサブディレクトリが表示されます。

```
Archive:  report_operator_<TIMESTAMP>.zip
  <snipped ···>
  creating: report_operator_<TIMESTAMP>/operator-logs/
  inflating: report_operator_<TIMESTAMP>/operator-logs/postgresql-operator-controller-
↪manager-66fb98dbc5-pxkmh-logs.jsonl
```

注釈

プラグインは、PREVIOUS オペレーターのログを取得しようとします。これは、再起動されたオペレーターを調査するときに役立ちます。すべての場合、現在のオペレーターログも取得しようとします。現在と以前のログが利用可能な場合、両方が表示されます。

```
====== Begin of Previous Log =====
2023-03-28T12:56:41.251711811Z {"level":"info","ts":"2023-03-28T12:56:41Z","logger":
↪"setup","msg":"Starting EDB Postgres for Kubernetes Operator","version":"1.30.0",
↪"build":{"Version":"1.30.0+dev107","Commit":"cc9bab17","Date":"2023-03-28"}}
2023-03-28T12:56:41.251851909Z {"level":"info","ts":"2023-03-28T12:56:41Z","logger":
↪"setup","msg":"Starting pprof HTTP server","addr":"0.0.0.0:6060"}
  <snipped ···>

====== End of Previous Log =====
2023-03-28T12:57:09.854306024Z {"level":"info","ts":"2023-03-28T12:57:09Z","logger":
↪"setup","msg":"Starting EDB Postgres for Kubernetes Operator","version":"1.30.0",
↪"build":{"Version":"1.30.0+dev107","Commit":"cc9bab17","Date":"2023-03-28"}}
2023-03-28T12:57:09.854363943Z {"level":"info","ts":"2023-03-28T12:57:09Z","logger":
↪"setup","msg":"Starting pprof HTTP server","addr":"0.0.0.0:6060"}
```

オペレーターが再起動されていない場合、`====== Begin ···`および`====== End ···`ガードが表示されますが、中にコンテンツはありません。

機密情報がデフォルトで編集されていることを確認できます。

```
cd report_operator_<TIMESTAMP>/manifests/
head postgresql-operator-ca-secret.yaml
```

```
data:
  ca.crt: ""
  ca.key: ""
metadata:
  creationTimestamp: "2022-03-22T10:42:28Z"
  managedFields:
    - apiVersion: v1
      fieldsType: FieldsV1
      fieldsV1:
```

`-S --stopRedaction`オプションを有効にすると、シークレットが表示されます。

```
kubectl cnp report operator -n <namespace> -f reportNonRedacted.zip -S
```

機密情報を表示しようとしていることを示すリマインダーが届きます。

```
WARNING: secret Redaction is OFF. Use it with caution
Successfully written report to "reportNonRedacted.zip" (format: "yaml")
```

```
unzip reportNonRedacted.zip
head postgresql-operator-ca-secret.yaml
```

```
data:
  ca.crt: LS0tLS1CRUdJTiBD···
  ca.key: LS0tLS1CRUdJTiBF···
metadata:
  creationTimestamp: "2022-03-22T10:42:28Z"
  managedFields:
    - apiVersion: v1
      fieldsType: FieldsV1
```

レポートクラスター

cluster サブコマンドは、以下を収集します。

- **クラスターリソース**クラスター情報、kubectl get cluster -o yaml と同じ
- **クラスターポッド**クラスター名と一致するクラスター名前空間内のポッド
- **クラスタージョブ**クラスター名と一致するクラスター名前空間内のジョブがある場合
- **events** クラスター名前空間のイベント
- **pod logs** JSON 行形式のクラスターポッドのログオプショナル、デフォルトでオフ
- **ジョブログ**ジョブによって作成されたポッドのログオプショナル、デフォルトでオフ JSON 行形式で

cluster サブコマンドは、 operator が行うように、-f および-o フラグを受け入れます。 -f フラグを使用しない場合、デフォルトのタイムスタンプ付きレポート名が使用されます。クラスター情報には構成シークレット/構成マップが含まれていないため、-S は無効になっていることに注意してください。

> **注釈**
>
> デフォルトでは、クラスターログは収集されませんが、 *--logs* フラグを使用してクラスターログ収集を有効にできます

使用法

```
kubectl cnp report cluster <clusterName> [flags]
```

operator サブコマンドとは異なり、 cluster サブコマンドの場合、クラスターがデフォルトの場合を除き、クラスター名前と、多くの場合、名前空間を指定する必要があることに注意してください。

```
kubectl cnp report cluster example -f report.zip -n example_namespace
```

そして

```
unzip report.zip
```

```
Archive:  report.zip
   creating: report_cluster_example_<TIMESTAMP>/
   creating: report_cluster_example_<TIMESTAMP>/manifests/
  inflating: report_cluster_example_<TIMESTAMP>/manifests/cluster.yaml
  inflating: report_cluster_example_<TIMESTAMP>/manifests/cluster-pods.yaml
  inflating: report_cluster_example_<TIMESTAMP>/manifests/cluster-jobs.yaml
  inflating: report_cluster_example_<TIMESTAMP>/manifests/events.yaml
```

--logs フラグを使用して、ポッドとジョブログを ZIP に追加できることに注意してください。

```
kubectl cnp report cluster example -n example_namespace --logs
```

結果は次のとおりです。

```
Successfully written report to "report_cluster_example_<TIMESTAMP>.zip" (format: "yaml
↪")
```

```
unzip report_cluster_<TIMESTAMP>.zip
```

```
Archive:  report_cluster_example_<TIMESTAMP>.zip
   creating: report_cluster_example_<TIMESTAMP>/
   creating: report_cluster_example_<TIMESTAMP>/manifests/
  inflating: report_cluster_example_<TIMESTAMP>/manifests/cluster.yaml
  inflating: report_cluster_example_<TIMESTAMP>/manifests/cluster-pods.yaml
  inflating: report_cluster_example_<TIMESTAMP>/manifests/cluster-jobs.yaml
  inflating: report_cluster_example_<TIMESTAMP>/manifests/events.yaml
   creating: report_cluster_example_<TIMESTAMP>/logs/
  inflating: report_cluster_example_<TIMESTAMP>/logs/cluster-example-full-1.jsonl
   creating: report_cluster_example_<TIMESTAMP>/job-logs/
  inflating: report_cluster_example_<TIMESTAMP>/job-logs/cluster-example-full-1-
↪initdb-qnnvw.jsonl
  inflating: report_cluster_example_<TIMESTAMP>/job-logs/cluster-example-full-2-join-
↪tvj8r.jsonl
```

OpenShift のサポート

report operator 指示子は、クラスターが OpenShift で実行されているかどうかを自動的に検出し、クラスターサービスバージョンとインストールプランを取得して、openshift サブフォルダーの下の zip に自動的に追加します。

> **注釈**
>
> OpenShift では名前空間が非常に重要になります。 CNP の OpenShift のデフォルトの名前空間は「openshift-operators」です。多くのほとんどのクライアントは、CNP オペレーターに別の名前空間を使用します。

```
kubectl cnp report operator -n openshift-operators
```

での結果

```
Successfully written report to "report_operator_<TIMESTAMP>.zip" (format: "yaml")
```

OpenShift 関連のファイルは `openshift` サブフォルダーにあります。

```
unzip report_operator_<TIMESTAMP>.zip
cd report_operator_<TIMESTAMP>/
cd openshift
head clusterserviceversions.yaml
```

```
apiVersion: operators.coreos.com/v1alpha1
items:
  - apiVersion: operators.coreos.com/v1alpha1
    kind: ClusterServiceVersion
    metadata:
      annotations:
        alm-examples: |-
          [
            {
              "apiVersion": "postgresql.k8s.enterprisedb.io/v1",
```

ログ

`kubectl cnp logs` コマンドを使用すると、CloudNativePG クラスターの EDB Postgres AI に関連するポッドのコレクションのログを一度にたどることができます。

現時点では、使用可能なサブコマンドが 1 つあります `cluster` 。

クラスターログ

`cluster` サブコマンドは、クラスターのすべてのポッドログを単一のストリームまたはファイルに収集します。これは、単一のターミナルウィンドウで、コマンドを 1 回呼び出して、すべてのポッドログを取得できることを意味します。

すべての cnp プラグインサブコマンドと同様に、`-h` フラグの指示とヘルプを取得できます。

```
kubectl cnp logs cluster -h
```

logs コマンドは、 --timestamps フラグを使用しない限り、JSON 行形式でログを表示します。その場合、人間が判読可能なタイムスタンプが各行の先頭に追加されます。この場合、行は有効な JSON でなくなり、jq などのツールが目的のように動作しない場合があります。

logs cluster サブコマンドに-f フラグ別名--follow が指定されている場合、クラスターポッドログをたどり、コマンドが呼び出された後にクラスターで作成された新しいポッドを監視します。再起動または再作成されたポッドを含む、見つかった新しいポッドもポッドが追跡されます。ログは端末の標準出力に表示されます。このコマンドは、クラスターにポッドが残っていない場合、またはユーザーによって中断された場合にのみ終了します。

-f オプションを指定せずに logs が呼び出された場合、呼び出されるまですべてのクラスターポッドからログを読み取り、ターミナルの標準出力に表示してから終了します。-o または--output フラグを提供して、標準出力で表示する代わりに、ログを保存するファイルの名前を指定できます。--tail フラグを使用して、クラスター内の各ポッドから取得するログ行数を指定できます。デフォルトでは、logs cluster サブコマンドは、クラスター内の各ポッドからのすべてのログを表示します。「follow」フラグ-f と組み合わせると、--tail で指定された数のログが現在まで取得され、それ以降新しいログが追跡されます。

注他の cnp プラグインコマンドとは異なり、 -f はファイルを指定するのではなく「フォロー」を示すために使用されます。これは、kubectl logs の規則に従いますが、 -f は、ログに従う必要があることを意味します。

使用法

```
kubectl cnp logs cluster <clusterName> [flags]
```

-f オプションを使用して次のことを行います。

```
kubectl cnp report cluster cluster-example -f
```

--tail オプションを使用して各ポッドから 3 行を表示し、 -f オプションを追跡します。

```
kubectl cnp report cluster cluster-example -f --tail 3
```

```
{"level":"info","ts":"2023-06-30T13:37:33Z","logger":"postgres","msg":"2023-06-30␣
↪13:37:33.142 UTC [26] LOG:  ending log output to stderr","source":"/controller/log/
↪postgres","logging_pod":"cluster-example-3"}
{"level":"info","ts":"2023-06-30T13:37:33Z","logger":"postgres","msg":"2023-06-30␣
↪13:37:33.142 UTC [26] HINT:  Future log output will go to log destination \"csvlog\
↪".","source":"/controller/log/postgres","logging_pod":"cluster-example-3"}
...
...
```

-o オプションを省略し、--output を指定する場合

```
kubectl cnp logs cluster cluster-example --output my-cluster.log
```

```
Successfully written logs to "my-cluster.log"
```

かなり

pretty サブコマンドは、標準入力からログストリームを読み取り、人間が判読可能な出力にフォーマットし、タイムスタンプでエントリを並べ替えようとします。

次の例に示すように、kubectl cnp logs cluster と組み合わせて使用できます。

```
kubectl cnp logs cluster cluster-example | kubectl cnp logs pretty
```

```
2024-10-15T17:35:00.336 INFO     cluster-example-1 instance-manager Starting EDB␣
↪Postgres AI for CloudNativePG Cluster Instance Manager
2024-10-15T17:35:00.336 INFO     cluster-example-1 instance-manager Checking for free␣
↪disk space for WALs before starting PostgreSQL
2024-10-15T17:35:00.347 INFO     cluster-example-1 instance-manager starting␣
↪tablespace manager
2024-10-15T17:35:00.347 INFO     cluster-example-1 instance-manager starting external␣
↪server manager
[...]
```

または、次の例のように、stern 、kubectl logs などの JSON 形式で{{name.abbr}}ログを生成する他のコマンドと組み合わせて使用できます。

```
kubectl logs cluster-example-1 | kubectl cnp logs pretty
```

```
2024-10-15T17:35:00.336 INFO     cluster-example-1 instance-manager Starting EDB␣
↪Postgres AI for CloudNativePG Cluster Instance Manager
2024-10-15T17:35:00.336 INFO     cluster-example-1 instance-manager Checking for free␣
↪disk space for WALs before starting PostgreSQL
2024-10-15T17:35:00.347 INFO     cluster-example-1 instance-manager starting␣
↪tablespace manager
2024-10-15T17:35:00.347 INFO     cluster-example-1 instance-manager starting external␣
↪server manager
[...]
```

pretty サブコマンドは、高度なログフィルタリングもサポートしているため、ユーザーは特定のポッドまたはロガーのログを表示したり、重大度レベルでログをフィルタリングしたりできます。次に例を示します。

```
kubectl cnp logs cluster cluster-example | kubectl cnp logs pretty --pods cluster-
↪example-1 --loggers postgres --log-level info
```

```
2024-10-15T17:35:00.509 INFO     cluster-example-1 postgres         2024-10-15␣
↪17:35:00.509 UTC [29] LOG:  redirecting log output to logging collector process
2024-10-15T17:35:00.509 INFO     cluster-example-1 postgres         2024-10-15␣
↪17:35:00.509 UTC [29] HINT:  Future log output will appear in directory "/
↪controller/log"...
2024-10-15T17:35:00.510 INFO     cluster-example-1 postgres         2024-10-15␣
```

(次のページに続く)

(前のページからの続き)

```
↪17:35:00.509 UTC [29] LOG:  ending log output to stderr
2024-10-15T17:35:00.510 INFO     cluster-example-1 postgres         ending log output␣
↪to stderr
[...]
```

pretty サブコマンドは、ログを推論しやすくするために、ログストリームをソートしようとします。これを行うために、ログをグループに収集し、グループ内でタイムスタンプで並べ替えます。pretty は「follow」モードのコマンドからパイプされる場合があるため、これは対話型で並べ替える唯一の方法です。サブコマンドは、ソートされた各グループの最後にグループ区切り線--- を追加します。次の例に示すように、グループのサイズは--sorting-group-size フラグデフォルト 1000 を介して構成できます。

```
kubectl cnp logs cluster cluster-example | kubectl cnp logs pretty --sorting-group-
↪size=3
```

```
2024-10-15T17:35:20.426 INFO     cluster-example-2 instance-manager Starting EDB␣
↪Postgres AI for CloudNativePG Cluster Instance Manager
2024-10-15T17:35:20.426 INFO     cluster-example-2 instance-manager Checking for free␣
↪disk space for WALs before starting PostgreSQL
2024-10-15T17:35:20.438 INFO     cluster-example-2 instance-manager starting␣
↪tablespace manager
- --
2024-10-15T17:35:20.438 INFO     cluster-example-2 instance-manager starting external␣
↪server manager
2024-10-15T17:35:20.438 INFO     cluster-example-2 instance-manager starting␣
↪controller-runtime manager
2024-10-15T17:35:20.439 INFO     cluster-example-2 instance-manager Starting␣
↪EventSource
- --
[...]
```

使用可能なすべてのオプションを確認するには、 -h フラグを使用して、サポートされているフラグとその使用法の詳細を確認します。

> **注釈**
>
> -v オプションを追加して、ログの詳細度を高めることもできます。

デストロイ

kubectl cnp destroy コマンドは、インスタンスと、関連するすべての PVC を Kubernetes クラスターから削除するのに役立ちます。

オプションの--keep-pvc フラグを指定すると、PVC を保持したまま、インスタンスによって設定されたすべての metadata.ownerReferences を削除できます。さらに、PVC の k8s.enterprisedb.io/pvcStatus ラベルが ready から detached に変更され、使用されなくなったことを示します。

--keep-pvc フラグを指定せずにコマンドを再度実行すると、分離された PVC が削除されます。

使用法

```
kubectl cnp destroy [CLUSTER_NAME] [INSTANCE_ID]
```

次の例は、cluster-example-2 ポッドと関連する PVC を削除します。

```
kubectl cnp destroy cluster-example 2
```

クラスターハイバーネーション

データを保持しながら CloudNativePG クラスター Cluster の EDB Postgres AI の実行を一時停止し、後でアクティビティを再開することが必要になる場合があります。この機能を**クラスターハイバーネーション**と呼びます。

Hibernation は、 kubectl cnp hibernate [on|off] コマンドを介してのみ使用できます。

CloudNativePG クラスタークラスターの EDB Postgres AI を休止することは、PostgreSQL プライマリインスタンスに属する PVC を除き、クラスターによって生成されたすべてのリソースを破棄することを意味します。

次の方法でクラスターを休止状態にできます。

```
kubectl cnp hibernate on <cluster-name>
```

これにより、

1. すべての PostgreSQL インスタンスをシャットダウンする
2. プライマリインスタンスのデータを含む PVC を切断し、最新のデータベースステータスと最新のクラスター構成でアノテーションを付けます。
3. 生成されたすべてのリソースを含む Cluster リソースを削除します-前述の PVC を除く

冬眠している場合、CloudNativePG クラスタークラスターの EDB Postgres AI は、PVC のグループだけで表され、PGDATA を含むものには、pg_controldata のコンテンツを含む最新の利用可能なステータスで注釈が付けられます。

> **警告**
>
> フェンシングもハイバーネーションプロシージャの一部であるため、フェンスされたインスタンスを持つクラスターは冬眠できません。

エラーが発生した場合、オペレーターはプロシージャーを元に戻すことはできません。次のコマンドを使用して操作を強制できます。

```
kubectl cnp hibernate on cluster-example --force
```

休止状態のクラスターは、次の方法で再開できます。

```
kubectl cnp hibernate off <cluster-name>
```

クラスターが休止状態になると、最後の構成と、シャットダウン後に PostgreSQL のステータスを表示できます。それは次の方法で行うことができます。

```
kubectl cnp status <cluster-name>
```

pgbench を使用したデータベースのベンチマーク

Pgbench は、次のコマンドを使用して、既存の PostgreSQL クラスターに対して実行できます。

```
kubectl cnp pgbench <cluster-name> -- --time 30 --client 1 --jobs 1
```

詳細については、 *Benchmarking pgbench section* を参照してください。

fio を使用したストレージのベンチマーク

fio は、次のコマンドを使用して既存のストレージクラスで実行できます。

```
kubectl cnp fio <fio-job-name> -n <namespace>
```

詳細については、 *Benchmarking fio section* を参照してください。

新しい物理バックアップの要求

`kubectl cnp backup` コマンドは、新しい `Backup` リソースを作成することにより、既存の Postgres クラスターの新しい物理バックアップを要求します。

> **注釈**
>
> リリース 1.21 から、 *backup* コマンドは、新しいフラグ `-m` を受け入れて、バックアップ方法を指定します。ボリュームスナップショットを使用してバックアップを要求するには、 *-m volumeSnapshot* を設定します

次の例は、特定のクラスターのオンデマンドバックアップを要求します。

```
kubectl cnp backup [cluster_name]
```

または、ボリュームスナップショットを使用する場合

```
kubectl cnp backup [cluster_name] -m volumeSnapshot
```

作成されるバックアップには、要求時間に基づいた名前が付けられます。

```
kubectl cnp backup cluster-example
```

```
backup/cluster-example-20230121002300 created
```

デフォルトでは、新しく作成されるバックアップは、クラスターで定義されたバックアップターゲットポリシーを使用して、実行するインスタンスを選択します。ただし、このポリシーは--backup-target オプションでオーバーライドできます。

ボリュームスナップショットバックアップの場合、 --online オプションを使用して、オンライン/ホットバックアップまたはオフライン/コールドバックアップを要求することもできます。さらに、--immediate-checkpoint および--wait-for-archive オプションを明示的に設定することにより、オンラインバックアップを調整することもできます。

Appendix B - Backup on object stores には、構成設定に関する詳細情報が含まれています。

psql の起動

kubectl cnp psql コマンドは、実際のポッドから実行しているかのように、既存の Postgres クラスターに接続された新しい PostgreSQL 対話型フロントエンドプロセス psql を起動します。これは、 postgres ユーザーを使用することを意味します。

```
kubectl cnp psql cluster-example
```

```
psql (17.0 (Debian 17.0-1.pgdg110+1))
Type "help" for help.

postgres=#
```

デフォルトでは、コマンドはプライマリインスタンスに接続します。ユーザーは--replica オプションを使用して、レプリカに対して作業することを選択できます。

```
kubectl cnp psql --replica cluster-example
```

```
psql (17.0 (Debian 17.0-1.pgdg110+1))

Type "help" for help.

postgres=# select pg_is_in_recovery();
 pg_is_in_recovery
- ------------------
 t
(1 row)

postgres=# \q
```

このコマンドは kubectl exec を起動します。正常に動作するには、PATH 変数で kubectl 実行可能ファイルに到達できる必要があります。

注釈

OpenShift で実行されているインスタンスに接続する場合、

security measure built intoOpenShift のため、ユーザー名を `psql` コマンドに明示的に渡す必要があります。

```
kubectl cnp psql cluster-example -- -U postgres
```

警告

kubectl cnp snapshot コマンドは削除されました。 **新しい物理バックアップの要求** を使用して、ボリュームスナップショットを使用したバックアップを要求してください。

評価/デモ目的でのみ pgAdmin4 を使用する

pgAdmin は、PostgreSQL の最も人気のある機能が豊富なオープンソースの管理および開発プラットフォームです。プロジェクトの詳細については、公式 documentation を参照してください。

pgAdmin 開発チームが公式の Docker コンテナイメージを維持していることを考えると、標準の Kubernetes 展開として環境に pgAdmin をインストールできます。

ただし、**デモと評価の目的で** 、CloudNativePG クラスターの EDB Postgres AI は適切なソリューションを提供します。 `cnp` プラグインは `pgadmin4` コマンドを実装し、特定のデータベース `Cluster` に接続し、`kind` などのローカル環境でそのコンテンツをナビゲートする簡単な方法を提供します。

たとえば、次のとおりに `cluster-example` クラスターの pgAdmin4 のデモ展開をインストールできます。

```
kubectl cnp pgadmin4 cluster-example
```

このコマンドは次を生成します。

```
ConfigMap/cluster-example-pgadmin4 created
Deployment/cluster-example-pgadmin4 created
Service/cluster-example-pgadmin4 created
Secret/cluster-example-pgadmin4 created

[...]
```

pgAdmin を展開した後、kubectl を使用してポートを転送し、画面の指示に従ってブラウザーを介して接続します。

いつものように、`--dry-run` オプションを使用して YAML ファイルを生成できます。

```
kubectl cnp pgadmin4 --dry-run cluster-example
```

pgAdmin4 はデスクトップまたはサーバーモードでインストールできます。デフォルトはサーバーです。

`server` モードでは、ランダムに生成されたパスワードを使用して認証が必要で、ユーザーは接続するデータベースを手動で指定する必要があります。

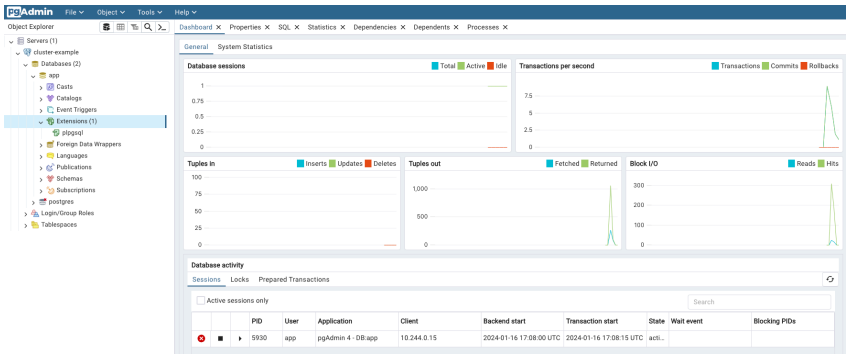


図 4: Screenshot of desktop installation of pgAdmin

一方、desktop モードは、認証を必要とせずに pgAdmin Web インターフェイスを開始します。 app ユーザーとして app データベースに自動的に接続するため、 kind を使用したローカル展開などの簡単なデモに最適です。

```
kubectl cnp pgadmin4 --mode desktop cluster-example
```

デモが終了した後、次のコマンドを実行して pgAdmin 展開を確実に終了します。

```
kubectl cnp pgadmin4 --dry-run cluster-example | kubectl delete -f -
```

> **警告**
>
> プラグインを使用して pgAdmin を運用環境に展開しないでください。

ロジカルレプリケーションのパブリケーション

cnp publication コマンドグループは、

PostgreSQL logical replication publications の作成と削除を合理化するように設計されています。これらのコマンドは、主に、特にリモートの PostgreSQL データベースでの論理レプリケーションパブリケーションの作成を支援することを目的としていることに注意してください。

> **警告**
>
> これらのコマンドを使用する前に、PostgreSQL のネイティブ論理レプリケーションシステムの機能と制限の両方をしっかりと理解することが重要です。特に、

logical replication restrictions に注意してください。

新しいパブリケーションの作成

論理レプリケーションパブリケーションを作成するには、cnp publication create コマンドを使用します。このコマンドの基本的な構造は次のとおりです。

```
kubectl cnp publication create \
  --publication <PUBLICATION_NAME> \
  [--external-cluster <EXTERNAL_CLUSTER>]
  <LOCAL_CLUSTER> [options]
```

2 つの主な使用例があります。

- --external-cluster の場合このオプションを使用して、外部クラスターつまり externalClusters スタンザで定義された上にパブリケーションを作成します。コマンドは<LOCAL_CLUSTER> から発行されますが、パブリケーションは<EXTERNAL_CLUSTER> のデータ用です。
- --external-cluster なし このオプションを使用して、<LOCAL_CLUSTER> PostgreSQL Cluster デフォルトでは app データベースにパブリケーションを作成します。

> **警告**
>
> 外部クラスターに接続するときは、指定されたユーザーに`CREATE PUBLICATION` コマンドを実行するための十分な権限があることを確認してください。

CREATE PUBLICATION と同様に、いくつかのオプションがあります

コマンドを使用して、レプリケートするテーブルのグループを定義します。重要なオプションには次のものが含まれます。

- --all-tables オプションを指定した場合、パブリケーション FOR ALL TABLES を作成します。
- または、次の複数のオカレンスを指定できます。
- --table 特定のテーブル式を追加します。
- --schema 指定されたデータベーススキーマ PostgreSQL 15 から利用可能にあるすべてのテーブルを含めます。

--dry-run オプションを使用すると、プラグインが実行する SQL コマンドをプレビューできます。

追加情報と詳細な手順については、次のコマンドを入力してください。

```
kubectl cnp publication create --help
```

例

source-cluster および destination-cluster を指定して、source-cluster 上のデータのパブリケーションを作成します。destination-cluster には、source-cluster を指す externalClusters スタンザにエントリーがあります。

次を実行できます。

```
kubectl cnp publication create destination-cluster  \
  --external-cluster=source-cluster --all-tables
```

これは、 destination-cluster で SQL コマンドを実行し、source-cluster のすべてのテーブルのパブリケーションを作成します。

または、その代わりに、次を実行できます。

```
kubectl cnp publication create source-cluster \
  --publication=app --all-tables
```

これは、 source-cluster 内のすべてのテーブルに app という名前のパブリケーションを作成し、ソースクラスターで SQL コマンドを実行します。

注釈

イラストとインスピレーションのために提供されている 2 つのサンプルファイル logical-source および logical-destination があります。

パブリケーションの削除

cnp publication drop コマンドは、パブリケーション名、クラスター名、オプションの外部クラスターを含む同様のキーオプションを提供することにより、create コマンドをシームレスに補完します。次のコマンド構造を使用して PUBLICATION をドロップできます。

```
kubectl cnp publication drop \
  --publication <PUBLICATION_NAME> \
  [--external-cluster <EXTERNAL_CLUSTER>]
  <LOCAL_CLUSTER> [options]
```

さらに詳細と正確な指示にアクセスするには、次のコマンドを使用します。

```
kubectl cnp publication drop --help
```

論理レプリケーションサブスクリプション

cnp subscription コマンドグループは、

PostgreSQL logical replication subscriptions の作成と削除を簡素化するように設計された専用のコマンドセットです。これらのコマンドは、特にリモートの PostgreSQL データベースを扱う場合、論理レプリケーションサブスクリプションの確立を支援するように特別に作成されています。

警告

これらのコマンドを使用する前に、PostgreSQL のネイティブ論理レプリケーションシステムの機能と制限の両方を包括的に理解することが重要です。特に、

logical replication restrictions に注意してください。

サブスクリプション管理に加えて、ソースクラスターからすべてのシーケンスを同期するための役立つコマンドを提供します。その適用性は異なる場合がありますが、このコマンドは、メジャーアップグレードまたはリモートサーバーからのデータのインポートを含むシナリオで特に役立ちます。

新しいサブスクリプションの作成

論理レプリケーションサブスクリプションを作成するには、`cnp subscription create` コマンドを使用します。このコマンドの基本的な構造は次のとおりです。

```
kubectl cnp subscription create \
  --subscription <SUBSCRIPTION_NAME> \
  --publication <PUBLICATION_NAME> \
  --external-cluster <EXTERNAL_CLUSTER> \
  <LOCAL_CLUSTER> [options]
```

このコマンドは、 `<LOCAL_CLUSTER>` の `externalClusters` スタンザで定義されているとおりに、指定された外部クラスター内の指定されたパブリケーションを対象としたサブスクリプションを構成します。

追加情報と詳細な手順については、次のコマンドを入力してください。

```
kubectl cnp subscription create --help
```

例

パブリケーションに関するセクションと同様に、`source-cluster` および `destination-cluster` があり、`app` というパブリケーションを既に作成しています。

次のコマンド

```
kubectl cnp subscription create destination-cluster \
  --external-cluster=source-cluster \
  --publication=app --subscription=app
```

は、宛先クラスターに `app` のサブスクリプションを作成します。

警告

非実稼働環境でのテストサブスクリプションを優先して、実稼働設定に実装する前にその有効性を確認し、潜在的な問題を特定します。

注釈

イラストとインスピレーションのために提供されている 2 つのサンプルファイル logical-source および logical-destination 。

サブスクリプションの削除

cnp subscription drop コマンドは、create コマンドをシームレスに補完します。次のコマンド構造を使用して SUBSCRIPTION をドロップできます。

```
kubectl cnp subcription drop \
  --subscription <SUBSCRIPTION_NAME> \
  <LOCAL_CLUSTER> [options]
```

さらに詳細と正確な指示にアクセスするには、次のコマンドを使用します。

```
kubectl cnp subscription drop --help
```

シーケンスの同期

パブリケーションとサブスクリプションを介して実装される PostgreSQL 論理レプリケーションの重要な制約の 1 つは、シーケンス同期の欠如です。これは、論理レプリケーションを利用する場合に特に関連しますライブデータベース移行、特に PostgreSQL の上位バージョンの場合。このプロセスの重要な手順には、アプリケーションを新しいデータベース_cutover_に移行する前にシーケンスの更新が含まれます。

この制限に対処するために、cnp subscription sync-sequences コマンドは解決策を提供します。このコマンドは、ソースデータベースとの接続を確立し、関連するすべてのシーケンスを取得し、その後、一致する ID でローカルシーケンスを更新しますデータベーススキーマとシーケンス名に基づいて。

以下に示すようにコマンドを使用できます。

```
kubectl cnp subscription sync-sequences \
  --subscription <SUBSCRIPTION_NAME> \
  <LOCAL_CLUSTER>
```

包括的な詳細と特定の手順については、次のコマンドを使用します。

```
kubectl cnp subscription sync-sequences --help
```

例

パブリケーションとサブスクリプションに関する以前のセクションと同様に、source-cluster および destination-cluster があります。パブリケーションとサブスクリプションは、両方とも app と呼ばれますが、既に存在します。

次のコマンドは、app サブスクリプションに関係するシーケンスをソースクラスターから宛先クラスターに同期します。

```
kubectl cnp subscription sync-sequences destination-cluster \
  --subscription=app
```

> **警告**
>
> 非実稼働環境でのテストサブスクリプションを優先して、実稼働設定に展開する前にその有効性を保証し、潜在的な問題を検出します。

5.31.3 K9s との統合

cnp プラグインは、Kubernetes クラスターと対話するための人気のターミナルベースの UI である K9s に簡単に統合できます。

詳細は、 *k9s/plugins.yml* を参照してください。

5.31.4 プラグインに必要な権限

プラグインには、実行するコマンドに応じて一連の Kubernetes 権限が必要です。これらの権限は、ポッド、PDB、PVC などのリソースやサブリソースに影響を与え、`get` 、`delete` 、`patch` などのアクションを有効にする場合があります。次の表には、完全な詳細が含まれています。

Command	Resource Permissions
backup	clusters: get backups: create
certificate	clusters: get secrets: get,create
destroy	pods: get,delete jobs: delete,list PVCs: list,delete,update
fencing	clusters: get,patch pods: get
fio	PVCs: create configmaps: create deployment: create
hibernate	clusters: get,patch,delete pods: list,get,delete pods/exec: create jobs: list PVCs: get,list,update,patch,delete
install	none
logs	clusters: get pods: list pods/log: get
maintenance	clusters: get,patch,list
pgadmin4	clusters: get configmaps: create deployments: create services: create secrets: create
pgbench	clusters: get jobs: create
promote	clusters: get clusters/status: patch pods: get
psql	pods: get,list pods/exec: create
publication	clusters: get pods: get,list pods/exec: create
reload	clusters: get,patch
report cluster	clusters: get pods: list pods/log: get jobs: list events: list PVCs: list
report operator	configmaps: get deployments: get events: list pods: list pods/log: get secrets: get services: get mutatingwebhookconfigurations: list[^1] validatingwebhookconfigurations: list[^1] If OLM is present on the K8s cluster, also: clusterserviceversions: list installplans: list subscriptions: list
restart	clusters: get,patch pods: get,delete

次のページに続く

表 9 – 前のページからの続き

Command	Resource Permissions
status	clusters: get pods: list pods/exec: create pods/proxy: create PDBs: list
subscription	clusters: get pods: get,list pods/exec: create
version	none

[^1] 権限は、クラススコープの ClusterRole リソースです。

///脚注はここにあります///

さらに、`clusters` に `list` 権限を割り当てると、複数のコマンドのオートコンプリートが有効になります。

ロールの例

権限が制限されたロールを作成することが可能です。次の例では、クラスターログにのみアクセスするロールを作成します。

```
- --
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: cnp-log
rules:
  - verbs:
      - get
    apiGroups:
      - postgresql.k8s.enterprisedb.io
    resources:
      - clusters
  - verbs:
      - list
    apiGroups:
      - ""
    resources:
      - pods
  - verbs:
      - get
    apiGroups:
      - ""
    resources:
      - pods/log
```

次の例は、プラグインの `status` コマンドを使用してクラスターステータスを取得するために必要な最小限の権限を持つロールを示しています。

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: cnp-status
rules:
  - verbs:
      - get
    apiGroups:
      - postgresql.k8s.enterprisedb.io
    resources:
      - clusters
  - verbs:
      - list
    apiGroups:
      - ""
    resources:
      - pods
  - verbs:
      - create
    apiGroups:
      - ""
    resources:
      - pods/exec
  - verbs:
      - create
    apiGroups:
      - ""
    resources:
      - pods/proxy
  - verbs:
      - list
    apiGroups:
      - policy
    resources:
      - poddisruptionbudgets
```

5.32 Automated failover

プライマリで予期しないエラーが.spec.failoverDelay デフォルトでは 0 秒より長く発生した場合、クラスターは**フェールオーバーモード**になります。これは、たとえば次の場合に発生する可能性があります。

- プライマリポッドにディスク障害が発生した
- プライマリポッドが削除された
- プライマリの postgres コンテナに何らかの持続的な障害がある

フェールオーバーシナリオでは、プライマリが正常に動作していると想定できません。

上記のような場合の後、プライマリポッドの準備状況プローブは失敗し始めます。これは、コントローラの調整ループで取得されます。コントローラは、2 つの手順でフェールオーバープロセスを開始します。

1. 最初に、`TargetPrimary` を `pending` としてマークします。この状態の変更は、プライマリポッドを強制的にシャットダウンし、レプリカの WAL レシーバーが停止します。クラスターは、フェールオーバーフェーズ「フェールオーバー」でマークされます。

2. すべての WAL レシーバーが停止すると、リーダー選挙が行われ、新しい予備選挙が名前付けられます。選択されたインスタンスはプライマリへの昇格を開始し、これが完了すると、クラスターは通常のオペレーションを再開します。一方、以前のプライマリポッドは再起動し、プライマリでなくなったことを検出し、レプリカノードになります。

障害が発生したプライマリがシャットダウンされている間

1. 最初に`.spec.switchoverDelay` 秒をタイムアウトとして PostgreSQL の_fast shutdown_を試行します。この正常なシャットダウンは、保留中の WAL をアーカイブしようとします。

2. 高速シャットダウンが失敗するか、そのタイムアウトを超過すると、PostgreSQL の_即時シャットダウン_が開始されます。

> **注釈**
>
> 「Fast」モードは、PostgreSQL クライアントが切断されるのを待機せず、進行中のオンラインバックアップを終了します。アクティブなトランザクションはすべてロールバックされ、クライアントは強制的に切断され、サーバーはシャットダウンされます。 「Immediate」モードは、クリーンシャットダウンせずに、すべての PostgreSQL サーバープロセスをすぐに中止します。

5.32.1 安全な予備選挙

任意の時点で、最大 1 つのインスタンスがプライマリに昇格するように、CloudNativePG クラスターの EDB Postgres AI は、`Cluster` にちなんで名前付けられ、その名前空間に存在する Kubernetes `Lease` オブジェクトを使用して上記の選択をサポートします。インスタンスは、プライマリに昇格する前に、このリースを取得および保持する必要があります。クリーンなシャットダウンでは、元のプライマリはリースを解放するため、資格のあるレプリカは、リースの有効期限が切れるのを待たずに引き継ぐことができます。

プライマリリースが保護するもの

以前のプライマリからストリーミングするのではなく、アーカイブから WAL ファイルを読み取ることにより、レプリカが追いつくことを検討します。PostgreSQL は、アーカイブが次の予想セグメントで「ファイルが見つかりません」を返すとすぐに再生を停止し、WAL ストリームの終わりとして扱います。以前のプライマリがまだ WAL を持っているときにレプリカが昇格した場合、アーカイブが終了していないため、その信号はストリームの真の終了前に到着します。レプリカは、以前のプライマリが確認され、これらの書き込みより前の LSN で新しいタイムラインをフォークします。失われています。

リースは、以前のプライマリがリリースするまでプロモーションを保持します。クリーンシャットダウンでは、PostgreSQL が残りの WAL をフラッシュしてアーカイブした後にリリースが発生するため、引き継ぐレプリカは最終的な端でアーカイブを参照します。以前のプライマリがリースを解放できない場合クラッ

シュ、ノード障害、またはインスタンスマネージャー自分自身に到達不能そのパスでは、アーカイブが追いついていない可能性があり、以前のプライマリがアーカイブを完了していなかった書き込みは失われます。

一次分離チェックとの関係

リースは、Kubernetes API サーバーへの接続を失ったものの、まだ実行されているプライマリをフェンスしません。これが **一次分離チェックとの関係** チェックのジョブです。2 つのメカニズムは補完的であり、両方ともデフォルトで有効になっています。

- リースは、_premature_プロモーションを防止します。元のプライマリがまだリースを保持している間、レプリカは昇格できません。
- 分離チェックは、_isolate_プライマリが書き込みの受け入れを継続しないようにします。

両方を有効にしたままにします。分離チェックを無効にすると、リースだけがプライマリの安全性を担当し、元のプライマリが API サーバーに到達できないが、それ以外は正常な場合、リースだけではスプリットブレインを防ぐことはできません。

プライマリリースの検査

リースはクラスターの名前を共有するため、直接検査できます。`cluster-example` というクラスターの場合

```
kubectl get lease cluster-example
```

`HOLDER` 列は、現在のプライマリであるリースを現在保持しているポッドをレポートします。

```
NAME              HOLDER              AGE
cluster-example   cluster-example-1   5m
```

有効なリース期間や最後の更新時刻を含む全体像を得るには、次を使用します。

```
kubectl get lease cluster-example -o yaml
```

プライマリリースのチューニング

リースタイミングは`.spec.primaryLease` の下で公開され、ほとんどのクラスターに適した値にデフォルトで設定されます。これらは、基になる Kubernetes のリーダー選択パラメーターに直接マッピングされます。

Field	Default	Description
leaseDurationSeconds	*15*	別のインスタンスが取得するまでのリースの有効期間。
renewDeadlineSeconds	*10*	プライマリが断念するまでにリースの更新を再試行し続ける期間。
retryPeriodSeconds	*2*	非所有者がリースの取得または更新を再試行する頻度。

次のページに続く

表 10 – 前のページからの続き

Field	Default	Description
releasedLeaseDurationSeconds	*1*	プライマリがクリーンシャットダウンでリースを解放したときに書き込まれる TTL。

例、低速または一時的に到達不能の API サーバーに対するクラスターの耐性を高めるには

```
spec:
  primaryLease:
    leaseDurationSeconds: 60
    renewDeadlineSeconds: 40
    retryPeriodSeconds: 15
```

警告

フェールオーバータイミングへの影響を理解する場合にのみ、これらの値を調整します。間隔が長くなると、クラスターの一時的な API サーバーの使用不可に対する耐性が高まりますが、正当なプロモーションが遅くなります。アドミッション Webhook によって 2 つの不変式が適用されます。*leaseDurationSeconds* は`renewDeadlineSeconds` より大きい必要があり、*renewDeadlineSeconds* は`retryPeriodSeconds` と`1.2` より大きい必要があります。どちらも、基になる Kubernetes リーダー選挙の要件を反映しています。

フェイルオーバーにより、サービスが影響を受ける *RTO* および/またはデータが失われる **リリース候補の目的** が発生する場合があります。

1. プライマリに障害が発生し始め、コントローラがフェールオーバー手順を開始する前に、転送中のクエリ、WAL 書き込み、チェックポイントおよび同様の操作が失敗する場合があります。
2. fast shutdown コマンドが発行されると、クラスターは接続を受け入れなくなるため、サービスは影響を受けますが、データは失われません。
3. 高速シャットダウンが失敗した場合、即時シャットダウンは WAL 書き込みを含む保留中のプロセスを停止します。データが失われる可能性があります。
4. プライマリがシャットダウンしており、新しいプライマリがまだ起動していない間、クラスターはプライマリなしで動作するため、機能が低下しますが、データの損失はありません。

注釈

高速シャットダウンを制御するタイムアウトは、スイッチオーバーの場合と同様に`.spec.switchoverDelay` によって設定されます。高速シャットダウンの時間を長くすることは、RPO の観点からより安全ですが、通常のオペレーションへの復帰が遅延する可能性があり、RTO に悪影響を与えます。

警告

インスタンスマネージャーのインプレース更新 で既に述べたように

スイッチオーバープロセスを説明するとき、 .spec.switchoverDelay オプションは PostgreSQL データベースの RPO と RTO に影響を与えます。低い値に設定すると、RPO よりも RTO が優先される場合がありますが、クラスターレベルおよび/またはバックアップレベルでデータの損失が発生します。逆に、高い値に設定すると、スイッチオーバー中にアクティブなプライマリのないクラスターを長時間放置する間のデータ損失のリスクを取り除くことができます。

5.32.2 遅延フェイルオーバー

上記で予想されたように、 .spec.failoverDelay オプションを使用すると、プライマリが異常であると検出された後、フェイルオーバープロシージャの開始を秒単位で遅延できます。デフォルトでは、この設定は 0 に設定され、フェイルオーバー手順がすぐにトリガーされます。

場合によっては、新しいプライマリへのフェイルオーバーの方が、プライマリがオンラインに戻るのを待つよりも混乱を招く可能性があります。これは、複数の層が影響を受けるダウンストリーム論理サブスクライバー、またはフェールオーバーの実行時間が予想される停止よりも長い場合、ネットワークの中断に特に当てはまります。

フェールオーバーを遅延する新しい構成オプションを有効にすることにより、短期間のネットワークまたはノードの不安定性による時期尚早のフェールオーバーを防止するメカニズムを提供します。

5.32.3 ノードレベルの障害の検出

プライマリをホストするノードが到達不能になるたとえば、kubelet のクラッシュまたはノードと Kubernetes API サーバー間のネットワーク分割が原因で、オペレーターはポッドの Ready 条件を利用して、プライマリがもはや使用不可であると判断します。ノードが正常な間、kubelet は準備状況プローブからその状態を最新の状態に保ちます。ノードが報告を停止すると、Kubernetes ノードライフサイクルコントローラは、ノード Unknown を宣言するとすぐに条件を False に反転するものです。

在庫の kube-controller-manager 設定では、移行は--node-monitor-grace-period によって管理されます Kubernetes 1.29-1.31 のデフォルト 40s 、1.32 以降では 50s に引き上げられます。そのウィンドウの後、コントローラはノード Unknown をマークし、同じモニタリングパスで、は、そのノードのポッドごとにパッチを発行して、Ready 条件を反転します。実際には、オペレーターは、ノードが到達不能になってから約 **40〜55 秒**、プライマリの準備ができていないことを観察します猶予期間プラス最大 1 つの--node-monitor-period ポーリング、デフォルト 5s 。管理対象 Kubernetes ディストリビューション GKE、EKS、AKS は、これらの値を調整する場合があります。観察されたタイミングが一致しない場合は、プロバイダーのドキュメントを参照してください。その後、フェイルオーバー手順が開始されます.spec.failoverDelay によってさらにゲートされます。

Ready 条件フリップは、部分ゾーンまたは大規模クラスターの中断中に Pod _eviction_を調整するレートリミッター --node-eviction-rate 、--secondary-node-eviction-rate 、--unhealthy-zone-threshold の影響を受けません。オペレーターは、ゾーンまたはクラスター全体の状態に関係なく、コントローラーがパッチを発行するとすぐに条件反転に反応します。

Pod _eviction_到達不能ノードからの実際の削除は、別のメカニズムであり、node.kubernetes.io/unreachable NoExecute テイントデフォルトで 300s の tolerationSeconds によって駆動されます。そのタイマーは、オペレーターのフェイルオーバーの決定を支持しません。CloudNativePG クラスターの EDB Postgres AI は、Ready 条件が反転するとすぐに、新しいプライマリをプロモートします。その時点で、分離ノードの kubelet はすでに古い PostgreSQL コンテナをローカルに停止しています。デフォルトの.spec.probes.liveness.isolationCheck.enabled: true では、インスタンスマネージャーは API

サーバーにもクラスターの残りの部分にも到達できず、独自の liveness プローブに失敗し、kubelet は kills 約 3 つのプローブ期間~30s 内のコンテナ。完全な高可用性オペレーターによる正常なノードでの古いプライマリの再作成は、実際にポッドを削除するテイントベースのエビクションでゲートされています。

5.32.4 Failover Quorum クォーラムベースのフェールオーバー

フェールオーバークォーラムは、CloudNativePG クラスター管理の PostgreSQL クラスターの EDB Postgres AI でのフェールオーバーイベント中のデータの耐久性と安全性を強化するメカニズムです。

クォーラムベースのフェールオーバーにより、コントローラーは、レプリカのクォーラムの状態に基づいて、レプリカをプライマリにプロモートせるかどうかを決定します。これは、**同期レプリケーションを介したゼロデータ損失クラスター** およびデフォルトの自動フェイルオーバープロシージャが提供するものよりも強力なデータの耐久性が必要な場合に役立ちます。

同期レプリケーションが有効になっていない場合、レプリカは昇格時にプライマリよりも遅延する場合があるため、フェールオーバー中にある程度のデータ損失が予想され受け入れられます。

同期レプリケーションが有効になっている場合、保証は、必要なすべての同期スタンバイが WAL データを安全に受信したことが判明するまで、アプリケーションがトランザクションの成功したコミットの明示的な確認を受け取らないことです。これは、オペレーターが最新のレプリカをプロモーションできることを保証するのに十分ではありません。

たとえば、同期レプリケーションが `ANY 1 (...)` に設定されている 3 ノードクラスターでは、コミットが確認される前にデータがプライマリと 1 つのスタンバイに書き込まれます。プライマリとアラインメントされたスタンバイの両方が使用不可になる場合ネットワーク分割中など、残りのレプリカには最新のデータがない場合があります。それを昇格させると、アプリケーションがコミットしたと考えられる一部のデータが失われる可能性があります。

クォーラムベースのフェイルオーバーは、オペレーターがプロモートするインスタンスに同期コミットされたデータの存在を確認できる場合にのみフェイルオーバーが発生し、それ以外の場合は発生しないようにすることにより、このリスクに対処します。

この機能を使用すると、ユーザーはデータの耐久性とデータの可用性の間で好みのトレードオフを選択できます。

`.spec.postgresql.synchronous.failoverQuorum` フィールドを `true` に設定することにより、フェールオーバークォーラムを有効にできます。

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3

  postgresql:
    synchronous:
      method: any
      number: 1
      failoverQuorum: true
```

(次のページに続く)

(前のページからの続き)

```
  storage:
    size: 1Gi
```

下位互換性のため、古いアノテーション alpha.k8s.enterprisedb.io/failoverQuorum は引き続きアドミッション Webhook でサポートされ、 Cluster 仕様オプションより優先されます。

- アノテーションが"true" と評価され、同期レプリケーションスタンザが存在する場合、Webhook は自動的に.spec.postgresql.synchronous.failoverQuorum を true に設定します。
- アノテーションが"false" と評価された場合、機能は常に無効になります

仕組み

レプリカをプライマリに昇格する前に、オペレーターは Dynamo R + W > N 一貫性モデル [^1] の原則に従って、クォーラムチェックを実行します。

クォーラムフェイルオーバーでは、これらの値は次の意味を想定しています。

- R は、_プロモテーブルレプリカ_読み取りクォーラムの数です。
- W は、 COMMIT がクライアント書き込みクォーラムに返される前に、書き込みを確認する必要があるレプリカの数です。
- N は、同期している可能性のあるレプリカの合計数です。

_プロモーブルレプリカ_は、次のプロパティを持つレプリカです。

- クラスターの一部です。
- 自分の状態をオペレーターに報告できます。
- 潜在的に同期しています。

R + W > N の場合、プロモートブルレプリカの中に、すべての同期コミットを確認した少なくとも 1 つがあることを確認でき、安全にプライマリに昇格できます。これに当てはまらない場合、コントローラはレプリカをプライマリにプロモートせず、状況が変化するまで待機します。

クォーラムチェックが失敗した場合でも、ユーザーは kubectl cnp promote コマンドを使用してレプリカをプライマリに強制的にプロモーションできます。

> **警告**
>
> 手動プロモーションは最後の手段としてのみ使用してください。続行する前に、データ損失のリスクを十分に理解し、アプリケーションの書き込みワークロードの再開を優先した場合の影響を注意深く検討してください。

追加の CRD は、クラスターのクォーラム状態を追跡するために使用されます。クォーラムフェイルオーバーが有効になった Cluster には、 Cluster リソースと同じ名前の FailoverQuorum リソースがあります。 FailoverQuorum CR は、クォーラムフェイルオーバーが有効になっているときにコントローラによって作成され、調整ループ中にプライマリインスタンスによって更新され、クォーラムチェック中にオペレー

ターによって読み取られます。これは、同期レプリケーションの最新の既知の構成を追跡するために使用されます。

FailoverQuorum リソースは、PostgreSQL 同期レプリケーションと連携して動作します。

> **警告**
>
> クライアントに返されたが、同期的に実行されていない`SET synchronous_commit TO local`で同期レプリケーションを明示的に無効にして作成された操作など、同期的に実行されていない`COMMIT`操作が昇格したレプリカに存在するという保証はありません。

クォーラムフェイルオーバーの例

次のシナリオでは、 R はプロモーブルレプリカの数、W はコミット前に書き込みを確認する必要があるレプリカの数、および N は潜在的な同期レプリカの合計数です。「Failover」列は、クォーラムフェールオーバールールでフェールオーバーが許可されるかどうかを示します。

シナリオ 13 ノードクラスター、障害のあるポッド

instances: 3 、synchronous.number=1 、および dataDurability=required を含むクラスター。

- プライマリにのみ障害が発生した場合、2 つのプロモーション可能なレプリカが残ります R=2。R + W > N (2 + 1 > 2) 以降、フェイルオーバーが許可され安全です。
- プライマリと 1 つのレプリカの両方に障害が発生した場合、1 つのプロモートブルレプリカ R=1 のみが残ります。R + W = N 1 + 1 = 2 ため、データ損失の可能性を防ぐために、フェイルオーバーは許可されません。

R	W	N	Failover
2	1	2	☒
1	1	2	No

シナリオ 23 ノードクラスター、ネットワークパーティション

instances: 3 、synchronous.number: 1 、および dataDurability: required を含むクラスターでネットワークパーティションが発生します。

- オペレーターがプライマリと通信できる場合、フェールオーバーは発生しません。プライマリがスタンバイに到達できない場合、クラスターは同期レプリケーション要件によりトランザクションをコミットしないため、クラスターは影響を受ける可能性があります。
- オペレーターがプライマリに到達できないが、両方のレプリカ R=2 に到達できる場合、フェールオーバーは許可されます。オペレーターが 1 つのレプリカ R=1 のみに到達できる場合、同期的なものは別のレプリカである可能性があるため、フェールオーバーは許可されません。

R	W	N	Failover
2	1	2	☒
1	1	2	NO

シナリオ 35 ノードクラスター、ネットワークパーティション

instances: 5、synchronous.number=2、および dataDurability=required を含むクラスターでネットワークパーティションが発生します。

- オペレーターがプライマリと通信できる場合、フェールオーバーは発生しません。プライマリが少なくとも 2 つのスタンバイに到達できない場合、同期レプリケーション要件のためにトランザクションをコミットしないため、クラスターは影響を受ける可能性があります。
- オペレーターがプライマリに到達できないが、少なくとも 3 つのレプリカ R=3 に到達できる場合、フェールオーバーは許可されます。オペレーターが 2 つのレプリカ R=2 のみに到達できる場合、同期的なものがもう一方である可能性があるため、フェールオーバーは許可されません。

R	W	N	Failover
3	2	4	☒
2	2	4	NO

シナリオ 4 リモート同期レプリカを使用した 3 ノードクラスター

instances: 3 と standbyNamesPre または standbyNamesPost で定義されたリモート同期レプリカを含むクラスター。プライマリに障害が発生していると仮定します。

このシナリオでは、重要な考慮事項が必要です。 standbyNamesPre または standbyNamesPost にリストされているレプリカは、R ではカウントされません昇格することはできませんが、 N に含まれます同期書き込みを受信した可能性があります。したがって、synchronous.number <= len(standbyNamesPre) + len(standbyNamesPost) の場合、ローカルレプリカが必要なデータを持つことを保証できないため、フェイルオーバーは不可能です。オペレーターは検証中にこのような構成を防止しますが、明確にするためにいくつかの無効な構成を以下に示します。

構成例

構成 1 有効

```
instances: 3
postgresql:
  synchronous:
    method: any
    number: 2
    standbyNamesPre:
      - angus
```

この構成では、プライマリに障害が発生すると、R = 2 ローカルレプリカ、W = 2 、およびつのローカルレプリカ+1 つのリモートでフェールオーバーが可能になります。追加のレプリカに障害 R = 1 が発生した場合、フェイルオーバーは許可されません。

R	W	N	Failover
3	2	4	☒
2	2	4	No

構成 2 無効

```
instances: 3
postgresql:
  synchronous:
    method: any
    number: 1
    maxStandbyNamesFromCluster: 1
    standbyNamesPre:
      - angus
```

この構成では、 R = 2 ローカルレプリカ、W = 1 、およびローカルレプリカ+1 リモート。この設定ではフェールオーバーは不可能であるため、この構成ではクォーラムフェールオーバーを有効にできません。

R	W	N	Failover
1	1	2	Yes

構成 3 無効

```
instances: 3
postgresql:
  synchronous:
    method: any
    number: 1
    maxStandbyNamesFromCluster: 0
    standbyNamesPre:
      - angus
      - malcolm
```

この構成では、 R = 0 ローカルレプリカ、W = 1 、およびローカルレプリカ+2 リモート。この設定ではフェールオーバーは不可能であるため、この構成ではクォーラムフェールオーバーを有効にできません。

R	W	N	Failover
0	1	2	Yes

シナリオ 53 ノードクラスター、優先データ耐久性、ネットワークパーティション

ネットワークパーティションが発生した `instances: 3` 、`synchronous.number=1` 、および `dataDurability=preferred` を含むクラスターを検討します。

- オペレーターがプライマリと API サーバーの両方と通信できる場合、プライマリは動作を継続し、到達不能なスタンバイを `synchronous_standby_names` セットから削除します。
- プライマリがオペレーターまたは API サーバーに到達できない場合、クォーラムチェックが実行されます。プライマリは新しい構成を受信できないため、`FailoverQuorum` ステータスは変更されることはできません。オペレーターが両方のレプリカに到達できる場合、フェイルオーバーが許可されます `R=2` 。 1 つのレプリカ `R=1` のみが到達可能である場合、フェイルオーバーは許可されません。

R	W	N	Failover
2	1	2	☒
1	1	2	No

^1

5.33 Fencing

CloudNativePG クラスターの EDB Postgres AI のフェンシングは、PostgreSQL クラスターの 1 つ以上、またはすべてのインスタンスに障害があると思われる場合に、データを保護する究極のプロセスです。インスタンスがフェンスされると、PostgreSQL サーバープロセス `postmaster` はシャットダウンが保証されますが、ポッドは実行されたままです。これにより、フェンスが解除されるまで、ポッド上のデータが PostgreSQL によって変更されず、デバッグとトラブルシューティングの目的でファイルシステムを調査できることが保証されます。

5.33.1 インスタンスをフェンスする方法

CloudNativePG クラスターの EDB Postgres AI では、次のことをフェンスできます。

- 特定のインスタンス
- インスタンスのリスト
- Postgres 全体 `Cluster`

フェンシングは、`k8s.enterprisedb.io/fencedInstances` アノテーションの内容を介して制御されます。これは、JSON 形式のインスタンス名のリストを想定しています。アノテーションが `'["*"]'` 、ワイルドカードを使用したシングルトンリストに設定されている場合、クラスター全体がフェンスされます。アノテーションが空の JSON リストに設定されている場合、オペレーターは、アノテーションが設定されていないかのように動作します。

例

- `k8s.enterprisedb.io/fencedInstances: '["cluster-example-1"]'` は `cluster-example-1` インスタンスのみをフェンスします

- k8s.enterprisedb.io/fencedInstances: '["cluster-example-1","cluster-example-2"]' は、 cluster-example-1 および cluster-example-2 インスタンスをフェンスします
- k8s.enterprisedb.io/fencedInstances: '["*"]' は、クラスター内のすべてのインスタンスをフェンスします。

アノテーションは、kubectl annotate コマンドを介して、または kubectl cnp fencing on サブコマンドを使用して透過的な方法で、Kubernetes オブジェクトに手動で設定できます。

```
#  to fence only one instance

kubectl cnp fencing on cluster-example 1

#  to fence all the instances in a Cluster

kubectl cnp fencing on cluster-example "*"
```

次に、以前にフェンスされたインスタンスを使用した Cluster の例を示します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
    annotations:
      k8s.enterprisedb.io/fencedInstances: ["cluster-example-1"]
[...]
```

5.33.2 フェンシングのリフティング方法

フェンシングは、注釈をクリアするか、別の値に設定することにより解除できます。

フェンシングに関しては、これは kubectl annotate を使用して手動で行うか、次のように kubectl cnp fencing サブコマンドを使用して実行できます。

```
#  to lift the fencing only for one instance

#  N.B.: at the moment this wont work if the whole cluster was fenced previously,

#        in that case you will have to manually set the annotation as explained above

kubectl cnp fencing off cluster-example 1

#  to lift the fencing for all the instances in a Cluster

kubectl cnp fencing off cluster-example "*"
```

5.33.3 フェンシングの仕組み

インスタンスがフェンシング用に設定されると、スイッチオーバーの場合と同様に、postmaster プロセスをシャットダウンする手順が開始されます。これは、タイムアウトを.spec.switchoverDelay に設定した最初の高速シャットダウンと、それに続く即時シャットダウンで構成されます。次に

- ポッドは生きたままになります
- ポッドは_Ready_としてマークされません
- Postgres インスタンスを起動する必要がないすべての変更がリコンサイルされます。
- 構成ファイル
- 証明書およびすべての暗号化マテリアル
- 1 に設定される cnp_collector_fencing_on を除き、メトリックは収集されません

> **警告**
>
> **プライマリインスタンス**がフェンスされている場合、そのポストマスタープロセスはシャットダウンされますが、フェイルオーバーは実行されず、アプリケーションの操作性が中断されます。フェンスが解除されると、プライマリインスタンスはフェイルオーバーを実行せずに再度起動されます。

そのため、厳密に必要な場合にのみプライマリインスタンスをフェンスすることをお勧めします。

5.34 Declarative hibernation

CloudNativePG クラスターの EDB Postgres AI は、PostgreSQL クラスターを起動、実行、いつでも利用できるように設計されています。

一部の種類のワークロードは、ワークロードがアクティブなときにのみデータベースを起動する必要があります。バッチ駆動のソリューションは、そのようなケースの 1 つです。

バッチ駆動ソリューションでは、バッチプロセスの実行中にのみデータベースを起動する必要があります。

宣言的ハイバーネーション機能を使用すると、データベース PVC を維持したままデータベースポッドを削除することにより、CPU パワーを節約できます。

5.34.1 ハイバネーション

クラスターを休止状態にするには、 k8s.enterprisedb.io/hibernation=on アノテーションを設定します。

```
$ kubectl annotate cluster <cluster-name> --overwrite k8s.enterprisedb.io/
↪hibernation=on
```

冬眠したクラスターには実行中のポッドはありませんが、PVC は保持されるため、後でクラスターをリハイドレートできます。レプリカ PVC は、プライマリの PVC に加えて保持されます。

ハイバーネーションプロシージャは、プライマリポッドを削除してからレプリカポッドを削除し、スイッチオーバーを回避して、レプリカの同期が保たれるようにします。

ハイバーネーションステータスは、k8s.enterprisedb.io/hibernation 条件を探すことにより監視できます。

```
$ kubectl get cluster <cluster-name> -o "jsonpath={.status.conditions[?(.type==\"k8s.
↪enterprisedb.io/hibernation\")]}"

{
        "lastTransitionTime":"2023-03-05T16:43:35Z",
        "message":"Cluster has been hibernated",
        "reason":"Hibernated",
        "status":"True",
        "type":"k8s.enterprisedb.io/hibernation"
}
```

ハイバーネーションステータスは、kubectl の cnp プラグインの status サブコマンドで読み取ることもできます。

```
$ kubectl cnp status <cluster-name>
Cluster Summary
Name:               cluster-example
Namespace:          default
PostgreSQL Image:   docker.enterprisedb.com/k8s/postgresql:18.4-standard-ubi9
Primary instance:   cluster-example-2
Status:             Cluster in healthy state
Instances:          3
Ready instances:    0

`Hibernation`
Status   Hibernated
Message  Cluster has been hibernated
Time     2023-03-05 16:43:35 +0000 UTC
[..]
```

5.34.2 水分補給

クラスターを rehydrate には、k8s.enterprisedb.io/hibernation アノテーションを off に設定します。

```
$ kubectl annotate cluster <cluster-name> --overwrite k8s.enterprisedb.io/
↪hibernation=off
```

または、完全に設定を解除します。

```
$ kubectl annotate cluster <cluster-name> k8s.enterprisedb.io/hibernation-
```

ポッドが再作成され、クラスターがオペレーションを再開します。

5.35 PostGIS

PostGIS は、 PostgreSQL の非常に人気のオープンソース拡張機能であり、GIS 地理情報システムオブジェクトをデータベースに保存し、SQL を介して照会するためのサポートを導入します。

CloudNativePG コミュニティは、維持される PostgreSQL Container images 上にビルドされるコンテナイメージを維持します。詳細については、次をご覧ください。

•

postgis-containers

•

postgis-containers

さらに、EDB は次のコンテナイメージを提供します。

- コミュニティ PostgreSQL + PostGIS
- EDB Postgres Advanced Server + PostGIS
- EDB Postgres Extended PGE + PostGIS

そして、 `docker.enterprisedb.com` の公式レジストリで`-postgis` サフィックスを使用して利用できるようにします。

5.35.1 PostGIS クラスターに関する基本的な概念

概念的に、PostGIS ベースの PostgreSQL クラスターまたは単に PostGIS クラスターは、他の PostgreSQL クラスターと似ています。唯一の違いは次のとおりです。

- PostGIS および関連ライブラリのシステム内の存在
- PostGIS 拡張機能のデータベース内の存在

CloudNativePG クラスターの EDB Postgres AI は不変アプリケーションコンテナーに基づいているため、PostGIS をプロビジョニングする唯一の方法は、オペランドに使用するコンテナイメージに追加することです。*Container Image Requirements* は、これを実現する方法に関する詳細な手順を提供します。もっと簡単には、以下の例にあるように、コミュニティからの PostGIS コンテナイメージを使用できます。

2 番目のステップは、PostgreSQL データベースに拡張機能をインストールすることです。これは 2 つの方法で行うことができます。

- アプリケーションデータベースにインストールします。これは、マイクロサービスアーキテクチャに従って、クラスターでホストするメインで唯一のデータベースです。または
- インスタンスがマルチプルのデータベースで共有されるモノリスアーキテクチャを採用する場合、`template1` データベースにインストールして、クラスター内に作成することになるすべてのデータベースで利用できるようにします

> **注釈**
>
> データベース内のマイクロサービスとモノリスアーキテクチャの詳細については、*How many databases should be hosted in a single PostgreSQL instance?* を参照してください。

または *Database import* 。

5.35.2 PostGIS で新しい PostgreSQL クラスターを作成する

PostGIS 3.6 を使用して新しい PostgreSQL 18 クラスターを作成することにします。

最初のステップは、オペランドに適切な PostGIS コンテナイメージを使用し、`Cluster` リソースの`.spec.imageName` オプションを適切に設定することです。

以下の postgis-example.yaml は、PostGIS クラスターの作成を行う方法に関するガイダンスを提供します。

> **警告**
>
> 構成より規約は CloudNativePG クラスターの EDB Postgres AI に適用されますが、運用用にシステムの構成とチューニングに時間を費やす必要があることを考慮してください。また、以下の例の `imageName` は、PostgreSQL 17 の最新の利用可能なイメージを意図的にポイントしています。真の不変性を実現するには、特定のイメージ名、またはできれば SHA256 ダイジェストを使用する必要があります。

```yaml
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: postgis-example
spec:
  instances: 1
  imageName: docker.enterprisedb.com/k8s/postgresql:17-postgis-ubi9
  storage:
    size: 1Gi
  postgresql:
    parameters:
      log_statement: ddl
- --
apiVersion: postgresql.cnpg.io/v1
kind: Database
metadata:
  name: postgis-example-app
spec:
  name: app
  owner: app
  cluster:
    name: postgis-example
```

(次のページに続く)

(前のページからの続き)

```
  extensions:
    - name: postgis
    - name: postgis_topology
    - name: fuzzystrmatch
    - name: postgis_tiger_geocoder
```

この例では、 Database リソースの宣言的拡張管理を活用して、指定された拡張機能を app データベースに追加します。

注釈

時間をかけて、 *postInitApplicationSQL* などの *API Reference - v1.30.0* から `.spec.bootstrap.initdb` で使用可能なオプションを確認します。

注釈

詳細については、 **データベース内の拡張機能の管理** を参照してください。

注釈

詳細については、 **データベース内の拡張機能の管理** を参照してください。

app データベースに接続することにより、コンテナ内にある PostGIS の利用可能なバージョンを簡単に確認できます。このドキュメントの値とは異なる値を取得する場合があります。

```
$ kubectl cnpg psql postgis-example -- app
psql (18.0 (Debian 18.0-1.pgdg13+3))
Type "help" for help.

app=# SELECT * FROM pg_available_extensions WHERE name ~ ^postgis ORDER BY 1;
           name           | default_version | installed_version |
     comment
- ------------------------+-----------------+-------------------+--------------------
↪---------------------------------------
 postgis                  | 3.6.0           | 3.6.0             | PostGIS geometry␣
↪and geography spatial types and functions
 postgis-3                | 3.6.0           |                   | PostGIS geometry␣
↪and geography spatial types and functions
 postgis_raster           | 3.6.0           |                   | PostGIS raster␣
↪types and functions
 postgis_raster-3         | 3.6.0           |                   | PostGIS raster␣
↪types and functions
```

(次のページに続く)

(前のページからの続き)

```
 postgis_sfcgal           | 3.6.0           |                   | PostGIS SFCGAL␣
↪functions
 postgis_sfcgal-3         | 3.6.0           |                   | PostGIS SFCGAL␣
↪functions
 postgis_tiger_geocoder   | 3.6.0           | 3.6.0             | PostGIS tiger␣
↪geocoder and reverse geocoder
 postgis_tiger_geocoder-3 | 3.6.0           |                   | PostGIS tiger␣
↪geocoder and reverse geocoder
 postgis_topology         | 3.6.0           | 3.6.0             | PostGIS topology␣
↪spatial types and functions
 postgis_topology-3       | 3.6.0           |                   | PostGIS topology␣
↪spatial types and functions
(10 rows)
```

次のステップは、Database リソースにリストされている拡張機能が app データベースに正しくインストールされていることを確認することです。

```
app=# \dx
                                              List of installed extensions
          Name          | Version | Default version |   Schema   |
    Description
- ----------------------+---------+-----------------+------------+-------------------
↪----------------------------------------
 fuzzystrmatch          | 1.2     | 1.2             | public     | determine␣
↪similarities and distance between strings
 plpgsql                | 1.0     | 1.0             | pg_catalog | PL/pgSQL␣
↪procedural language
 postgis                | 3.6.0   | 3.6.0           | public     | PostGIS geometry␣
↪and geography spatial types and functions
 postgis_tiger_geocoder | 3.6.0   | 3.6.0           | tiger      | PostGIS tiger␣
↪geocoder and reverse geocoder
 postgis_topology       | 3.6.0   | 3.6.0           | topology   | PostGIS topology␣
↪spatial types and functions
```

最後に

```
app=# SELECT postgis_full_version();
                                                                                postgis_
↪full_version
- ----------------------------------------------------------------------------------------
↪----------------------------------------------------------------------------------------
↪---
 POSTGIS="3.6.0 4c1967d" [EXTENSION] PGSQL="180" GEOS="3.13.1-CAPI-1.19.2" PROJ="9.6.
↪0 NETWORK_ENABLED=OFF URL_ENDPOINT=https://cdn.proj.org USER_WRITABLE_DIRECTORY=/
↪tmp/proj DATABASE_PATH=/usr/share/proj/proj.
```

(次のページに続く)

(前のページからの続き)

```
db" (compiled against PROJ 9.6.0) LIBXML="2.9.14" LIBJSON="0.18" LIBPROTOBUF="1.5.1"␣
↪WAGYU="0.5.0 (Internal)" TOPOLOGY
(1 row)
```

5.36 Container Image Requirements

Kubernetes の CloudNativePG クラスターオペレーターの EDB Postgres AI は、次の要件を満たす互換性のある PostgreSQL コンテナイメージで動作するように設計されています。

- PostgreSQL 実行可能ファイルはシステムパスで利用できる必要があります。
- `initdb`
- `postgres`
- `pg_ctl`
- `pg_controldata`
- `pg_basebackup`
- 適切なロケール設定が構成済み

オプションのコンポーネント

- PGAudit 拡張機能監査ログが必要な場合にのみ必要
- du `kubectl cnp status` に使用

> **注釈**
>
> Barman Cloud 実行可能ファイルは、CloudNativePG クラスターの EDB Postgres AI では必要ありません。推奨されるアプローチは、専用の Barman Cloud Plugin を使用することです。

イメージ定義にエントリーポイントもコマンドも必要ありません。CloudNativePG クラスターの EDB Postgres AI は、インスタンスマネージャーで自動的にオーバーライドします。

> **警告**
>
> CloudNativePG クラスターの EDB Postgres AI は、PostgreSQL アプリケーションコンテナイメージの**マルチプル/オプショナルのホットスタンバイサーバーアーキテクチャを備えたプライマリ**のみをサポートしています。

5.36.1 イメージタグの要件

オペレーターが情報に基づいた決定を下せるように、PostgreSQL メジャーバージョンを正確に検出する必要があります。この検出は 2 つの方法で発生できます。

1. `imageCatalogRef` の `major` フィールドが定義されている場合、それを利用します。

2. 明示的に指定されていない場合、`imageName` のイメージタグからメジャーバージョンを自動検出します。

自動検出が機能するには、イメージタグが特定の形式に準拠している必要があります。これは、有効な PostgreSQL メジャーバージョン番号 15.6 または 16 などで始まり、オプションでドットとパッチレベルが続く必要があります。

これに続いて、タグには、Docker タグで受け入れられる有効な文字の組み合わせを含めることができ、前にドット、アンダースコア、またはマイナス記号が続きます。

受け入れられるイメージタグの例

- `12.1`
- `13.3.2.1-1`
- `13.4`
- `14`
- `15.5-10`
- `16.0`

警告

latest は、画像の有効なタグとは見なされません。

注釈

イメージタグ要件は、カタログで定義されたイメージには適用されません。

5.37 Image Volume Extensions

CloudNativePG クラスターの EDB Postgres AI は、 *Kubernetes `ImageVolume* feature <https://kubernetes.io/docs/tasks/configure-pod-container/image-volumes/>`_ を使用して、ポッドの起動時に `Cluster` への **PostgreSQL 拡張機能の動的ロード** をサポートします

PostgreSQL 18 で導入された `extension_control_path` GUC とともに、EDB Postgres AI for CloudNativePG Cluster プロジェクトが貢献した機能です。

この機能を使用すると、OCI 準拠のコンテナイメージとしてパッケージされた PostgreSQL extension を、実行中のポッド内の指定されたファイルシステムパスに読み取り専用で不変のボリュームとしてマウントできます。

`CREATE EXTENSION` コマンドを介してデータベースレベルのインストールが必要な拡張機能の場合、*Postgres データベースの宣言的管理*

すべての PostgreSQL データベースで一貫した自動セットアップを保証します。

5.37.1 公式拡張機能のイメージとカタログ

CloudNativePG クラスターコミュニティの EDB Postgres AI は、pgvector を含む拡張コンテナイメージスイートを維持します

および postgres-extensions-containers の一部としての PostGIS 。これらのイメージは、*official PostgreSQL `minimal* images <https://github.com/enterprisedb/docker-postgres?tab=readme-ov-file#minimal-images>`_ の上にビルドされます。

> **注釈**
>
> このドキュメントは、サードパーティが独自のイメージとカタログをビルドするために必要な技術仕様を提供しますが、次の手順では、公式の拡張機能イメージとカタログの展開と使用に特に焦点を当てています。

この機能は、PostgreSQL オペランドイメージから拡張機能のディストリビューションを分離することにより、コンテナで PostgreSQL を実行する際の重要な障壁を取り除きます。ビルド時に拡張機能を埋め込む必要がなくなり、公式の最小限のオペランドイメージを使用し、必要な拡張機能のみを `Cluster` 定義に直接または **イメージカタログを使用する** を介して追加できます。

このアプローチでは、コアデータベースコンテナに操作に必要な重要なバイナリのみが含まれるようにすることにより、データベースクラスターの攻撃対象領域を大幅に削減します。不要な拡張機能、ライブラリ、およびビルド時の依存関係を除外することにより、エクスプロイトの潜在的なエントリーポイントを最小限に抑え、脆弱性管理を簡素化します。このアーキテクチャは、データワークロードの不変の最小限のベースイメージを維持することにより、サプライチェーンのセキュリティを強化し、運用オーバーヘッドを削減します。

> **重要**
>
> 拡張イメージは、 Image Specifications に従ってビルドする必要があります。

CloudNativePG クラスターの EDB Postgres AI でイメージボリューム拡張を使用するには、以下が必要です。

- **PostgreSQL 18 以降** `extension_control_path` サポートに必要です。
- **Kubernetes 1.35 以降** `ImageVolume` 機能はデフォルトで有効になっています。 Kubernetes 1.33 および 1.34 のユーザーは、`ImageVolume` 機能ゲートを手動で有効にする必要があります。
- **``ImageVolume`` サポートを備えたコンテナランタイム**
- `containerd` v2.1.0 以降、or
- `CRI-O` v1.31 以降。
- **CloudNativePG クラスター互換の拡張コンテナイメージの EDB Postgres AI** 、以下を保証します。
- `Cluster` リソースの PostgreSQL メジャーバージョンとの一致。
- `Cluster` リソースの互換性のあるオペレーティングシステムディストリビューション。
- `Cluster` リソースの CPU アーキテクチャと一致します。

5.37.2 その仕組み

`.spec.postgresql.extensions` スタンザを使用して、拡張イメージを新しいまたは既存の `Cluster` リソースに追加できます。

重要

新しい拡張機能が実行中の `Cluster` に追加されると、CloudNativePG クラスターの EDB Postgres AI は自動的に ローリング更新戦略の例 - unsupervised すべての場合のプライマリの自動更新 レプリカがアップグレードされたデフォルト - supervised を実行するには手動の監視が必要です プライマリのスイッチオーバー primaryUpdateStrategy: unsupervised をトリガーして、新しいイメージボリュームを各ポッドにアタッチします。実稼働環境に新しい拡張機能を追加する前に、ステージング環境で完全にテストして、PostgreSQL クラスターを異常な状態にする可能性のある構成の問題を防止してください。

構成ソースと優先順位

`extensions` スタンザは、エントリのリストを受け入れます。各エントリは、クラスター内で一意である必要がある `name` を必要とします。

重要

name は、小文字の英数字、アンダースコア `_` またはハイフン `-` で構成され、英数字で開始および終了する必要があります。 59 文字に制限されており、拡張機能の Kubernetes ボリューム名を取得するときに CloudNativePG クラスターの EDB Postgres AI が追加するプレフィックス用の余地を残しています 63 文字の RFC 1123 制限で制限されています。

- **Via Image Catalog** クラスターが、現在の PostgreSQL メジャーバージョンの拡張機能を定義する `ImageCatalog` を参照する場合、これらの定義はデフォルト構成として機能します。これにより、カタログは `image.reference` を一元管理できますが、クラスターは単に名前で拡張機能を有効にするだけです。
- **クラスター内に直接**クラスターがイメージカタログを使用しない場合、`image.reference` フィールドは必須で、拡張機能イメージの有効なコンテナレジストリパスを明示的に指定する必要があります。`image` スタンザは

Kubernetes `ImageVolume` API <https://kubernetes.io/docs/concepts/storage/volumes/#image>`_ の後にあります。

_“convention over configuration”_パラダイムに従って、CloudNativePG クラスターの EDB Postgres AI は、完全な柔軟性を提供します。`image.reference` を含むカタログから継承した値は、特定のローカル要件を満たすように `Cluster` 定義内で選択的にオーバーライドできます。

マウンティングと PostgreSQL の構成

各拡張機能は、ポッド内の`/extensions/<EXTENSION_NAME>` に読み取り専用ボリュームとしてマウントされます。

デフォルトでは、CloudNativePG クラスターの EDB Postgres AI は、次の設定により関連する GUC を自動的に管理します。

- `extension_control_path` ～`/extensions/<EXTENSION_NAME>/share` 、 PostgreSQL が`/extensions/<EXTENSION_NAME>/share/extension` 内の拡張制御ファイルを見つけられるようにします
- `dynamic_library_path` ～`/extensions/<EXTENSION_NAME>/lib`

> **注釈**
>
> アンダースコアを含む拡張名たとえば`pg_ivm` は、RFC 1123 DNS ラベル要件に準拠するように Kubernetes ボリューム名にハイフンたとえば`pg-ivm` を使用するように変換されます。サニタイズ後に同じになる拡張名は使用しないでくださいたとえば、*pg_ivm* と `pg-ivm` は両方とも `pg-ivm` にサニタイズします。 Webhook 検証は、このような競合を防ぎます。

これらの値は、拡張機能が `extensions` リストで定義された順序で追加され、PostgreSQL 内の決定論的なパスの解決が保証されます。これにより、PostgreSQL は、ポッド内の手動構成を必要とせずに拡張機能を検出してロードできます。

カスタム画像レイアウトに一致するようにこれらのパスを手動で調整することはできますが、CloudNativePG クラスターカタログの公式 EDB Postgres AI カタログは、これらのオプションを各拡張機能の正しい値に事前構成し、すぐに機能するように保証します。

> **重要**
>
> 拡張イメージに共有ライブラリが含まれる場合、オペランドイメージと同じ PostgreSQL メジャーバージョン、オペレーティングシステムディストリビューション、および CPU アーキテクチャ用にコンパイルする必要があります。 CloudNativePG クラスターカタログに公式の EDB Postgres AI を使用すると、この互換性が自動的に保証されます。カタログは、クラスターの特定の環境と一致するように設計されており、ライブラリまたはアーキテクチャの不一致によって発生するランタイムの問題を防ぎます。

データベース内の拡張機能の管理 を活用できます

標準の PostgreSQL `CREATE EXTENSION` メカニズムをサポートする拡張機能の最終的なアクティブ化を自動化します。

拡張機能イメージがマウントされ、GUC が構成されると、拡張機能は PostgreSQL インスタンスで利用できるようになりますが、特定のデータベース内ではまだアクティブになっていません。インストールするには、`Database` リソースを定義します。次の例では、*official `postgres-extensions-containers* project <https://github.com/cloudnative-pg/postgres-extensions-containers/tree/main/pgvector>`_ の実装標準に従って、`pgvector` を使用します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Database
metadata:
  name: cluster-example-app
spec:
  name: app
  owner: app
```

(次のページに続く)

(前のページからの続き)

```
  cluster:
    name: cluster-example
  extensions:
    - name: vector
      version: "<VERSION>"
```

CloudNativePG クラスターの EDB Postgres AI は、ターゲットデータベース内で `CREATE EXTENSION IF NOT EXISTS vector` を実行することにより、このリソースを自動的にリコンサイルします。これにより、目的の状態が維持され、すべてのインスタンスに一貫して適用されます。

> **注釈**
>
> モジュールと呼ばれることが多い一部の PostgreSQL コンポーネントは、 *CREATE EXTENSION* メカニズムを使用しません。これらは通常、サーバーの起動時に`shared_preload_libraries` を介してロードする必要がある共有ライブラリで構成されます。イメージカタログを使用してこれらのバイナリの配布を簡素化する場合でも、ライブラリ名を *shared_preload_libraries* に手動で追加する必要があります

`Cluster` 定義で変更して、メモリにロードされるようにします。

以前に予想されたように、CloudNativePG クラスターの EDB Postgres AI は、拡張イメージを定義する 2 つの方法を提供します。どちらもポッドレベルで同じ結果を達成しますが、一貫した運用準備が整ったサプライチェーンを維持するには、イメージカタログを使用することをお勧めします。

画像カタログ経由推奨

> **注釈**
>
> イメージカタログの拡張コンテナイメージのサポートは、CloudNativePG クラスター 1.29 の EDB Postgres AI で導入されました。

official ones のような

コミュニティが提供するシステムでは、特定のコンテナイメージ `reference` や必要なファイルシステムパスなどの複雑な構成の詳細が一元管理されます。

`Cluster` 定義は、名前で拡張機能に「オプトイン」するだけでよいため、クリーンで宣言的なままです。オペレーターは、カタログの定義に基づいて、画像の解像度と必要な PostgreSQL 設定を自動的に処理します。

カタログから `pgvector` のような拡張機能を有効にするには、次の抜粋のように、クラスターの `extensions` リストに追加します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
```

(次のページに続く)

(前のページからの続き)

```
spec:
  # ... <snip>
  imageCatalogRef:
    apiGroup: postgresql.k8s.enterprisedb.io
    kind: ClusterImageCatalog
    name: postgresql-minimal-ubi9
    major: 18

  postgresql:
    extensions:
      - name: pgvector # Resolves all details, including the image reference, from␣
↪the catalog
      # ... <snip>
```

この方法により、拡張イメージは、同じカタログエントリで定義された PostgreSQL オペランドイメージと常に互換性があることが保証されます。

クラスター内に直接

> **注釈**
>
> *Cluster* リソースに拡張機能を直接定義することは元の方法であり、CloudNativePG クラスター 1.29 の EDB Postgres AI より前のバージョンの唯一のオプションのままです。現在のカタログに存在しない拡張機能を使用する必要がある場合、またはカスタムイメージをテストする場合にも役立ちます。

次の例は、公式コンテナイメージを明示的に指定することにより pgvector を追加する方法を示しています。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  # ... <snip>
  postgresql:
    extensions:
      - name: pgvector
        image:
          reference: ghcr.io/cloudnative-pg/pgvector:<TAG>
```

> **Tip**
>
> *Cluster* で直接提供される構成が優先されることに注意してください。カタログを参照しているが、*Cluster* スタンザでも同じ拡張名を定義している場合、 *Cluster* の設定がカタログの設定をオーバーライドします。 *ExtensionConfiguration* のすべてのフィールド

Cluster レベルでオーバーライドして、完全な柔軟性を提供できます。name フィールドは例外です。

拡張機能はイメージカタログと直接クラスター定義の両方から調達できるため、CloudNativePG クラスターの EDB Postgres AI は、Cluster ステータスの構成の「解決」ビューを提供します。これは、オペレーターがポッドをプロビジョニングするために使用する最終的な有効な状態です。

status.pgDataImageInfo.extensions フィールドを検査して、継承またはオーバーライドの結果を確認できます。

```
#  ... <snip>

status:
  # ... <snip>
  pgDataImageInfo:
    image: # registry path for your PostgreSQL image
    majorVersion: 18
    extensions:
      - name: foo
        image:
          reference: # registry path for your `foo` extension image
      - name: bar
        image:
          reference: # registry path for your `bar` extension image
    # ... <snip>
```

このセクションは、次の場合に特に役立ちます。

- **解決策の検証** Cluster の名前で要求された拡張機能が、関連するカタログから image.reference を正しくプルしたことの確認。
- **オーバーライドの確認**クラスターレベルのオーバーライド特定のイメージバージョンなどによって、カタログのデフォルト値が正常に置き換えられたことを確認します。
- **トラブルシューティング**ポッドがローリング更新される前に、目的のすべての拡張機能がオペレーターによって認識されることを確認します。

5.37.3 PostgreSQL クラスターからの拡張機能の削除

拡張機能の削除には、インフラストラクチャとデータベースのメタデータの両方をクリーンアップする 2 段階のプロセスが含まれます。

「library not found」エラーを回避するために、最初にデータベースから拡張機能を削除する必要があります。宣言的管理を使用している場合は、拡張機能に ensure: absent を設定して Database リソースを更新します。

```
spec:
  # ... <snip>
  extensions:
    - name: pgvector
```

(次のページに続く)

(前のページからの続き)

```
    ensure: absent
  # ... <snip>
```

これにより、CloudNativePG クラスターの EDB Postgres AI がトリガーされ、データベース内で DROP EXTENSION を実行します。

拡張機能が shared_preload_libraries に追加された場合、Cluster 構成から削除する必要もあります。

次に、 Cluster リソースの.spec.postgresql.extensions リストから拡張エントリを削除します。オペレーターは、ローリング更新を実行して ImageVolume を切断し、関連する GUC パスを更新します。

5.37.4 高度なトピック

場合によっては、特に次の場合、デフォルトの予想される構造では拡張機能イメージには不十分である可能性があります。

- 拡張機能には追加のシステムライブラリが必要です。
- 複数の拡張機能が同じイメージにバンドルされています。
- イメージはカスタムディレクトリ構造を使用します。
- 拡張機能が正常に機能するには、外部バイナリが必要です。
- 拡張機能には、特定の環境変数を設定する必要があります。

_“convention over configuration”_パラダイムに従って、CloudNativePG クラスターの EDB Postgres AI を使用すると、次のフィールドを介して各拡張イメージの構成を細かく制御できます。

- extension_control_path PostgreSQL の extension_control_path に追加されるコンテナイメージ内の相対パスのリスト、拡張子制御ファイルを見つけられるようにします。
- dynamic_library_path PostgreSQL の dynamic_library_path に追加されるコンテナイメージ内の相対パスのリスト、拡張機能の共有ライブラリファイルを見つけられるようにします。
- ld_library_path Postgres プロセスの LD_LIBRARY_PATH 環境変数に追加されるコンテナイメージ内の相対パスのリストにより、PostgreSQL は実行時に必要なシステムライブラリを見つけられます。
- bin_path Postgres プロセスの PATH 環境変数に追加されるコンテナイメージ内の相対パスのリストにより、PostgreSQL は実行時にバイナリを見つけられます。
- env Postgres プロセス内の環境変数として設定される name および value ペアのリスト。

この柔軟性により、明瞭性と予測可能性を維持しながら、複雑なまたは非標準の拡張画像をサポートできます。

カスタムパスの設定

拡張イメージがライブラリと制御ファイルにデフォルトの lib および share ディレクトリを使用しない場合、extension_control_path および dynamic_library_path を明示的に設定することによりデフォルトをオーバーライドできます。

例

```yaml
spec:
  # ... <snip>
  postgresql:
    extensions:
      - name: my-extension
        extension_control_path:
          - my/share/path
        dynamic_library_path:
          - my/lib/path
        image:
          reference: # registry path for your extension image
      # ... <snip>
    # ... <snip>
  # ... <snip>
```

CloudNativePG クラスターの EDB Postgres AI は、次を使用して PostgreSQL を構成します。

- `extension_control_path` に`/extensions/my-extension/my/share/path` を追加
- `dynamic_library_path` に`/extensions/my-extension/my/lib/path` を追加

これにより、PostgreSQL は、標準以外のレイアウトでも、拡張機能の制御ファイルと共有ライブラリを正しく検出できます。

マルチ拡張子画像

同じコンテナイメージ内に複数の拡張機能を含め、各拡張機能のファイルが独自のサブディレクトリに存在する構造を採用する必要がある場合があります。

たとえば、PostGIS と pgRouting を単一のイメージに、それぞれが独自のサブディレクトリにパッケージ化するには。

```yaml
#  ...

spec:
  # ... <snip>
  postgresql:
    extensions:
      - name: geospatial
        extension_control_path:
          - postgis/share
          - pgrouting/share
        dynamic_library_path:
          - postgis/lib
          - pgrouting/lib
        # ... <snip>
        image:
          reference: # registry path for your geospatial image
```

(次のページに続く)

(前のページからの続き)

```
      # ... <snip>
    # ... <snip>
  # ... <snip>
```

システムライブラリを含める

PostGIS などの一部の拡張機能は、ベースの PostgreSQL イメージに存在しない場合があるシステムライブラリを必要とします。これらの要件をサポートするには、拡張機能コンテナイメージ内に必要なライブラリをパッケージし、`ld_library_path` フィールドを使用して PostgreSQL で利用できるようにします。

たとえば、拡張機能イメージに、必要なライブラリを含む `system` ディレクトリが含まれる場合

```
#  ...

spec:
  # ... <snip>
  postgresql:
    extensions:
      - name: postgis
        # ... <snip>
        ld_library_path:
          - system
        image:
          reference: # registry path for your PostGIS image
      # ... <snip>
    # ... <snip>
  # ... <snip>
```

CloudNativePG クラスターの EDB Postgres AI は、`/extensions/postgis/syslib` を含むように `LD_LIBRARY_PATH` 環境変数を設定し、PostgreSQL が実行時にこれらのシステムライブラリを見つけてロードできるようにします。

> **重要**
>
> PostgreSQL プロセスの開始時に `ld_library_path` を設定する必要があるため、この値を変更するには、新しい値を有効にするには**クラスターの再起動**が必要です。 CloudNativePG クラスターの EDB Postgres AI は、現在この再起動を自動的にトリガーしません。 *ld_library_path* を変更した後、クラスターを手動で再起動する必要がありますたとえば、*cnp restart* を使用して。

一部の拡張機能は、拡張機能自分自身と対話するために必要なバイナリを提供する必要がある場合があります。これらの要件をサポートするには、拡張機能コンテナイメージ内に必要なバイナリをパッケージし、`bin_path` フィールドを使用して PostgreSQL で利用できるようにします。

たとえば、拡張機能イメージに、必要なバイナリを含む `bin` ディレクトリが含まれる場合

```
#  ...

spec:
  # ... <snip>
  postgresql:
    extensions:
      - name: my-extension
        # ... <snip>
        bin_path:
          - bin
        image:
          reference: # registry path for your extension image
      # ... <snip>
    # ... <snip>
  # ... <snip>
```

CloudNativePG クラスターの EDB Postgres AI は、Postgres プロセスの `PATH` 環境変数に`/extensions/my-extension/bin` を追加し、PostgreSQL が実行時にこれらのバイナリを見つけられるようにします。

> **⚠ 警告**
>
> PostgreSQL プロセスの開始時に `bin_path` を設定する必要があるため、この値を変更するには、**クラスターの再起動**して新しい値を有効にする必要があります。 CloudNativePG クラスターの EDB Postgres AI は、現在この再起動を自動的にトリガーしません。 *bin_path* を変更した後、クラスターを手動で再起動する必要がありますたとえば、*cnp restart* を使用して。

特定の拡張機能が正常に機能するには、PostgreSQL プロセス内で特定の環境変数を設定する必要があります。拡張機能の構成内の env フィールドを使用して、これらの変数を定義できます。

次の例は、静的変数 `FOO` とプレースホルダーを使用した変数 `BAR` を設定する方法を示しています。

```
#  ...

spec:
  # ... <snip>
  postgresql:
    # ... <snip>
    extensions:
      - name: my-extension
        image:
          reference: "ghcr.io/org/my-extension:latest"
        env:
          - name: FOO
            value: "static_value"
          - name: BAR
```

(次のページに続く)

(前のページからの続き)

```
            value: "${image_root}/lib"
      # ... <snip>

#  ...
```

動的プレースホルダー展開

value フィールドは、プレースホルダー展開をサポートし、拡張機能の環境に基づいて動的構成を可能にします。次のプレースホルダーが現在サポートされています。

Placeholder	Description
${image_root}	拡張機能のボリューム`/extensions/my-extension`などの絶対マウントパスに解決します。

${image_root} プレースホルダーは、拡張イメージ内のアプリケーションまたは共有ライブラリが、独自のマウントされたボリューム内の特定のサブディレクトリまたはファイルをポイントする環境変数が必要な場合に特に役立ちます。

上記の例では、拡張機能が/extensions/my-extension にマウントされている場合、変数 BAR は/extensions/my-extension/lib に解決され、オペレーターが選択した特定のマウントパスに関係なく、ライブラリがその依存関係を見つけられるようにします。

Tip

認識されないプレースホルダーたとえば *${typo}* は、入場時に拒否されます。値にリテラル`${...}` が必要な場合は、ドル記号を 2 倍にしてエスケープします`$${...}` 。たとえば、値`$${not_expanded}` はリテラル文字列`${not_expanded}` を生成します。

PostgreSQL プロセスの環境変数は、複数の場所で定義できます。同じ変数名が複数の場所に出現する場合、次の優先順位が適用されます最後に最も高い優先順位。

1. **``spec.env`` / ``spec.envFrom``** クラスターレベルで定義された変数は、Kubernetes によってコンテナ環境に注入され、ベースとして機能します。
2. **拡張機能 ``env``** 各拡張機能の env フィールドで定義された変数は、同じ名前のクラスターレベル変数をオーバーライドします。
3. **拡張オーダー**複数の拡張が同じ変数を定義している場合、extensions 配列の最後にリストされている拡張が優先されます。

CloudNativePG クラスターの EDB Postgres AI は、拡張機能環境変数が以前の拡張機能によって既に設定された値を上書きすると、インスタンスマネージャーログに警告を発し、潜在的な競合の診断に役立ちます。

重要

予約変数 オペレーター使用のために予約された環境変数`PG` または`CNP_`、および`POD_NAME`、*NAMESPACE*、および`CLUSTER_NAME` で始まる名前および専用フィールドによって管理される変数`PATH` via *bin_path*、*LD_LIBRARY_PATH* via edb_tran_the9 では設定できません。 *env* フィールドに入力され、入場時に拒否されます。

CloudNativePG クラスターの EDB Postgres AI の標準の拡張コンテナイメージには、ルートに 2 つの必要なディレクトリが含まれます。

- `/share/` 拡張制御ファイルたとえば `<EXTENSION>.control` および対応する SQL ファイルを含む `extension` サブディレクトリが含まれています。
- `/lib/` 拡張機能の共有ライブラリたとえば `<EXTENSION>.so` およびその他の必要なライブラリが含まれています。

この構造に従うと、手動構成を必要とせずに、CloudNativePG クラスターの EDB Postgres AI 内の PostgreSQL で拡張機能が自動的に検出可能で使用できるようになります。

重要

PostgreSQL 拡張機能開発者とサードパーティプロバイダーは、このレイアウトに従って OCI 準拠の拡張機能イメージを公開することをお勧めします。実際の実装の詳細については、 CloudNativePG クラスター拡張イメージの公式 EDB Postgres AI をビルドするためのリファレンスとして機能する postgres-extensions-containers を確認することをお勧めします。

理想的には、拡張機能画像には次のことを行う必要があります。

- 特定のオペレーティングシステムディストリビューションと CPU アーキテクチャのセットをターゲットとします。
- 特定の PostgreSQL メジャーバージョンに関連付けられること。
- **ディストリビューションのネイティブパッケージシステムたとえば *.deb* または `.rpm` パッケージなどを使用してビルドして、クラスターで使用される PostgreSQL オペランドイメージとの一貫性、セキュリティ、および互換性を保証します。 .. ::**
  ##注意事項

現在、拡張機能イメージを追加、削除、または更新すると、PostgreSQL ポッドの再起動がトリガーされます。この動作は、image volumes の方法から継承されます

Kubernetes での作業。

拡張機能の更新を実行する前に、次のことを確認します。

- ステージング環境で更新プロセスを徹底的にテストしました。
- 拡張イメージには、現在インストールされているバージョンとターゲットバージョン間の必要なアップグレードパスが含まれていることを確認しました。
- 関連する `Database` リソース定義の拡張機能の `version` フィールドを更新して、イメージの新しいバージョンに合わせて調整します。

これらの手順は、拡張機能の更新中に PostgreSQL クラスターでのダウンタイムまたはデータの不一貫性を防ぐのに役立ちます。

5.38 Custom Pod Controller

Kubernetes は Controller pattern を使用します

をクリックして、現在のクラスターの状態を目的の状態に調整します。

ステートフルアプリケーションは、通常、StatefulSet で管理されます

コントローラ、同じ仕様から構築されたポッドのセットを作成および調整し、それらにスティッキー ID を割り当てます。

CloudNativePG クラスターの EDB Postgres AI は、 `StatefulSet` コントローラーに依存するのではなく、PostgreSQL インスタンスを管理するための独自のカスタムコントローラーを実装します。このデザインの選択により、実装がより複雑になる一方で、PostgreSQL クラスターのトポロジについて透過的でありながら、クラスターの管理方法についてより柔軟にオペレーターに提供されます。

デザイン領域の多くの選択と同様に、ものが異なると他の妥協が発生します。以下のセクションでは、このデザインの選択により、CloudNativePG クラスターの EDB Postgres AI の実装がより信頼性があり、理解しやすくなったと考えられるいくつかのポイントについて説明します。

5.38.1 PVC のサイズ変更

これは `StatefulSet` のよく知られた制限です。PVC のサイズ変更はサポートされていません。これはデータベースにとって不便です。ボリュームのサイズを変更するには、複雑な回避策が必要です。

対照的に、CloudNativePG クラスターの EDB Postgres AI は、構成されたストレージクラスを活用して基になる PVC を直接管理し、ストレージクラスがサポートしている場合は PVC のサイズ変更を処理できます。

5.38.2 プライマリインスタンスとレプリカ

`StatefulSet` コントローラは、1 つのテンプレートからポッドのセットを作成するように設計されています。PostgreSQL インスタンスごとに 1 つの `Pod` を使用することを考えると、2 種類のポッドがあります。

1. プライマリインスタンス 1 つだけ
2. レプリカマルチプル、オプショナル

この違いは、特定の操作に対して実行する正しい展開戦略を決定する際に関連します。

一部の操作は、最初にレプリカで実行し、次にプライマリで実行する必要がありますが、更新されたレプリカが新しいプライマリとして昇格した後にのみです。たとえば、別の PostgreSQL イメージバージョンを適用する場合、または `max_connections` treated specially by PostgreSQL because EDB Postgres AI for CloudNativePG Cluster uses hot standbyreplicas のような構成パラメーターを増加する場合。

その際、CloudNativePG クラスターの EDB Postgres AI は、シリアル番号だけでなく、PostgreSQL インスタンスのロールを考慮します。

場合によっては、オペレーターは逆のプロセスに従う必要があります。最初にプライマリで作業し、次にレプリカで作業します。例、max_connections を下げた場合。その場合、CloudNativePG クラスターの EDB Postgres AI は次のことを行います。

- 新しい設定をプライマリインスタンスに適用します
- 再起動します
- レプリカに新しい設定を適用します

StatefulSet コントローラはアプリケーションに依存しないため、PostgreSQL のネイティブレプリケーションテクノロジーに固有のこの動作を組み込むことができません。

5.38.3 PVC のコヒーレンス

PostgreSQL インスタンスは、複数の PVC で動作するように構成できます。これが、WAL ストレージを PGDATA から分離する方法です。

2 つのデータストアは、同時に使用されるため、PostgreSQL の観点から一貫性がある必要があります。インスタンスの WAL ストレージに対応する PVC を削除すると、PGDATA が保存されている PVC は使用できなくなります。

この動作は PostgreSQL に固有であり、StatefulSet コントローラには実装されていません - 後者はアプリケーション固有ではありません。

ユーザーが PVC をドロップした後、StatefulSet はそれを再作成するだけで、破損した PostgreSQL インスタンスが発生します。

CloudNativePG クラスターの EDB Postgres AI は、代わりに残りの PVC を使用不可として分類し、別のインスタンスの PVC の新しいペアの作成を開始してクラスターに正しく参加します。

5.38.4 ローカルストレージ、リモートストレージ、およびデータベースのサイズ

場合によっては、アップグレードを行うために Kubernetes ノードをダウンする必要がある場合があります。アップグレード後、アップグレード戦略によっては、更新されたノードが再び起動するか、新しいノードに置き換えられる場合があります。

使用できないノードが PostgreSQL インスタンスをホストしていたと仮定すると、データベースのサイズとクラウドインフラストラクチャによっては、次のいずれかのアクションを選択することができます。

1. ダウンしたノードにある PVC とポッドをドロップします。別の PVC からデータのクローンを作成する新しい PVC を作成します。その後、そのためのポッドをスケジュールします
2. ポッドをドロップし、別のノードでポッドをスケジュールし、そこから PVC をマウントします
3. ポッドと PVC をそのままにして、ノードがバックアップされるのを待ちます。

最初の解決策は、データベースのサイズが許可する場合に実用的で、目的の数のレプリカをすぐに戻すことができます。

2 番目の解決策は、ローカルノードのストレージを使用していない場合にのみ実行可能であり、別のホストでの PVC の再マウントは合理的な時間で可能ですあなたとあなたの組織のみが知っています。

3 番目のソリューションは、データベースが大規模で、パフォーマンスとデータの耐久性を最大化するためにローカルノードストレージを使用する場合に適しています。

CloudNativePG クラスターコントローラの EDB Postgres AI は、これらの戦略をすべて実装して、ユーザーがクラスターレベルで優先される動作を選択できるようにします。詳細については、*Kubernetes upgrade and maintenance* セクションを読んでください。

`StatefulSet` は汎用であるため、このレベルのカスタマイズはできません。

5.39 Networking

CloudNativePG クラスターの EDB Postgres AI は、基になる Kubernetes クラスターに必要な接続が既に設定されていることを前提としています。Kubernetes でのネットワーキングは、重要かつ拡張されたトピックです。詳細については、

Kubernetes documentation を参照してください。

クイックスタートガイドに従って、CloudNativePG クラスターの EDB Postgres AI をローカル KinD または K3d クラスターにインストールしている場合、どちらのプラットフォームもデフォルトでネットワーク制限を追加しないため、ネットワーク問題が発生することはありません。

ただし、既存のインフラストラクチャに CloudNativePG クラスターの EDB Postgres AI を展開する場合、ネットワーク制限が適用され、オペレーターと PostgreSQL クラスターとの通信を損なう可能性があります。具体的には、既存の Network Policies

特定の種類のトラフィックを制限する場合があります。

または、セキュリティを強化するために環境にネットワークポリシーを追加することに興味がある場合があります。*security document* で述べたように、オペレーターがポート 8000 および 5432 ですべてのクラスターポッドに到達でき、ポッドが相互に接続できることを確認してください。

5.39.1 オペレーターのクロス名前空間ネットワークポリシー

クイックスタートガイドに従うか、展開に helm チャートを使用すると、オペレーターが専用の名前空間デフォルトで `postgresql-operator-system` にインストールされます。クラスターは別の名前空間に作成することをお勧めします。

オペレーターはクラスターポッドに接続できる必要があります。これは、名前空間を越えたトラフィックを制限する `NetworkPolicy` がある場合、妨げられる可能性があります。

例、kubernetes guide on network policies

デフォルトですべての入力トラフィックを拒否するポリシーの例が含まれています。

ローカル kubernetes セットアップにこの種の制限的なネットワークポリシーがある場合、`NetworkPolicy` を作成して、オペレーター名前空間とポッドからクラスター名前空間とポッドへの接続を明示的に許可する必要があります。このリポジトリの networkpolicy-example.yaml ファイルに例があります。特定のセットアップに一致するようにクラスター名とクラスター名前空間を調整する必要があることに注意してください。デフォルトの名前空間でない場合は、オペレーター名前空間も調整する必要があります。

5.39.2 クロスクラスターネットワーキング

別のクラスターから データディレクトリのブートストラップ中のエラー を使用している間、または externalClusters セクションを使用している場合、関係するすべてのクラスター、オブジェクトストア、および名前空間間の接続を確認します。

繰り返しになりますが、Kubernetes documentation を参照してください。

セットアップ情報については。

networkpolicy-example.yaml

```yaml
#  This network policy allows ingress from the CNP operator

#  installed in namespace postgresql-operator-system into the pods corresponding

#  to a cluster named cluster-example

#  NOTE: in OpenShift the default namespace for the operator is openshift-operators

apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-operator
spec:
  podSelector:
    matchLabels:
      cnpg.io/cluster: cluster-example # The label value must be the cluster name
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              kubernetes.io/metadata.name: postgresql-operator-system # Or on␣
↪OpenShift: openshift-operators
          podSelector:
            matchLabels:
              app.kubernetes.io/name: cloud-native-postgresql # Matches the Operator␣
↪pod
      ports:
        - port: 8000
        - port: 5432
```

5.40 Benchmarking

CNP kubectl プラグインは、CloudNativePG クラスターの EDB Postgres AI を使用して、Kubernetes で PostgreSQL 展開をベンチマークする簡単な方法を提供します。

ベンチマークは、2 つの側面に焦点を当てています。

- **データベース** 、 pgbench に依存すること
- **ストレージ** fio に依存すること

5.40.1 pgbench

kubectl CNP プラグインコマンド pgbench は、既存の Postgres クラスターに対してユーザー定義の pgbench ジョブを実行します。

--dry-run フラグを介して、後の変更/実行のためにジョブのマニフェストを生成できます。

pgbench を使用した一般的なコマンド構造は次のとおりです。

```
kubectl cnp pgbench \
  -n <namespace> <cluster-name> \
  --job-name <pgbench-job> \
  --db-name <db-name> \
  -- <pgbench options>
```

ジョブで使用される特定のオプションについては、

この例では、 pgbench OLTP のような目的で、スケールファクター 1000 を使用して cluster-example という名前の Cluster の app データベースを初期化する pgbench-init というジョブを作成します。

```
kubectl cnp pgbench \
  --job-name pgbench-init \
  cluster-example \
  -- --initialize --scale 1000
```

> **注釈**
>
> これにより、100000000 レコードを含むデータベースが生成され、ディスク上に約 13GB の領域が必要です。

ジョブの進行状況は次の方法で確認できます。

```
kubectl logs jobs/pgbench-run
```

次の例では、単一の接続を使用して、以前に初期化されたデータベースに対して 30 秒間 pgbench を実行する pgbench-run というジョブを作成します。

```
kubectl cnp pgbench \
  --job-name pgbench-run \
  cluster-example \
  -- --time 30 --client 1 --jobs 1
```

次の例では、 --db-name フラグと pgbench 名前空間を使用して、既存のデータベースに対して pgbench を実行します。

```
kubectl cnp pgbench \
  --db-name pgbench \
  --job-name pgbench-job \
  cluster-example \
  -- --time 30 --client 1 --jobs 1
```

デフォルトでは、ジョブに有効期限はありません。 `--ttl` フラグを使用して自動削除を有効にできます。ジョブは指定された期間秒単位で削除されます。

```
kubectl cnp pgbench \
  --job-name pgbench-run \
  --ttl 600 \
  cluster-example \
  -- --time 30 --client 1 --jobs 1
```

特定のワーカーノードで pgbench ジョブを実行する場合は、`--node-selector` オプションを使用できます。 `workload=pgbench` ラベルを持つノードで以前の初期化ジョブを実行する場合、次のことを実行できます。

```
kubectl cnp pgbench \
  --db-name pgbench \
  --job-name pgbench-init \
  --node-selector workload=pgbench \
  cluster-example \
  -- --initialize --scale 1000
```

ジョブステータスは、次のコマンドを実行して取得できます。

```
kubectl get job/pgbench-job -n <namespace>

NAME       COMPLETIONS   DURATION   AGE
job-name   1/1           15s        41s
```

ジョブが完了すると、次のことを実行して結果を収集できます。

```
kubectl logs job/pgbench-job -n <namespace>
```

5.40.2 fio

kubectl CNP プラグインコマンド `fio` は、デフォルト値と読み取り操作を使用して fio ジョブを実行します。`--dry-run` フラグを介して、後の変更/実行のためにジョブのマニフェストを生成できます。

> **注釈**
>
> kubectl プラグインコマンド `fio` は、ConfigMap を使用して事前定義された fio ジョブ値を使用してデプロイメントを作成します。カスタムジョブ値を提供する場合は、 *--dry-run* フラグを使用してマニフェ

ストを生成し、生成された ConfigMap でカスタムジョブ値を提供することをお勧めします。

デフォルトの使用例

```
kubectl cnp fio <fio-name>
```

カスタム値を使用した例

```
kubectl cnp fio <fio-name> \
  -n <namespace>  \
  --storageClass <name> \
  --pvcSize <size>
```

fio 名前空間の standard および pvcSize: 2Gi という名前の StorageClass に対して fio コマンドを実行する方法の例

```
kubectl cnp fio fio-job \
  -n fio  \
  --storageClass standard \
  --pvcSize 2Gi
```

展開ステータスは、次のコマンドを実行して取得できます。

```
kubectl get deployment/fio-job -n fio

NAME          READY   UP-TO-DATE   AVAILABLE   AGE
fio-job        1/1     1            1           14s
```

kubectl プラグインコマンド fio を実行した後。

それは次のことを行います。

1. PVC を作成する
2. fio ジョブの構成を表す ConfigMap を作成する
3. 単一のポッドで構成される fio 展開を作成し、PVC で fio を実行し、ベンチマークの完了後にグラフを作成し、ウェブサーバーで生成されたファイルの提供を開始します。そのためには、fio-tools イメージを使用します。

デプロイメントによって作成されたポッドは、結果の提供を開始すると準備が整います。デプロイメントによって作成されるポッドのポートを転送できます

```
kubectl port-forward -n <namespace> deployment/<fio-name> 8000
```

次に、ブラウザを使用して http://localhost:8000/ に接続してデータを取得します。

PostgreSQL ワークロードをエミュレートするために、デフォルトの 8k ブロックサイズが選択されました。使用可能な IOPS の量を制限するディスクは、このパラメーターを変更すると、非常に異なるスループット値を示す場合があります。

以下は、専用の Kubernetes ノードにマウントされたローカルディスクへのシークエンシャル書き込みの図の例 1 時間ベンチマーク。

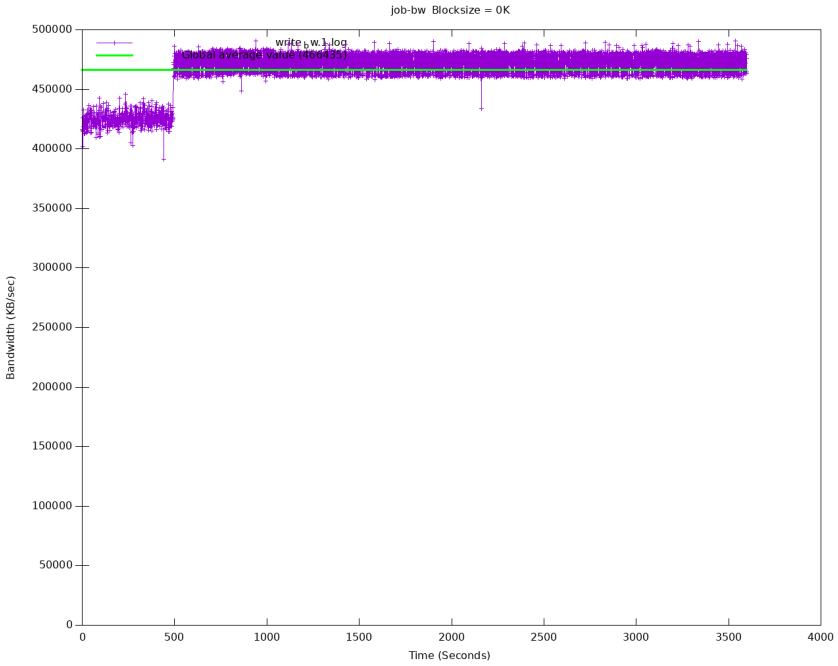


図 5: Sequential writes bandwidth

すべてのテストが完了したら、fio 展開とリソースを次の方法で削除できます。

```
kubectl cnp fio <fio-job-name> --dry-run | kubectl delete -f -
```

fio 展開の作成に使用したものと同じ名前を使用して、必要に応じて名前空間を追加します。

第6章　EDB ENHANCEMENTS

6.1 Free evaluation

CloudNativePG クラスターの EDB Postgres AI は、無料評価が利用できます。

EDB アカウントを使用して、CloudNativePG クラスターの EDB Postgres AI を評価します。アカウントをお持ちでない場合は、register をクリックします。次に、*installation guide* に従って、EDB アカウントから取得したアクセストークンを使用して、オペレーターをインストールします。

6.1.1 PostgreSQL を使用した評価

デフォルトでは、CloudNativePG クラスターの EDB Postgres AI は、Community Postgresql の最新バージョンをインストールします。

6.2 License and License keys

ライセンスキーは、CloudNativePG クラスターの EDB Postgres AI の古い管理メカニズムです。EDB サブスクリプショントークンを使用してインストールした場合、ライセンスキーは必要ありません。この場合、このセクションのライセンスコマンドは無視できます。

EDB サブスクリプショントークンを使用せず、パブリックリポジトリからインストールする場合、ライセンスキーが必要です。唯一の例外は、Community PostgreSQL でオペレーターを実行する場合です。この場合、ライセンスキーが設定されていない場合、クラスターはデフォルトの試用ライセンスで起動されます。30 日後に自動的に有効期限が切れます。これは、CloudNativePG クラスターの EDB Postgres AI をトライアルする推奨される方法ではありません。推奨されるオプションについては、*installation guide* を参照してください。

6.2.1 会社レベルのライセンスキー

ライセンスキーを使用すると、インストール内に無制限の数の PostgreSQL クラスターを作成できます。

ライセンスキーは、オペレーターがデプロイされている名前空間の `Secret` で利用できる必要があります。`ConfigMap` も利用できますが、ライセンスキーにはお勧めできません。

```
kubectl get cluster cluster_example -o yaml
[...]
status:
  [...]
  licenseStatus:
    licenseExpiration: "2021-11-06T09:36:02Z"
```

(次のページに続く)

(前のページからの続き)

```
    licenseStatus: Trial
    valid: true
    isImplicit: false
    isTrial: true
[...]
```

YAML マニフェストを介した Kubernetes のインストール

YAML マニフェストを使用して Kubernetes にオペレーターがインストールされると、デフォルトで postgresql-operator-system 名前空間にデプロイされます。

名前空間名とライセンスキーを指定して、次のコマンドを使用して構成マップを作成できます。

```
kubectl create configmap -n [NAMESPACE_NAME_HERE] \
    postgresql-operator-controller-manager-config \
    --from-literal=EDB_LICENSE_KEY=[LICENSE_KEY_HERE]
```

新しい構成を適用するには、オペレーターポッドを再作成する必要があります。次のコマンドを使用できます。

```
kubectl rollout restart deployment -n [NAMESPACE_NAME_HERE] \
    postgresql-operator-controller-manager
```

6.2.2 クラスターレベルのライセンスキー

各 Cluster リソースの定義に licenseKey パラメーターがあります。クラスターのステータスで、有効期限とライセンスの詳細情報を見つけることができます。

```
kubectl get cluster cluster_example -o yaml
[...]
status:
  [...]
  licenseStatus:
    licenseExpiration: "2021-11-06T09:36:02Z"
    licenseStatus: Trial
    valid: true
    isImplicit: false
    isTrial: true
[...]
```

クラスターライセンスキーは、新しいものに更新して、有効期限を延長したり、クラスターを運用ライセンスに移動したりできます。

6.2.3 クラスターレベルでのライセンスキーシークレット

各 Cluster リソースには、シークレットの名前とキーを含む licenseKeySecret パラメーターを持つこともできます。このシークレットには、EDB が提供するライセンスキーが含まれています。

このフィールドは licenseKey より優先されます。有効期限を延長するためにシークレットを変更したとき、または試用ライセンスから実稼働ライセンスに切り替えたときに更新されます。

CloudNativePG クラスターの EDB Postgres AI は、enterprisedb.com/legal/EDB-Eula で入手可能な EDB エンドユーザーライセンス契約に基づいて配布されています。

CloudNativePG クラスターの EDB Postgres AI Copyright (C) 2019-2022 EnterpriseDB Corporation.

6.2.4 ライセンスの有効期限が切れるとどうなるか

ライセンスの有効期限が切れると、オペレーターはクラスターでの調整の試行を停止し、事実上そのステータスの管理を停止します。これには、自動フェイルオーバーやスイッチオーバーなどの自己修復機能と高可用性機能も含まれます。

ポッドとデータは引き続き利用できます。

6.3 EDB private container registry

_EDB Postgres AI for CloudNativePG Cluster_オペレーターのイメージおよびさまざまなオペランドは、docker.enterprisedb.com の下のプライベートコンテナイメージレジストリに保持されます。

次の情報を収集します。

1. あなたの EDB account token
2. リポジトリの名前 k8s になります
3. リポジトリーサーバー docker.enterprisedb.com になります

明確にするために、次の例では、トークンが EDB_SUBSCRIPTION_TOKEN という名前の環境変数にあることを前提としています。

6.3.1 docker login を使用した例

ターミナルから Docker を介してログインできます。この文脈では、

- サーバーは docker.enterprisedb.com です
- ユーザー名はリポジトリ k8s です
- パスワードは EDB アカウントトークン $EDB_SUBSCRIPTION_TOKEN に保存されています

```
docker login docker.enterprisedb.com  \
  --username k8s \
  --password "$EDB_SUBSCRIPTION_TOKEN"
__OUTPUT__
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
Login Succeeded
```

オペランドイメージ

CloudNativePG クラスターの EDB Postgres AI は、同じプライベートレジストリから利用可能なイメージを持つさまざまな PostgreSQL ディストリビューションをサポートしています。

- コミュニティ PostgreSQL
- EDB Postgres Advanced EPAS
- EDB Postgres Extended PGE

これらのイメージは、 *Container Image Requirements* で説明されている要件と規則に従います。

EDB Postgres AI for CloudNativePG クラスタードキュメントのページ

以下の表では、各 Postgres ディストリビューションのイメージ名前の接頭辞を見つけることができます。

Postgres distribution	Image name
Community PostgreSQL	*postgresql*
EDB Postgres Extended (PGE)	*edb-postgres-extended*
EDB Postgres Advanced (EPAS)	*edb-postgres-advanced*

6.3.2 使用可能なイメージタグの参照

プライベートレジストリは OCI 準拠です。つまり、 *OCI Distribution Specification `list-tags* API <https://github.com/opencontainers/distribution-spec/blob/main/spec.md#tags>`_

使用できる Postgres ディストリビューションのバージョンを見つけるために、特定のイメージ名で使用可能なタグを参照します。

例、`edb-postgres-extended` イメージで使用可能なタグをリストするには

```
curl -s -u "k8s:$EDB_SUBSCRIPTION_TOKEN" \
  "https://docker.enterprisedb.com/v2/k8s/edb-postgres-extended/tags/list" | jq
```

URL の `edb-postgres-extended` は、上記の表にリストされているイメージ名に置き換えることができます。

> **注釈**
>
> *curl* に加えて、 crane などの OCI 準拠クライアントツールを使用できます。

または skopeo 、レジストリ内のタグを参照し、イメージを検査します。

EPAS または PGE オペランドを使用してクラスターを展開する方法

プライベートレジストリから CloudNativePG クラスターオペレーターの EDB Postgres AI を既にインストールしている場合、イメージプルシークレットを既に設定している必要があります。まだの方は、次のセクションに興味があるかもしれません。

オペレーターの既存のインストールがある場合、プライベートレジストリから EPAS または PGE のイメージをプルするには、kubernetes.io/dockerconfigjson を作成する必要があります。

create a pull secret from credentials ができます。

```
kubectl create secret docker-registry registry-pullsecret \
  -n <CLUSTER-NAMESPACE> --docker-server=docker.enterprisedb.com \
  --docker-username=k8s \
  --docker-password="$EDB_SUBSCRIPTION_TOKEN"
```

前述のように、docker-username はレジストリの名前つまり k8s です。 docker-password は、EDB portal から取得したトークンです。

プルシークレットが作成されたら、クラスターマニフェストの imageName に加えて imagePullSecrets フィールドを忘れずに設定します。以下のマニフェストは、 PG Extended を実行するクラスターを作成します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: postgresql-extended-cluster
spec:
  instances: 3
  imageName: docker.enterprisedb.com/k8s/edb-postgres-extended:18.1-standard-ubi9
  imagePullSecrets:
    - name: registry-pullsecret

  storage:
    storageClass: standard
    size: 1Gi
```

EDB プライベートレジストリを使用してオペレーターをインストールする方法

前述のように、docker の_username_はリポジトリの名前であり、トークンは_password_です。kubernetes は、kubernetes.io/dockerconfigjson を設定することにより、同じ資格情報を使用してレジストリにアクセスできます。

installation document で述べたように、オペレーターをインストールするにはいくつかの異なる方法があります。

プライベートレジストリからのイメージを使用してインストールする場合は、前述のようにプルシークレットを作成し、展開マニフェストの OPERATOR_IMAGE_NAME パラメーターをカスタマイズする必要があります。

インストールには Helm チャートを使用することをお勧めします。これにより、プルシークレットの作成とオペレーターイメージリポジトリのカスタマイズが行われます。

詳細については、 Helm chart ページをご覧ください。

例として、次のコマンドトークンを提供すると、EDB EnterpriseDB プランのリポジトリを使用する場合、{{name.abbr}}オペレーターをインストールします。

```
helm upgrade --install edb-pg4k \
  --namespace postgresql-operator-system \
  --create-namespace \
  --set image.repository=docker.enterprisedb.com/k8s/edb-postgres-for-cloudnativepg \
  --set image.imageCredentials.username=k8s \
  --set image.imageCredentials.password="$EDB_SUBSCRIPTION_TOKEN" \
  --set image.imageCredentials.create=true \
  --set "imagePullSecrets[0].name"=postgresql-operator-pull-secret \
  --set config.data.PULL_SECRET_NAME=postgresql-operator-pull-secret \
  edb/edb-postgres-for-kubernetes
```

6.4 Red Hat OpenShift

CloudNativePG クラスターの EDB Postgres AI は Red Hat OpenShift Container Platform (OCP) version 4.x で実行することが認定されています

Red Hat Catalog から直接利用できます。

このセクションの目的は、組織のセキュリティとアクセス制御ポリシーに基づいて、CloudNativePG クラスターの EDB Postgres AI の最適なインストール方法を決定するのを助けることです。

最初の重要なステップは、OpenShift 環境での PostgreSQL クラスターの *Architecture* をデザインすることです。

アーキテクチャが明確になったら、インストールを続行できます。CloudNativePG クラスターの EDB Postgres AI は、次の方法でインストールおよび管理できます。

- *Web* コンソールを介したインストール
- フル制御の場合、oc と呼ばれる OpenShift command-line interface (CLI)

CloudNativePG クラスターの EDB Postgres AI は、OpenShift で定義された使用可能なインストールモードをすべてサポートしています。

- クラスタ全体、すべての名前空間
- ローカル、単一の名前空間
- ローカル、複数の名前空間を監視する oc を使用してのみ使用可能

注釈

プロジェクトは、追加のアノテーションを備えた Kubernetes 名前空間であり、一般のユーザーのリソースへのアクセスが管理される中心的な手段です。

および以下の **ユーザーと権限**)。

Default cluster roles より、*cluster-admin* は_"任意のプロジェクトで任意のアクションを実行できるスーパーユーザー。ローカルバインディングでユーザーにバインドされると、プロジェクト内のすべてのリソースのクォータとすべてのアクションを完全に制御できます。"_。

汎用 *Kubernetes/PostgreSQL architecture page* に含まれているのと同じ概念

同様に OpenShift を適用します。

ここでも、重要な要素は、OpenShift 環境のアベイラビリティーゾーンまたはデータセンターの数です。

Disaster Recovery Strategies for Applications Running on OpenShift で概説されている

ステートフルアプリケーションについて 2020 年に Raffaele Spazzoli が書いたブログ記事、CloudNativePG クラスターの EDB Postgres AI を完全に活用するには、3 つ以上のアベイラビリティーゾーンにまたがる OpenShift クラスターを計画、設計、実装する必要があります。これは、ほとんどのパブリッククラウドプロバイダーの展開では問題を引き起こすことはありませんが、オンプレミスのシナリオでは間違いなく課題です。

OpenShift クラスターに **1 つのアベイラビリティーゾーン**のみの場合、そのゾーンは高可用性の観点からの単一ポイント障害 SPoF です。シェアナッシングアーキテクチャを賢明に採用し、PostgreSQL クラスターが次のことを確認している場合。少なくとも 1 つのスタンバイ同期レプリケーションを使用する場合は 2 つ、および各 PostgreSQL インスタンスは別のストレージを使用する別の Kubernetes ワーカーノードで実行されていること。継続的なバックアップデータが OpenShift クラスターの外部のストレージサービスに追加で保存されることを確認して、データセンターを超えてディザスターリカバリー操作を実行できます。

ほとんどの場合、アクティブ/パッシブ戦略では、同じ大都市圏または別のリージョンの別のデータセンターに別の OpenShift クラスターが存在します。特集ページで説明しているように、これは主にディザスターリカバリーソリューションであることを理解した上で、独立した **レプリカクラスター** をセットアップできます。非常に効果的ですが、手動介入を必要とするいくつかの制限があります。カスケード方法でも、同じソリューションを追加の OpenShift クラスターに適用できます。

一方、OpenShift クラスターがリージョン内の**マルチプルのアベイラビリティーゾーン**にまたがる場合、復元力と自己修復のためのオペレーターの機能を完全に活用でき、リージョンは SPoF になることができます。つまり、リージョン全体を停止して、クラスターをダウンさせます。さらに、前述のように、レプリカクラスターを設定することにより、さまざまなリージョンの複数の OpenShift クラスターを利用できます。

6.4.1 PostgreSQL ワークロードのノードの予約

PostgreSQL データベース操作で最適なパフォーマンスとリソース割り当てを得るには、本番で特定のワーカーノードを `postgres` 専用に割り当てて、PostgreSQL ワークロードを分離することを強くお勧めします。これは、単一のアベイラビリティーゾーンで動作しているか、マルチアベイラビリティーゾーン環境で動作しているかにかかわらず、特に重要です。

PostgreSQL ワークロードの実行専用の OpenShift のワーカーノードは、通常 **Postgres ノード** または `postgres` ノードと呼ばれます。

この専用のアプローチにより、PostgreSQL ワークロードが他のアプリケーションとリソースを競合しないことが保証され、安定性とパフォーマンスの向上につながります。

詳細については、 *PostgreSQL* ワークロードのノードの予約 を参照してください。

ドキュメント。OpenShift で作業する場合の主な違いは、以下で説明するように、ラベルとテイントがノードに適用される方法に関係します。

ノードを `postgres` ノードとしてラベル付けするには、次のコマンドを実行します。

```
oc label node <NODE-NAME> node-role.kubernetes.io/postgres=
```

`postgres` テイントをノードに適用するには、次のコマンドを使用します。

```
oc adm taint node <NODE-NAME> node-role.kubernetes.io/postgres=:NoSchedule
```

ノードを正しくラベル付けとテインメントすることにより、アフィニティと寛容を介してこれらの専用ノードで PostgreSQL ワークロードのみがスケジュールされ、データベース環境の安定性とパフォーマンスが強化されます。

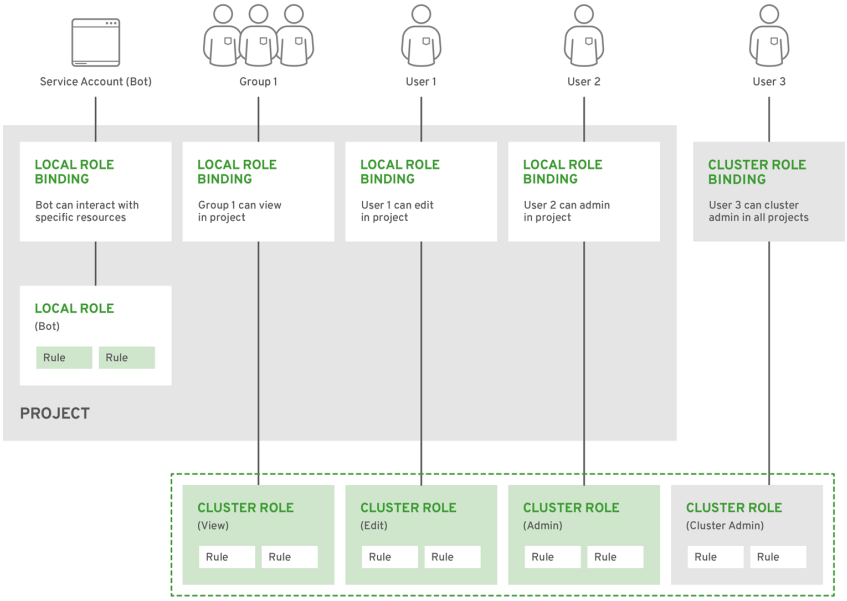
OpenShift の重要な概念

CloudNativePG クラスターオペレーターの EDB Postgres AI が OpenShift 環境にどのように適合するかを理解するには、次の Kubernetes 関連のトピックに精通する必要があります。

- オペレーター
- 認証
- ロールベースのアクセス制御 RBAC を介した承認
- サービスアカウントとユーザー
- ルール、ロール、およびバインディング
- プロジェクトを介したクラスター RBAC とローカル RBAC

これは、オペレーターのデフォルトのクラスター全体のインストールで必要とされる昇格した権限に慣れていない場合に特に当てはまります。

また、クラスターロール、ローカルロール、クラスターロールバインディング、ローカルロールバインディング、ユーザー、グループ、サービスアカウント間の関係を明確に示しているため、OpenShift ドキュメントから以下の図を選択しました。



OPENSHIFT_415489_0218 以下の *Predefined RBAC objects* には、CloudNativePG クラスターの EDB Postgres AI が Kubernetes および OpenShift RBAC 実装にどのように準拠しているかに関する重要な情報が含まれており、デフォルトでインストールされているクラスターロール、ロール、サービスアカウントをカバーしています。

上記の概念に精通している場合は、選択したインストール方法に直接進むことができます。それ以外の場合、OpenShift ドキュメントと Red Hat ブログから取得した次のリソースを読むことをお勧めします。

-

Operator Lifecycle Manager (OLM) concepts and resources

-

Understanding authentication

-

Role-based access control (RBAC)
、承認のルール、ロール、バインディング、およびプロジェクトを介したクラスター RBAC とローカル RBAC をカバー

-

Default project service accounts and roles

- With Kubernetes Operators comes great responsibility

6.4.2 クラスターサービスバージョン CSV

技術的には、オペレーターは、EDB によって定義されたクラスターサービスバージョン CSV に従って、Operator Lifecycle ManagerOLM を介して OpenShift で実行されるように設計されています。

CSV は、ユーザーインターフェイス Web ダッシュボードから利用可能だけでなく、オペレーターが必要とする RBAC ルール、およびオペレーターが定義および所有するカスタムリソースたとえば `Cluster` などを定義する YAML マニフェスト。CSV は、オペレーターの使用可能な `installModes` 、つまり `AllNamespaces`

クラスター全体、SingleNamespace シングルプロジェクト、MultiNamespace マルチプロジェクト、および OwnNamespace も定義します。

Operator group membership を読むと、CSV とインストールモードの詳細を見つけることができます

と Defining cluster service versions (CSVs)

OpenShift ドキュメントより。

Red Hat OpenShift Container Platform は、単一のクラスターにオペレーターのさまざまなバリエーションを同時にインストールすることに対する限定的なサポートを提供します。他のオペレーターと同様に、CloudNativePG クラスターの EDB Postgres AI は、コントロールプレーンの拡張機能になります。コントロールプレーンは OpenShift クラスターのすべてのテナントプロジェクトの間で共有されるため、オペレーターもマルチテナント環境では共有リソースになります。

Operator Lifecycle ManagerOLM は、異なる名前空間にオペレーターを複数回インストールできますが、1 つの重要な制限があります。すべてが同じ API バージョンのオペレーターを共有する必要があります。

詳細は、Operator groups を参照してください。

OpenShift ドキュメント。

チャンネル

CloudNativePG クラスターの EDB Postgres AI は、次の OLM channels を介して配布されます。それぞれが明確な目的を果たします。

- candidate このチャンネルは、次の潜在的な fast リリースへの早期アクセスを提供します。これには、新機能と修正を含む最新のプレリリースバージョンが含まれていますが、実験的であると考えられており、サポートされていません。このチャネルは、テストと検証の目的にのみ使用します。**運用環境ではない** 。 candidate のバージョンは、それ以上の更新が推奨されない場合、他のチャンネルに表示されない場合があります。
- fast 最新の安定した機能とパッチにタイムリーにアクセスしたいユーザー向けに設計されています。fast チャネルのヘッドは、常に Kubernetes 用 EDB Postgres の最新の**マイナーリリース** の最新の **パッチリリース** を指します。
- stable fast に似ていますが、現在 EDB の長期サポート LTS ポリシーのの下で最新の**マイナーリリース**に制限されています。継続的な安定性とメンテナンスの恩恵を受けながら、予測可能な更新と公式サポートを必要とするユーザー向けに設計されています。
- stable-vX.Y 特定のマイナーバージョンたとえば stable-v1.26 内の最新のパッチリリースを追跡します。これらのチャネルは、安定したマイナーリリース内のバージョンの固定と予測可能な更新を必要とする環境に最適です。

fast および stable チャンネルは複数のマイナーバージョンにまたがる場合がありますが、各 stable-vX.Y チャンネルは特定のマイナーリリース内のパッチ更新に制限されています。

CloudNativePG クラスターの EDB Postgres AI は、_トランクベースの開発_および_継続的デリバリー_の原則に従います。その結果、一般的には fast チャンネルを使用して、最新の安定した改善と修正で最新の状態を保つことをお勧めします。

Web コンソールを介したインストール

6.4.3 EDB プライベートレジストリへのアクセスの保証

プライベートリポジトリへの資格情報を取得したら、 openshift-operators 名前空間に次の名前のプルシークレットを作成する必要があります。

- postgresql-operator-pull-secret 、CloudNativePG クラスターオペレーターイメージの EDB Postgres AI 用

Get your token で説明しているように、 <TOKEN> を EDB アカウントのリポジトリトークンに置き換えることにより、 oc create コマンドを使用してこのシークレットを作成できます。

```
oc create secret docker-registry postgresql-operator-pull-secret \
  -n openshift-operators \
  --docker-server=docker.enterprisedb.com \
  --docker-username=k8s \
  --docker-password="<TOKEN>"
```

CloudNativePG クラスターオペレーターの EDB Postgres AI は、OpenShift ダッシュボードから Red Hat OperatorHub で直接見つけることができます。

1. Web コンソールで Operators -> OperatorHub ページに移動します。

1. Database セクションをスクロールするか、Filter by keyword ボックスこの場合は「PostgreSQL」にキーワードを入力して、CloudNativePG クラスターオペレーターの EDB Postgres AI を見つけて、選択します。

1. Operator に関する情報を読み、 Install を選択します。

2. 次の Operator installation ページでは、以下を選択することを想定しています。

    - インストールモード クラスター全体のインストール または Single project installation installation
    - 更新チャンネル詳細については チャンネル を参照してください-不明な場合は fast を選択します
    - 承認戦略、マーケットプレイスでの入手可能性に従って Red Hat によって認定されたオペレーターの新しいリリース
    - Automatic OLM は、実行中のオペレーターを新しいバージョンに自動的にアップグレードします
    - Manual Installed Operators セクションで承認を必要とすることにより、OpenShift は人間の介入を待機します

edb_excla_notran_14 edb_excla_notran_15 If you are running with OpenShift 4.20 or later, OperatorHub has been integrated into the Software Catalog. In the web console, navigate to Operators -> Software Catalog and select a Project to view the software catalog.

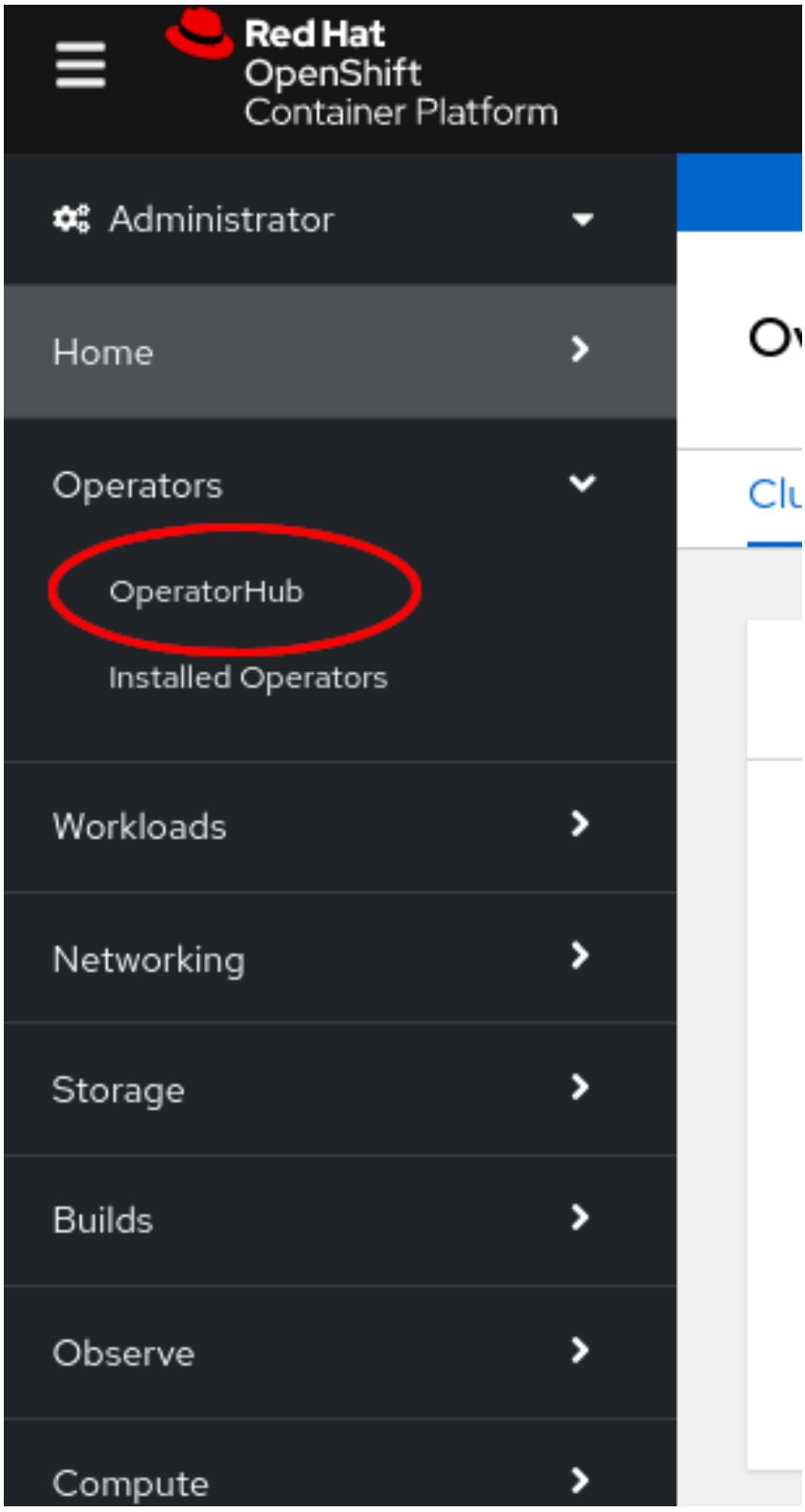


図 1: Menu OperatorHub

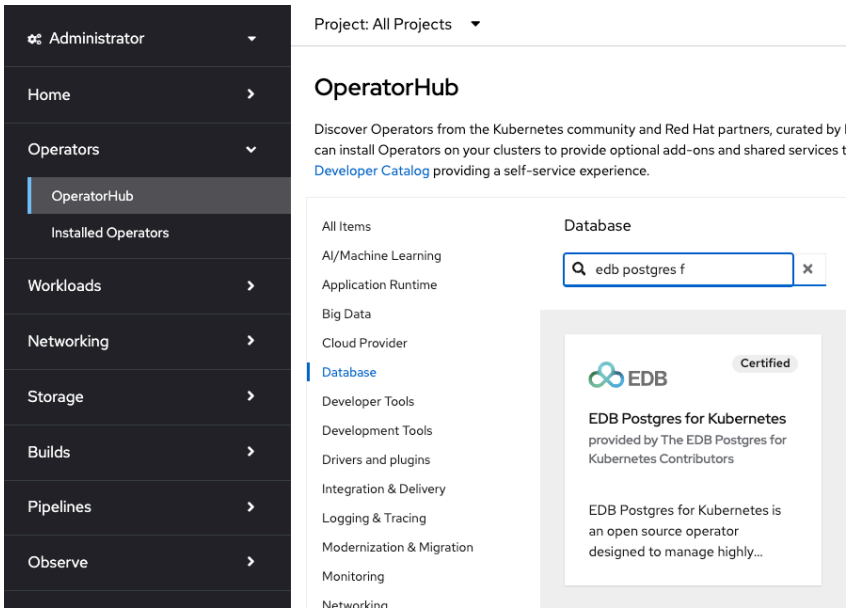
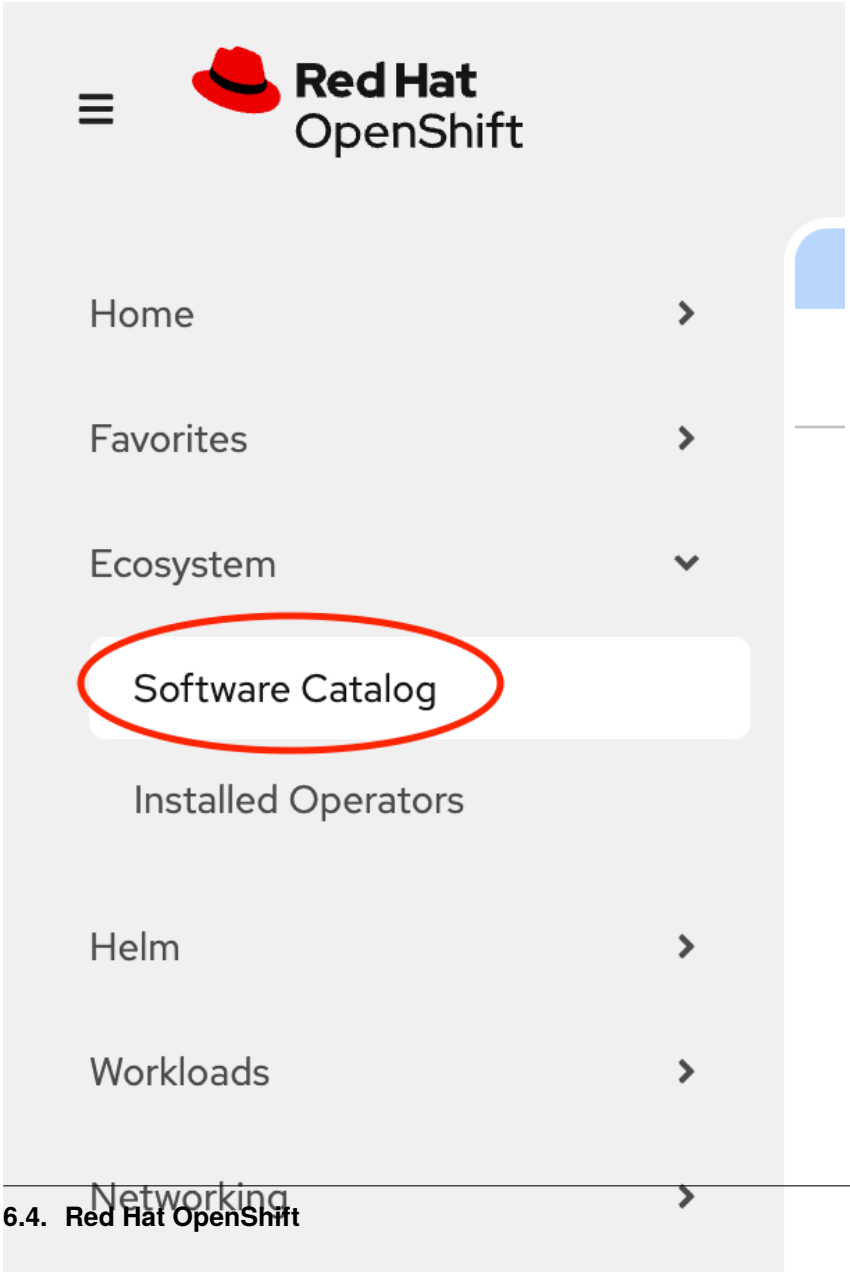


図 2: Install OperatorHub



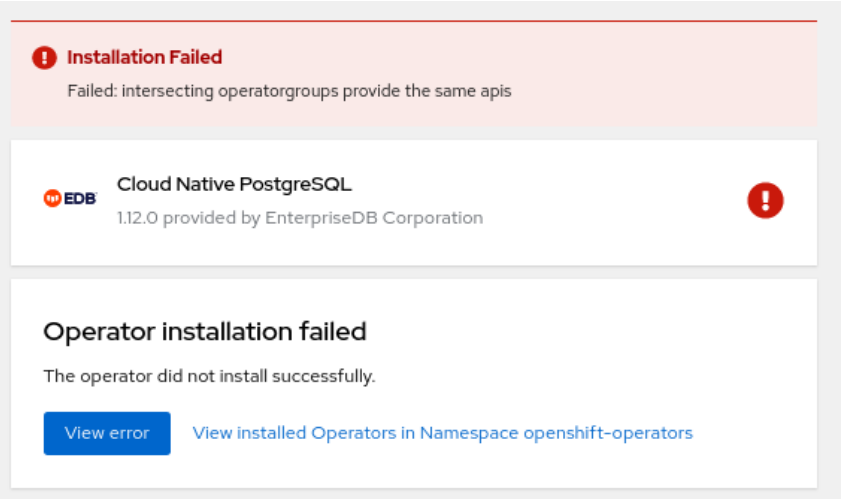


図 3: Install failing

6.4.4 クラスター全体のインストール

クラスター全体のインストールでは、OpenShift にオペレーターをデフォルトの `openshift-operators` 名前空間にインストールし、クラスター内のすべてのプロジェクトで利用できるようにします。これは、CloudNativePG クラスターの EDB Postgres AI をインストールするデフォルトの通常推奨されるアプローチです。

警告

これは、OpenShift クラスター内のすべてのユーザーが CloudNativePG クラスターオペレーターの EDB Postgres AI を使用でき、*Cluster* オブジェクトをデプロイ、または独自の名前空間で実行されている `Cluster` オブジェクトを表示できることを意味するわけではありません。 CloudNativePG クラスターの管理対象カスタムリソース主に `Cluster` の EDB Postgres AI と対話するためには、ユーザーが名前空間に必要な特別なロールがいくつかあります。詳細は **ユーザーと権限** を参照してください。

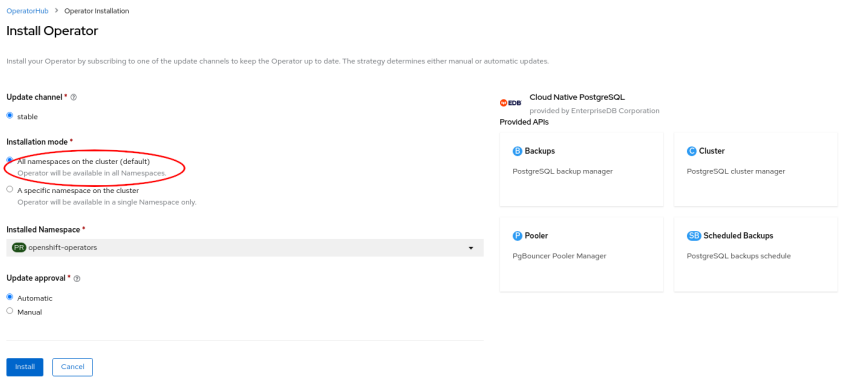


図 4: Install all namespaces

その結果、演算子はすべての名前空間で可視されます。それ以外の場合は、他の OpenShift オペレーターと同様に、さらにトラブルシューティングを行う問題を報告している `Workloads` → `Pods` ページの `openshift-operators` プロジェクトのポッドのログを確認します。

単一プロジェクトのインストールでは、OpenShift に Operator を特定の名前空間にインストールし、そのプロジェクトのみで使用できるようにします。

> **警告**
>
> これは、名前空間内のすべてのユーザーが CloudNativePG クラスターオペレーターの EDB Postgres AI を使用できること、*Cluster* オブジェクトを展開、または名前空間で実行されている `Cluster` オブジェクトを表示できることを意味するわけではありません。クラスター全体のインストールモードと同様に、CloudNativePG クラスターの管理対象カスタムリソース主に `Cluster` の EDB Postgres AI と対話するためには、ユーザーが名前空間に必要な特別なロールがいくつかあります。 **ユーザーと権限** を参照してください。

詳細については。

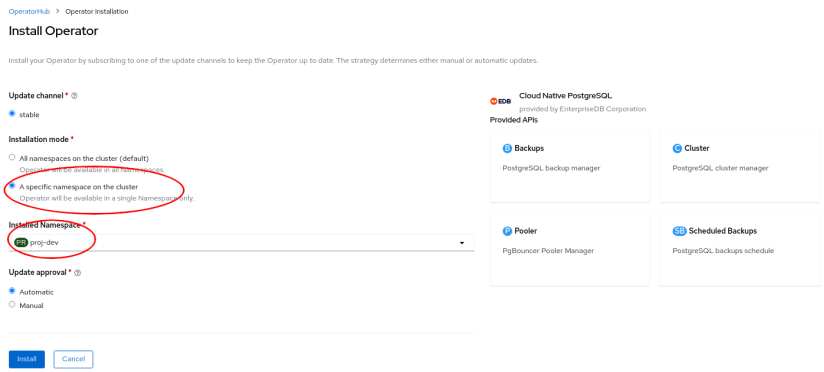


図 5: Install one namespace

その結果、演算子は選択した名前空間でのみ可視されます。これは、`Installed operators` ページから確認できます。

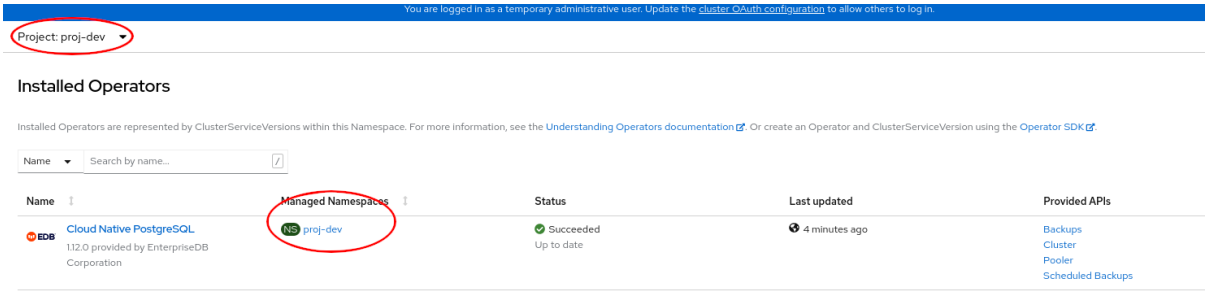


図 6: Install one namespace list

問題が発生した場合は、 `Workloads` → `Pods` ページから、さらにトラブルシューティングを行う問題を報告している、選択したインストール名前空間のポッドのログを確認します。

oc CLI を介したインストール

oc コマンドラインインターフェイスのインストール方法については、

> **警告**
>
> 複数の名前空間のインストールは、現在 OpenShift でサポートされています。ただし、

definition of multiple target namespaces for an operator may be removed in future versions of OpenShift .

> **注釈**
>
> この例では、オペレーターを `my-operators` 名前空間にインストールし、*web-staging* 、*web-prod* 、*bi-staging* 、および `bi-prod` 名前空間でのみ使用できるようにします。プロジェクトの名前を自由に変更するか、名前空間を追加/削除します。

```
oc get packagemanifests -n openshift-marketplace cloud-native-postgresql
```

1. オペレーターを検査して、インストールモード特に `MultiNamespace` と使用可能なチャネルを確認します。

```
oc describe packagemanifests -n openshift-marketplace cloud-native-postgresql
```

1. `my-operators` 名前空間に `OperatorGroup` オブジェクトを作成して、`web-staging` 、`web-prod` 、`bi-staging` 、および `bi-prod` 名前空間をターゲットとするようにします。

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: cloud-native-postgresql
  namespace: my-operators
spec:
  targetNamespaces:
  - web-staging
  - web-prod
  - bi-staging
  - bi-prod
```

Target namespace selection で説明しているように、ラベルセレクターを使用して名前空間をリストすることもできます。 .. Note Important::

1. *my-operators* 名前空間に `Subscription` オブジェクトを作成して、*openshift-marketplace certified-operators* ソースで使用可能な `cloud-native-postgresql` 演算子の `fast` チャネルをサブスクライブします 前に手順 1 および 2 で見つけたとおり。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
```

(次のページに続く)

(前のページからの続き)

```
metadata:
  name: cloud-native-postgresql
  namespace: my-operators
spec:
  channel: fast
  name: cloud-native-postgresql
  source: certified-operators
  sourceNamespace: openshift-marketplace
```

1. OperatorGroup および Subscription オブジェクトの上記の YAML ファイル定義で oc apply -f を使用します。

このセクションで説明する方法は、CloudNativePG クラスターの事前定義された ClusterRole s の EDB Postgres AI を選択した名前空間の特定のユーザーにマッピングできるため、適切な RoleBinding オブジェクトと組み合わせて非常に強力です。

注釈

上記の手順は、単一のプロジェクトのバインドに使用することもできます。唯一の違いは、指定されたターゲット名前空間の数 1 つと、場合によっては演算子グループの名前空間理想的にはターゲット名前空間と同じです。

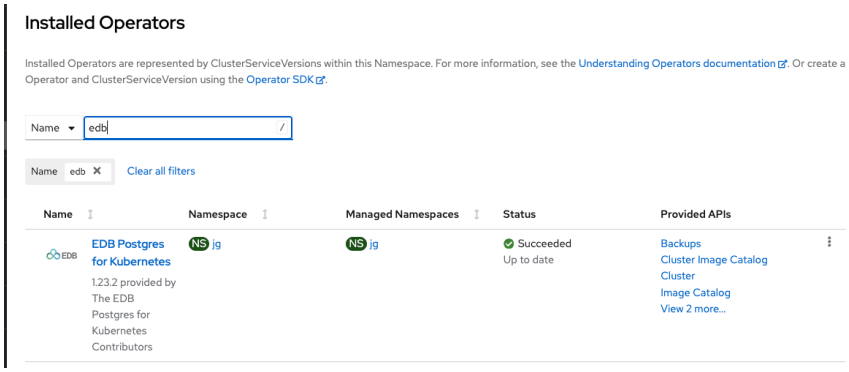


図 7: Multi namespace installation displayed in the web console

6.4.5 oc を使用したクラスター全体のインストール

必要に応じて、 oc を使用してオペレーターをグローバルにインストールし、openshift-operators 名前空間の global-operators と呼ばれるデフォルトの OperatorGroup を利用し、同じ名前空間の cloud-native-postgresql オペレーターの新しい Subscription オブジェクトを作成することもできます。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
```

(次のページに続く)

(前のページからの続き)

```
  name: cloud-native-postgresql
  namespace: openshift-operators
spec:
  channel: fast
  name: cloud-native-postgresql
  source: certified-operators
  sourceNamespace: openshift-marketplace
```

上記の YAML ファイルで `oc apply -f` を実行すると、すべての名前空間で演算子を使用できるようになります。

6.4.6 OpenShift CLI oc のインストール

oc コマンドは、OpenShift コマンドラインインターフェイス CLI を表します。システムにインストールすることを強くお勧めします。以下に、OpenShift ダッシュボードから oc をインストールするための基本的な手順を示します。

最初に、ダッシュボードの右上隅にあるクエスチョンマークを選択します。

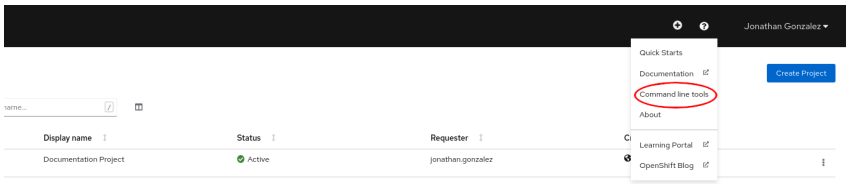


図 8: Command Line Tools menu

次に、表示される指示に従って、オペレーティングシステムとアーキテクチャの点でニーズに合ったバイナリをダウンロードします。

OpenShift CLI documentation を参照してください。

Red Hat によって直接保守されています。

CloudNativePG クラスターの EDB Postgres AI には、RBAC ポリシー構成に関して重要な役割を果す事前定義されたリソースのセットが付属しています。

6.4.7 カスタムリソース定義 CRD

CloudNativePG クラスターオペレーターの EDB Postgres AI は、次のカスタムリソース定義 CRD を所有しています。

- `Backup`
- `Cluster`
- `Pooler`
- `ScheduledBackup`

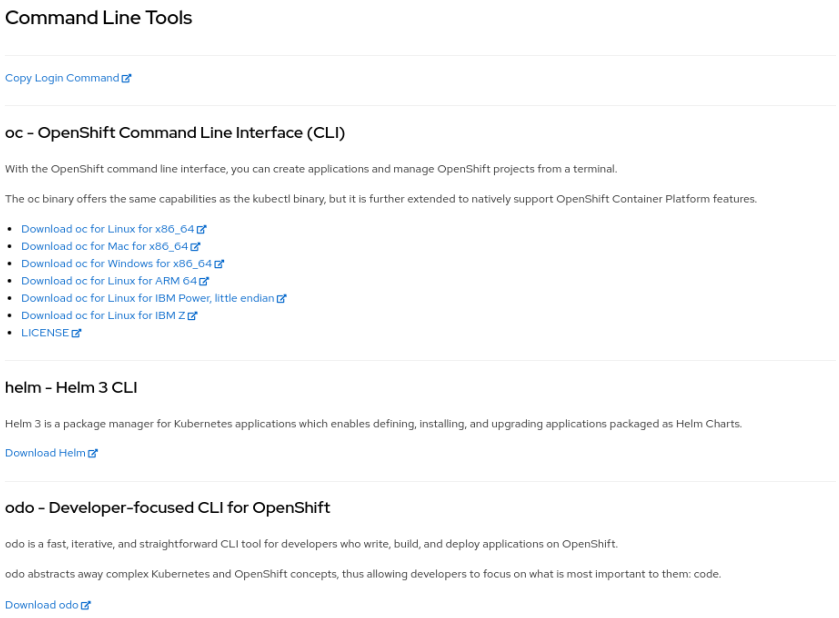


図 9: Command Line Tools

- `ImageCatalog`
- `ClusterImageCatalog`

これは、次を実行して確認できます。

```
oc get customresourcedefinitions.apiextensions.k8s.io | grep postgresql
```

これは、次のようなものを返します。

```
backups.postgresql.k8s.enterprisedb.io                    20YY-MM-DDTHH:MM:SSZ
clusterimagecatalogs.postgresql.k8s.enterprisedb.io       20YY-MM-DDTHH:MM:SSZ
clusters.postgresql.k8s.enterprisedb.io                   20YY-MM-DDTHH:MM:SSZ
imagecatalogs.postgresql.k8s.enterprisedb.io              20YY-MM-DDTHH:MM:SSZ
poolers.postgresql.k8s.enterprisedb.io                    20YY-MM-DDTHH:MM:SSZ
scheduledbackups.postgresql.k8s.enterprisedb.io           20YY-MM-DDTHH:MM:SSZ
```

6.4.8 サービスアカウント

オペレーターがインストールされている名前空間デフォルトでは `openshift-operators` には、次の事前定義されたサービスアカウントが含まれています。`builder` 、`default` 、`deployer` 、そして最も重要な `postgresql-operator-manager` CSV によって管理されます。

```
oc get serviceaccounts -n openshift-operators
```

これは、次のようなものを返します。

```
NAME                          SECRETS   AGE
builder                       2         ...
default                       2         ...
deployer                      2         ...
postgresql-operator-manager   2         ...
```

default サービスアカウントは、Kubernetes によって自動的に作成され、すべての名前空間に存在します。builder および deployer サービスアカウントは、OpenShift によって自動的に作成されます

Default project service accounts and roles を参照してください。

postgresql-operator-manager サービスアカウントは、クラウドネイティブの PostgreSQL クラスターを管理する際に Kubernetes / OpenShift コントロールプレーンの一部として動作するために使用されるサービスアカウントです。

Operator Lifecycle ManagerOLM は、クラスターロールオブジェクトのセットを自動的に作成して、ロールバインディング定義と RBAC ポリシーの詳細な実装を促進します。一部のクラスターロールには、CloudNativePG クラスターの EDB Postgres AI の一部であるカスタムリソース定義に適用されるルールがありますが、より広範な Kubernetes / OpenShift レルムの一部であるものもあります。

6.4.9 CloudNativePG™クラスター CRD の EDB Postgres AI のクラスターロール

every CRD owned by EDB Postgres AI for CloudNativePG Cluster' CSV の場合、OLM は、顧客対応ユーザーおよびサービスアカウントが使用できるいくつかの事前定義されたクラスターロールを展開します。特に

- リソースの完全な管理のためのロール admin サフィックス
- リソースを編集するロール edit サフィックス
- リソースを表示するロール view サフィックス
- 実際の CRD crdview サフィックスを表示するロール

```
oc get clusterroles | grep postgresql
```

これは、次のようなものを返します。

```
backups.postgresql.k8s.enterprisedb.io-v1-admin                  YYYY-MM-DDTHH:MM:SSZ
backups.postgresql.k8s.enterprisedb.io-v1-crdview                YYYY-MM-DDTHH:MM:SSZ
backups.postgresql.k8s.enterprisedb.io-v1-edit                   YYYY-MM-DDTHH:MM:SSZ
backups.postgresql.k8s.enterprisedb.io-v1-view                   YYYY-MM-DDTHH:MM:SSZ
cloud-native-postgresql.VERSION-HASH                             YYYY-MM-DDTHH:MM:SSZ
clusterimagecatalogs.postgresql.k8s.enterprisedb.io-v1-admin     YYYY-MM-DDTHH:MM:SSZ
clusterimagecatalogs.postgresql.k8s.enterprisedb.io-v1-crdview   YYYY-MM-DDTHH:MM:SSZ
clusterimagecatalogs.postgresql.k8s.enterprisedb.io-v1-edit      YYYY-MM-DDTHH:MM:SSZ
clusterimagecatalogs.postgresql.k8s.enterprisedb.io-v1-view      YYYY-MM-DDTHH:MM:SSZ
clusters.postgresql.k8s.enterprisedb.io-v1-admin                 YYYY-MM-DDTHH:MM:SSZ
clusters.postgresql.k8s.enterprisedb.io-v1-crdview               YYYY-MM-DDTHH:MM:SSZ
clusters.postgresql.k8s.enterprisedb.io-v1-edit                  YYYY-MM-DDTHH:MM:SSZ
```

(次のページに続く)

(前のページからの続き)

```
clusters.postgresql.k8s.enterprisedb.io-v1-view                    YYYY-MM-DDTHH:MM:SSZ
imagecatalogs.postgresql.k8s.enterprisedb.io-v1-admin              YYYY-MM-DDTHH:MM:SSZ
imagecatalogs.postgresql.k8s.enterprisedb.io-v1-crdview            YYYY-MM-DDTHH:MM:SSZ
imagecatalogs.postgresql.k8s.enterprisedb.io-v1-edit               YYYY-MM-DDTHH:MM:SSZ
imagecatalogs.postgresql.k8s.enterprisedb.io-v1-view               YYYY-MM-DDTHH:MM:SSZ
poolers.postgresql.k8s.enterprisedb.io-v1-admin                    YYYY-MM-DDTHH:MM:SSZ
poolers.postgresql.k8s.enterprisedb.io-v1-crdview                  YYYY-MM-DDTHH:MM:SSZ
poolers.postgresql.k8s.enterprisedb.io-v1-edit                     YYYY-MM-DDTHH:MM:SSZ
poolers.postgresql.k8s.enterprisedb.io-v1-view                     YYYY-MM-DDTHH:MM:SSZ
scheduledbackups.postgresql.k8s.enterprisedb.io-v1-admin           YYYY-MM-DDTHH:MM:SSZ
scheduledbackups.postgresql.k8s.enterprisedb.io-v1-crdview         YYYY-MM-DDTHH:MM:SSZ
scheduledbackups.postgresql.k8s.enterprisedb.io-v1-edit            YYYY-MM-DDTHH:MM:SSZ
scheduledbackups.postgresql.k8s.enterprisedb.io-v1-view            YYYY-MM-DDTHH:MM:SSZ
```

get コマンドを使用して、他の Kubernetes リソースと同様に、実際のロールを検査できます。例

```
oc get -o yaml clusterrole clusters.postgresql.k8s.enterprisedb.io-v1-admin
```

以下の関連するスキミング出力を見ると、clusters.postgresql.k8s.enterprisedb.io-v1-admin クラスターロールが、 postgresql.k8s.enterprisedb.io API グループで定義された cluster リソースのすべてを有効にしていることがわかります。

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: clusters.postgresql.k8s.enterprisedb.io-v1-admin
rules:
  - apiGroups:
      - postgresql.k8s.enterprisedb.io
    resources:
      - clusters
    verbs:
      - "*"
```

特定の名前空間に Subscription オブジェクトをインストールすると、オペレーターのクラスター全体のインストールの openshift-operators など、OLM は、オペレーターが使用する postgresql-operator-manager サービスアカウントに権限を付与するために使用されるクラスターロールも作成します。このクラスターロールの名前は、インストールされたオペレーターのバージョンとインストール時間によって異なるため、異なります。

次のコマンドを実行して、取得できます。

```
oc get clusterrole --selector=olm.owner.kind=ClusterServiceVersion
```

次に、上記のクエリーで返された名前は cloud-native-postgresql.VERSION-HASH のフォームを持つ必要がありますを使用して、 describe コマンドを介してルール、リソース、および動詞を確認できます。

```
oc describe clusterrole cloud-native-postgresql.VERSION-HASH
```

```
Name:         cloud-native-postgresql.VERSION.HASH
Labels:       olm.owner=cloud-native-postgresql.VERSION
              olm.owner.kind=ClusterServiceVersion
              olm.owner.namespace=openshift-operators
              operators.coreos.com/cloud-native-postgresql.openshift-operators=
Annotations:  <none>
PolicyRule:
  Resources                                                  Non-Resource URLs ␣
↪Resource Names  Verbs
  ---------                                                  -----------------  ---
↪-----------  -----
  configmaps                                                 []                 []
              [create delete get list patch update watch]
  secrets                                                    []                 []
              [create delete get list patch update watch]
  services                                                   []                 []
              [create delete get list patch update watch]
  deployments.apps                                           []                 []
              [create delete get list patch update watch]
  poddisruptionbudgets.policy                                []                 []
              [create delete get list patch update watch]
  backups.postgresql.k8s.enterprisedb.io                     []                 []
              [create delete get list patch update watch]
  clusters.postgresql.k8s.enterprisedb.io                    []                 []
              [create delete get list patch update watch]
  poolers.postgresql.k8s.enterprisedb.io                     []                 []
              [create delete get list patch update watch]
  scheduledbackups.postgresql.k8s.enterprisedb.io            []                 []
              [create delete get list patch update watch]
  persistentvolumeclaims                                     []                 []
              [create delete get list patch watch]
  pods/exec                                                  []                 []
              [create delete get list patch watch]
  pods                                                       []                 []
              [create delete get list patch watch]
  jobs.batch                                                 []                 []
              [create delete get list patch watch]
  podmonitors.monitoring.coreos.com                          []                 []
              [create delete get list patch watch]
  serviceaccounts                                            []                 []
              [create get list patch update watch]
  rolebindings.rbac.authorization.k8s.io                     []                 []
              [create get list patch update watch]
```

(次のページに続く)

(前のページからの続き)

```
roles.rbac.authorization.k8s.io                                 []                  []
           [create get list patch update watch]
leases.coordination.k8s.io                                      []                  []
           [create get update]
events                                                          []                  []
           [create patch]
mutatingwebhookconfigurations.admissionregistration.k8s.io      []                  []
           [get list update]
validatingwebhookconfigurations.admissionregistration.k8s.io    []                  []
           [get list update]
customresourcedefinitions.apiextensions.k8s.io                  []                  []
           [get list update]
namespaces                                                      []                  []
           [get list watch]
nodes                                                           []                  []
           [get list watch]
clusters.postgresql.k8s.enterprisedb.io/status                  []                  []
           [get patch update watch]
poolers.postgresql.k8s.enterprisedb.io/status                   []                  []
           [get patch update watch]
configmaps/status                                               []                  []
           [get patch update]
secrets/status                                                  []                  []
           [get patch update]
backups.postgresql.k8s.enterprisedb.io/status                   []                  []
           [get patch update]
scheduledbackups.postgresql.k8s.enterprisedb.io/status          []                  []
           [get patch update]
pods/status                                                     []                  []
           [get]
clusters.postgresql.k8s.enterprisedb.io/finalizers              []                  []
           [update]
poolers.postgresql.k8s.enterprisedb.io/finalizers               []                  []
           [update]
```

オペレーターがこれらの昇格された権限を必要とする理由の詳細については、*Security / Cluster / RBAC* を参照してください。

ユーザーと権限

CloudNativePG クラスターオペレーターの EDB Postgres AI を使用する非常に一般的な方法は、`cluster-admin` ロールに依存し、リソースを一元管理することです。

または、他のオペレーターまたはリソースと同様に、Kubernetes / OpenShift で利用可能な RBAC フレームワークを使用できます。

たとえば、他のクラウドネイティブアプリケーションと同様に、clusters.postgresql.k8s.enterprisedb.io-v1-admin クラスターロールを特定の名前空間の特定のグループまたはユーザーにバインドすることに関心がある場合があります。次の例は、そのクラスターロールを web-prod プロジェクト内の特定のユーザーにバインドします。

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: web-prod-admin
  namespace: web-prod
subjects:
  - kind: User
    apiGroup: rbac.authorization.k8s.io
    name: mario@cioni.org
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: clusters.postgresql.k8s.enterprisedb.io-v1-admin
```

他の事前定義された ClusterRole でも同じプロセスを繰り返すことができます。

一方、クラスターロールを使用したくない場合は、この例のように特定の名前空間ロールを作成できます。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: web-prod-view
  namespace: web-prod
rules:
  - apiGroups:
      - postgresql.k8s.enterprisedb.io
    resources:
      - clusters
    verbs:
      - get
      - list
      - watch
```

次に、このロールを特定のユーザーに割り当てます。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: web-prod-view
  namespace: web-prod
roleRef:
  apiGroup: rbac.authorization.k8s.io
```

(次のページに続く)

(前のページからの続き)

```
  kind: Role
  name: web-prod-view
subjects:
  - apiGroup: rbac.authorization.k8s.io
    kind: User
    name: web-prod-developer1
```

この最後の例では、その名前空間 web-prod のオペレーターによって管理されているすべてのリソースへの管理権限 verbs が* と等しいロールを作成します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: web-prod-admin
  namespace: web-prod
rules:
  - apiGroups:
      - postgresql.k8s.enterprisedb.io
    resources:
      - clusters
      - backups
      - scheduledbackups
      - poolers
    verbs:
      - "*"
```

ポッドセキュリティ標準

OpenShift の CloudNativePG クラスターの EDB Postgres AI は、restricted-v2 SCC SecurityContextConstraints と動作します。

オペレーターは、最初からセキュリティを重視して開発されたため、常に Red Hat 認定プロセスを遵守することに加えて、CloudNativePG クラスターの EDB Postgres AI は、OpenShift 4.11 で導入された新しい SCC の下で動作します。

デフォルトでは、CloudNativePG クラスターの EDB Postgres AI はすべての機能をドロップします。これにより、ライフサイクル中にオペレーターが安全でない機能を利用しないことが保証されます。

OpenShift では、OpenShift 展開から各ポッドの SecurityContext.SeccompProfile を継承します。これは、Pod Security Admission Controller によって設定されます。

> **注釈**
>
> *nonroot-v2* および `hostnetwork-v2` が制限の少ない SCC として認定されている場合でも、当社はそれらに対してテストを実行していないため、これらの SCC が動作することを保証できません。そうは言っ

ても、 *nonroot-v2* および `hostnetwork-v2` は `restricted-v2` のルールのサブセットであるため、これらが機能しないと信じる理由はありません。

デフォルトでは、 `Pooler` リソースは、`docker.enterprisedb.com/k8s/pgbouncer` イメージで実行される `pgbouncer` コンテナを持つポッドを作成します。

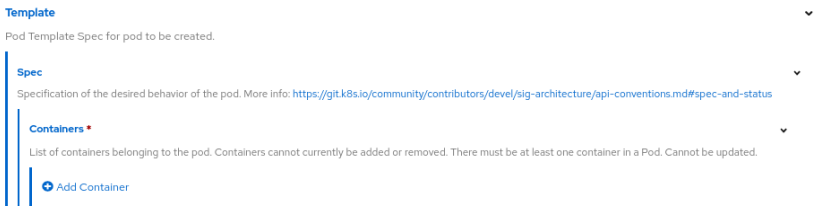


図 10: Pooler template

次に

- コンテナの名前として `pgbouncer` を設定しますポッドテンプレートの必須フィールド
- 必要に応じて_“Image”_フィールドを設定します以下の画像を参照してください

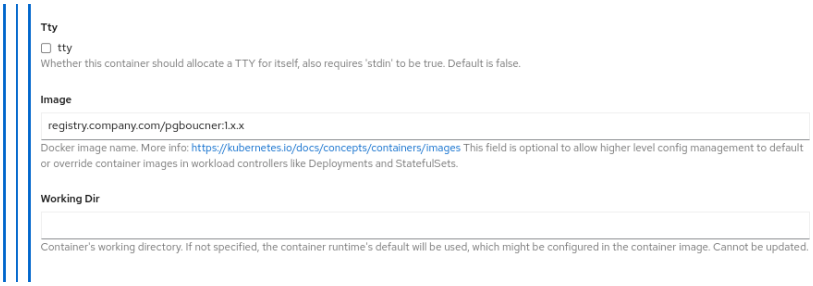


図 11: Pooler image

Velero の OADP

CloudNativePG クラスターオペレーターの EDB Postgres AI は、Velero を管理するために Openshift API for Data Protection オペレーターの使用を推奨します

OpenShift 環境。 CloudNativePG クラスターの EDB Postgres AI が Velero と統合する方法に関する具体的な詳細は、アドオンドキュメントの *Velero section* で見つけることができます。

OADP operator

は、EDB で直接サポートされていないコミュニティオペレーターです。OADP オペレーターは、 EDB Postgres で Velero を使用する必要はありませんが、OpenShift に Velero をインストールする便利な方法です。

モニタリングとメトリック

OpenShift には、CloudNativePG クラスターの EDB Postgres AI を含むユーザー定義プロジェクトに活用できる Prometheus インストールが含まれています。

Grafana は OpenShift に含まれなくなったため、Grafana 統合はこのガイドの範囲外です。

このセクションでは、デフォルトの OpenShift インストールを活用して、基本的なオブザーバビリティを開始する方法を示します。

OpenShift monitoring stack overview を参照してください。

さらなる背景については。

OpenShift 構成によっては、CloudNativePG Cluster クラスターの EDB Postgres AI をモニタする前に、少しのセットアップを行う必要がある場合があります。

OpenShift を enable monitoring for user-defined projects に構成する必要があります。

おそらく OpenShift 管理者に、インストールに `cluster-monitoring-config` configMap があるかどうか、存在する場合はユーザーワークロードモニタリングが有効になっているかどうかを確認する必要があります。

この `configMap` の存在を確認できます。`openshift-monitoring` 名前空間にある必要があることに注意してください。

```
oc -n openshift-monitoring get configmap cluster-monitoring-config
```

ユーザーワークロードモニタリングを有効にするには、構成マップを `oc apply` または `oc edit` 次のようにすることができます。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    enableUserWorkload: true
```

`enableUserWorkload` を設定すると、複数の監視コンポーネントが `openshift-user-workload-monitoring` 名前空間に作成されます。

```
$ oc -n openshift-user-workload-monitoring get pod
NAME                                   READY   STATUS    RESTARTS   AGE
prometheus-operator-58768d7cc-28xb5    2/2     Running   0          5h10m
prometheus-user-workload-0             6/6     Running   0          5h10m
prometheus-user-workload-1             6/6     Running   0          5h10m
thanos-ruler-user-workload-0           3/3     Running   0          5h10m
thanos-ruler-user-workload-1           3/3     Running   0          5h10m
```

これで、メトリックを有効にしているクラスターからメトリックを表示できるようになります。

たとえば、 `foo` 名前空間での監視を使用して次のクラスターを作成できます。

```
kubectl apply -n foo -f - <<EOF
- --
```

(次のページに続く)

(前のページからの続き)

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-with-metrics
spec:
  instances: 3

  storage:
    size: 1Gi

  monitoring:
    enablePodMonitor: true
EOF
```

これで、この例でインストールされるデフォルトのメトリックを照会できるようになります。OpenShift の Observe セクション開発者パースペクティブに、PromQL クエリーを作成できる Metrics サブメニューが表示されます。オートコンプリートが有効になっているため、cnp_ 接頭辞をピークにできます。

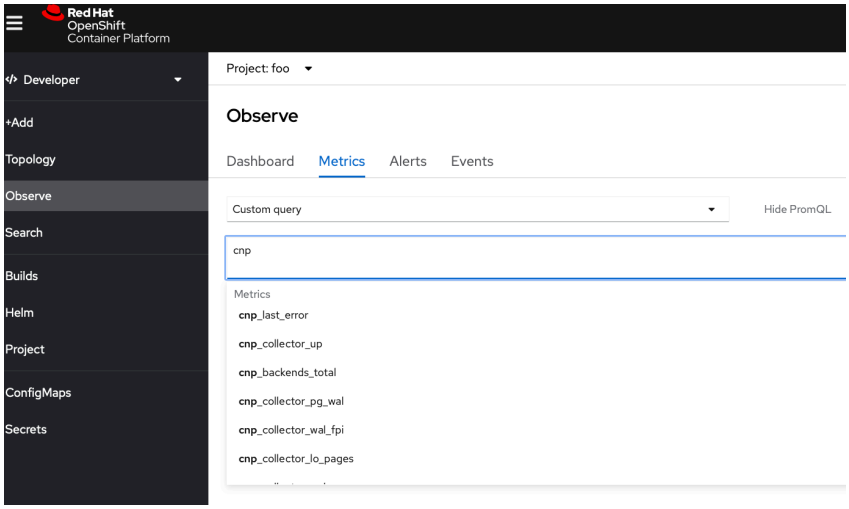


図 12: Prometheus queries

PrometheusRules としてのデフォルトメトリックに基づいてアラートを定義するのは簡単です。

prometheusrule.yaml にルールの例がいくつかあります。

ファイル、ダウンロードできます。

ルールを適用する前に、やはり OpenShift のセットアップが必要になる場合があります。

ルールを適用および監視するユーザーに monitoring-rules-edit または少なくとも monitoring-rules-view ロールを割り当てる必要があります。

これには、名前空間にその権限で RoleBinding を作成することが含まれます。再度、

relevant OpenShift document page を参照してください

詳細については、特に、 _Granting userpermissions by using the web console_セクションが重要です。

`monitoring-rules-edit` の RoleBinding は名前空間に適用されるため、適切なものを取得していることを確認してください。

以前にダウンロードした `cnp-prometheusrule.yaml` ファイルにアラートが含まれているとします。これらのルールは次のようにインストールできます。

```
oc apply -n foo -f cnp-prometheusrule.yaml
```

これで、アラートがある場合は表示されるはずです。最初はアラートが存在しない場合があることに注意してください。

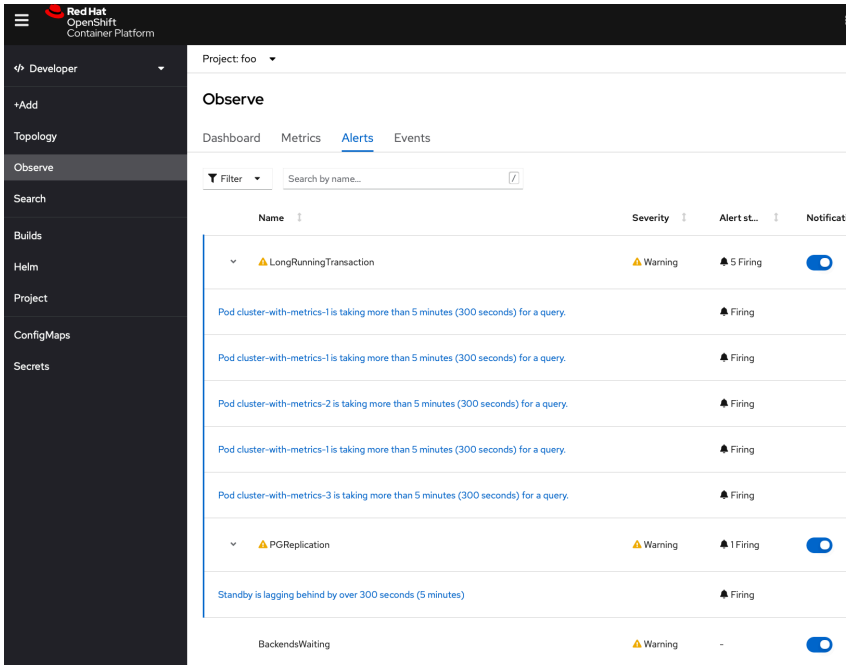


図 13: Prometheus alerts

アラートルーティングと通知は、このガイドの範囲を超えています。

6.5 Iron Bank

CloudNativePG Cluster{{name.abbr}}の EDB Postgres AI は、Iron Bank で入手できます。 overview page で読めるように

> Iron Bank は、国防総省の強化コンテナのソースです。 > > […中略…] >> Iron Bank は最終的には誰でも消費または貢献できます。ただし、特に次のペルソナをターゲットとしています。> > - 強化されたコンテナと各コンテナのアイアンバンク BoE 証拠本文を使用したい国防総省組織> - コンテナへのコントリビューションを支援したい国防総省組織例バグ修正、新しいアプリケーション、更新) > - アプリケーションに関連するリスクを理解したい国防総省承認職員 > - アプリケーションを国防総省に持ち込みたい商用ベンダー

Iron Bank は、国防総省の Platform One の一部です。

EDB Postgres AI for CloudNativePG Cluster の Iron Bank ページにアクセスするには、Iron Bank の資格情報が必要です。

6.5.1 Iron Bank から EDB {{name.short}}とオペランドイメージを取得する

画像は別の Iron Bank container registry から取得されます。 Iron Bank レジストリからイメージをプルできるようにするには、 instructions from Iron Bank に従ってください。

具体的には、 registry1 を使用する必要があります

イメージをプルするための資格情報。

目的の演算子またはオペランドイメージを見つけるには、画像に示すように、検索ツールを使用して文字列 `enterprisedb` で検索し、`Tags` でフィルターして `stable` を探すことをお勧めします。そこから、イメージをプルする指示を取得できます。

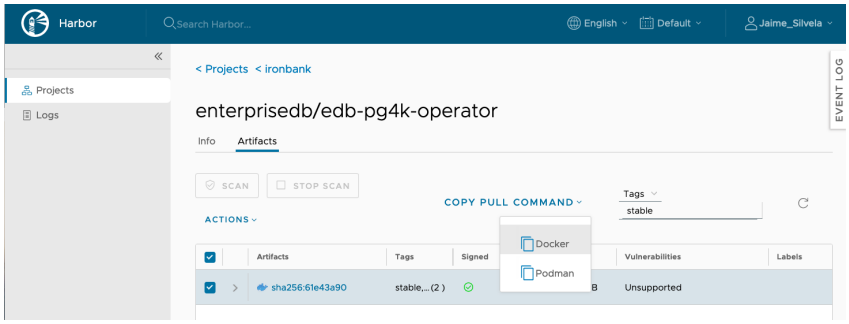


図 14: pulling-ironbank-images

たとえば、Ironbank から EPAS 18 オペランドをプルするには、次のコマンドを実行できます。

```
docker pull registry1.dso.mil/ironbank/enterprisedb/edb-postgres-advanced-18:18
```

同様に、EPAS 16 または 17 の場合

```
docker pull registry1.dso.mil/ironbank/enterprisedb/edb-postgres-advanced-17:17
docker pull registry1.dso.mil/ironbank/enterprisedb/edb-postgres-advanced-16:16
```

より具体的なタグを選択するか、特定の SHA を使用したい場合は、Harbor ページから見つける必要がありますたとえば EPAS 18 、 EPAS 17 、 EPAS 16 。

6.5.2 Iron Bank イメージを使用した{{name.short}}オペレーターのインストール

インストールするには、Iron Bank イメージをポイントする展開マニフェストが必要です。

installation instructions for EDB {{name.short}} から展開マニフェストを取得できます。たとえば、1.22.0 リリースの場合、マニフェストは `https://get.enterprisedb.io/cnp/postgresql-operator-1.22.0.yaml` で入手できます。IronBank イメージのイメージパスを設定する必要がある場所がいくつかあります。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app.kubernetes.io/name: cloud-native-postgresql
  name: postgresql-operator-controller-manager
```

(次のページに続く)

(前のページからの続き)

```
  namespace: postgresql-operator-system
spec:
  [··· snipped ···]
  template:
    metadata:
      labels:
        app.kubernetes.io/name: cloud-native-postgresql
    spec:
      containers:
      - args:
        - controller
          [··· snipped ···]
        env:
        - name: PULL_SECRET_NAME
          value: postgresql-operator-pull-secret
        - name: OPERATOR_IMAGE_NAME
          value: <INSERT-YOUR-OPERATOR-IMAGE>

        [··· snipped ···]
        image: <INSERT-YOUR-OPERATOR-IMAGE>
```

オペレーターを Iron Bank から直接展開する場合は、上記のように、レジストリへの資格情報を使用してプルシークレットを作成および設定する必要があります。

サイトの手順を使用して Iron Bank からイメージを取得し、そこから再タグ付けしてローカルレジストリに公開するか、Kubernetes ノードに直接プッシュする方が簡単な場合があります。

これを配置すると、

installation instructions で説明しているように、`kubectl apply -f` を使用してマニフェストを適用できます。

6.5.3 IronBank イメージを使用した EPAS オペランドを使用したクラスターの展開

EPAS operand を使用してクラスターを展開するには、`Cluster` リソース YAML で Ironbank オペランドイメージを適切に参照する必要があります。例、EPAS 18 オペランドを使用して{{name.short}}クラスターを展開するには

1. 次の内容を使用して `Cluster` リソース YAML ファイルを作成または編集します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example-full
spec:
  imageName: registry1.dso.mil/ironbank/enterprisedb/edb-postgres-advanced-18:18
  imagePullSecrets:
    - name: my_ironbank_secret
```

EPAS 17 または 16 の場合、対応する画像を使用します。

- EPAS 17 `registry1.dso.mil/ironbank/enterprisedb/edb-postgres-advanced-17:17`
- EPAS 16 `registry1.dso.mil/ironbank/enterprisedb/edb-postgres-advanced-16:16`

2. YAML を適用します。

```
kubectl apply -f <filename>
```

3. リソースのステータスを確認します。

```
kubectl get clusters
```

6.6 Transparent Data Encryption (TDE)

Transparent Data EncryptionTDE は、 **保存データ**、つまりディスク上のデータベースファイルを暗号化するために複数のデータベースベンダーが使用しているテクノロジーです。ただし、TDE は使用中のデータを暗号化しません。

TDE は、バージョン 15 から EDB Postgres Advanced Server および EDB Postgres Extended Server に含まれ、CloudNativePG Cluster オペレーターの EDB Postgres AI でサポートされています。

TDE feature in the EPAS documentation を理解してください。

TDE をアクティブにすると、WAL ファイルとテーブルのファイルの両方が暗号化されます。データの暗号化/復号化は、アプリケーションの変更やクライアントドライバーの更新を必要とせずにデータベースによって管理されるため、ユーザーに対して完全に透過的です。

> **注釈**
>
> 以下に示すコードサンプルでは、 YAML マニフェストの `postgresql` の `epas` サブセクションを使用して TDE をアクティブにします。 *epas* セクションを使用して、PG 拡張イメージおよび EPAS イメージの TDE を有効にできます。

CloudNativePG クラスターの EDB Postgres AI は、TDE を使用する 3 つの方法を提供します。

- パスフレーズを含むシークレットを使用する
- カスタムパスフレーズコマンドを含むシークレットを使用する
- カスタム wrap/unwrap コマンドを含むシークレットのペアを使用する

6.6.1 パスフレーズシークレット

基本的なアプローチは、パスフレーズを Kubernetes シークレットに保存することです。このようなパスフレーズは、EPAS バイナリキーの暗号化に使用されます。

the EPAS documentation を参照してください。

EPAS 暗号化キーの詳細については、

オペレーターで TDE をアクティブにするのは簡単です。マニフェストの epas セクションで、 tde スタンザを使用して TDE を有効にし、TDE 暗号化キーを保持する Kubernetes シークレットを設定します。

例

```
[···]
postgresql:
  epas:
    tde:
      enabled: true
      secretKeyRef:
        name: tde-key
        key: key
```

cluster-example-tde.yaml に例があります。

> **注釈**
>
> このファイルには、暗号化キーを保持するためのシークレットの定義も含まれています。この目的でシークレットを作成する方法に関する例については、次のセクションを参照してください。

シークレットに保存されているキーは、openssl を呼び出して EPAS 暗号化キーをラップ/ラップ解除するためのパスフレーズとして使用されます。

パスフレーズを含むシークレットを作成する方法

最初にパスフレーズを選択します。ランダムに生成されたパスフレーズを使用することをお勧めしますが、この例では、パスフレーズとして PostgresRocks を使用し、kubectl に依存してシークレット定義を生成します。

```
kubectl create secret generic -o yaml tde-key \
    --from-literal=key=PostgresRocks
```

これにより、次のようなものが返されます。

```
apiVersion: v1
data:
  key: UG9zdGdyZXNSb2Nrcw==
kind: Secret
metadata:
  creationTimestamp: "YYYY-MM-DDTHH:MM:SSZ"
  name: tde-key
  namespace: default
  resourceVersion: ....
  uid: ....
type: Opaque
```

クラスター内にシークレットを実際に作成するには、 kubectl apply を実行するか、上記の create コマンドの-o yaml オプションを削除してください。

6.6.2 カスタムパスフレーズコマンド

上記のクラスターマニフェストスニペットの secretKeyRef の代わりに、シークレットに保存されている passphraseCommand を指定できます。 passphrase コマンドを実行して、openssl で使用するパスフレーズを生成できます。

```
[···]
postgresql:
  epas:
    tde:
      enabled: true
      passphraseCommand:
        name: tde-passphrase
        key: command
```

passphrase コマンドは標準出力に書き込む必要があります。たとえば、単に echo my-passphrase を使用できます。

コマンドによって生成されたパスフレーズは、secretKeyRef が使用されたのと同じ方法、つまり openssl のパスフレーズ引数として使用されます。

6.6.3 カスタム wrap/unwrap コマンド

openssl のデフォルトの呼び出しに依存するのではなく、wrap および unwrap コマンドを指定することもできます。これは、カスタムコマンドを含むシークレットを作成し、tde スタンザでこれらのシークレットを宣言することにより実行できます。

次のスニペットは、カスタムコマンドを使用して TDE が有効になっているクラスターを示しています。

```
[···]
postgresql:
  epas:
    tde:
      enabled: true
      wrapCommand:
        name: tde-wrap-command
        key: command
      unwrapCommand:
        name: tde-unwrap-command
        key: command
```

カスタムコマンドは、次の規則に従う必要があります。

1. カスタム wrap コマンドは、EPAS がバイナリキーをフィードするために使用する標準入力からの入力を受け入れる必要があります。明示的な引数シェルのリダイレクトではなくを介してファイルに

書き込む必要があります。さらに、file 引数には文字列「%p」を指定する必要があります。これは、EPAS が新しいラップされた暗号化キーファイルのファイルパスを渡すために使用するプレースホルダーです。

2. custom unwrap コマンドは、標準出力に書き込む必要があります。入力には明示的なファイルパス引数が必要ですシェルリダイレクトではありません。繰り返しますが、file 引数には文字列「%p」を指定する必要があります。これは、EPAS がラップされた暗号化キーファイルパスで埋められるプレースホルダーです。

例

- Wrap コマンド `openssl enc -aes-128-cbc -pass pass:temp-pass -e -out %p`
- unwrap コマンド `openssl enc -aes-128-cbc -pass pass:temp-pass -d -in %p`

6.6.4 HashiCorp Vault を使用した例

次の例は、HashiCorp Vault を使用して暗号化キーを保存し、それを使用して TDE をアクティブにする方法を示しています。`vault` CLI は、Vault と対話するために使用され、デフォルトで EDB Postgres Advanced Server イメージに含まれています。

まず、ボールトを実行している場所はどこでも、Transit Secrets エンジンを有効にし、キーを作成する必要があります。

```
vault secrets enable transit
vault write -f transit/keys/pg-tde
```

次に、カスタム wrap/unwrap コマンドを含むシークレットを作成します。Wrap および unwrap コマンドは、EPAS イメージにあるバイナリを「ラップ」します。バイナリは vault API と対話して、EPAS 暗号化を暗号化/復号化します。

バイナリには、5 つのフラグが必要です `--file`、`--host`、`--secret`、`--key`、`--vault-endpoint` `--host` フラグは `http://vault-host:vault-port` のフォーマットであり、Vault に到達するには提供する必要があります。server`--secret` フラグは、ボールトトークンを含む Kubernetes シークレットの名前であり、`--key` フラグは、ボールトトークンをポイントするそのシークレットのキーです。`--vault-endpoint` フラグは、ボールト内で作成されたキーの名前です。上記の例では、`pg-tde` です。

Kubernetes で Vault オペレーターを実行している場合、次の 2 つのコマンドからルートトークンを取得できます。

```
kubectl exec vault-0 -- vault operator init -key-shares=1 -key-threshold=1 -
↪format=json > cluster-keys.json
cat cluster-keys.json | jq -r ".root_token"
```

```
kubectl create secret generic -o yaml vault-token \
    --from-literal=wrap="/bin/vault wrap --file %p --host http://vault:8200 --secret␣
↪vault-token --key token --vault-endpoint pg-tde" \
    --from-literal=unwrap="/bin/vault unwrap --file %p --host http://vault:8200 --
↪secret vault-token --key token --vault-endpoint pg-tde" \
    --from-literal=token="hvs.whatever"
```

これで、シークレットを参照するクラスターを作成できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: hashicorp-vault-tde
spec:
  instances: 3
  storage:
    size: 1Gi
  postgresql:
    epas:
      tde:
        enabled: true
        wrapCommand:
          name: vault-token
          key: wrap
        unwrapCommand:
          name: vault-token
          key: unwrap
```

TLS を有効にする

バイナリには、6 つのフラグが必要です --file 、--host 、--secret 、--key 、--vault-endpoint 、--enable-tls --host フラグは http://vault-host:vault-port のフォーマットであり、Vault に到達するには提供する必要があります。server--secret フラグは、ボールトトークンを含む Kubernetes シークレットの名前であり、--key フラグは、ボールトトークンをポイントするそのシークレットのキーです。--vault-endpoint フラグは、ボールト内で作成されたキーの名前です。上記の例では、pg-tde です。 TLS を有効にするには、--enable-tls を true に設定する必要があります。

```
kubectl create secret generic -o yaml vault-token \
    --from-literal=wrap="/bin/vault wrap --file %p --host http://vault:8200 --secret␣
↪vault-token --key token --vault-endpoint pg-tde --enable-tls true" \
    --from-literal=unwrap="/bin/vault unwrap --file %p --host http://vault:8200 --
↪secret vault-token --key token --vault-endpoint pg-tde --enable-tls true" \
    --from-literal=token="hvs.whatever"
```

次に、シークレットまたは configMap で環境変数を指定します。

```
kubectl create secret generic -o yaml env-var-secret \
- -from-literal=VAULT_CACERT="/projected/certificate/vault-ca.pem" \
```

クラスター仕様の envFrom セクションのシークレットを参照して、バイナリがポッドに注入されるときに環境変数を使用できるようにします。1 つ以上の環境変数がファイルを参照している場合、projectedVolumeTemplate に依存してカスタムファイルをマウントし、コンテンツにシークレットまたは構成マップを依存します。この例では、tls-vault-secret キー ca の値がファイルとしてパス/projected/certificate/vault-ca.pem にマウントされます。

```
kubectl create secret generic -o yaml tls-vault-secret --from-file=ca=vault-ca.crt
```

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: hashicorp-vault-tde
spec:
  envFrom:
    - secretRef:
        name: env-var-secret
  projectedVolumeTemplate:
    sources:
      - secret:
          name: tls-vault-secret
          items:
            - key: ca
              path: certificate/vault-ca.pem
  instances: 3
  storage:
    size: 1Gi
  postgresql:
    epas:
      tde:
        enabled: true
        wrapCommand:
          name: vault-token
          key: wrap
        unwrapCommand:
          name: vault-token
          key: unwrap
```

cluster-example-tde.yaml

```
- --
apiVersion: v1
kind: Secret
metadata:
  name: tde-key
data:
  key:␣
↪bG9zcG9sbGl0b3NkaWNlbnBpb3Bpb3Bpb2N1YW5kb3RpZW5lbmhhbWJyZWN1YW5kb3RpZW5lbmZyaW8=

- --
apiVersion: postgresql.k8s.enterprisedb.io/v1
```

(次のページに続く)

(前のページからの続き)

```
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3
  imageName: docker.enterprisedb.com/k8s/edb-postgres-advanced:18-standard-ubi9
  postgresql:
    epas:
      tde:
        enabled: true
        secretKeyRef:
          name: tde-key
          key: key

  storage:
    size: 1Gi
```

6.7 Add-ons

CloudNativePG クラスターの EDB Postgres AI は、クラスターごとに有効にできるアドオンをサポートしています。これらのアドオンは次のとおりです。

1. 外部バックアップアダプター
2. カステン
3. ヴェレロ

> **注釈**
>
> OpenShift で Velero を使用する予定がある場合は、Openshift ドキュメントの *OADP section* を参照してください。

オペレーターはすべてのアドオンを自動的に利用できるようになり、使用するには、`k8s.enterprisedb.io/addons` アノテーションを介してクラスターレベルで有効にする必要があります。

6.7.1 外部バックアップアダプター

外部バックアップアダプタアドオンは、ラベルおよび/または注釈を介して識別するカスタマイズ可能な方法を介して、バックアップ用のサードパーティツールに CloudNativePG クラスターの EDB Postgres AI を統合する汎用方法を提供します。

- どの PVC グループをバックアップするか
- クラスターに 1 つ以上のアクティブなレプリカがある場合、どの PVC を除外するか

- バックアップのために選択された PostgreSQL インスタンスを実行しているポッドスタンバイまたは使用できない場合はプライマリ

バックアップシステムのアダプターを構成できる方法のみが異なる 2 つのアドオンから選択できます。

- `external-backup-adapter` 構成マップまたはシークレットを介してオペレーターの構成レベルで動作をカスタマイズ、オペレーターの展開によって管理されるすべての Postgres クラスターと共有する場合 the `external-backup-adapter section below <Backup>` を参照してください。
- `external-backup-adapter-cluster` 特定のアノテーションを介して Postgres クラスターレベルでアダプターの動作をカスタマイズする場合 the `external-backup-adapter-cluster section below <Backup>` を参照してください

このようなアドオンを使用すると、オペレーターが選択したポッドでバックアップを取得する前または後に実行するコマンドを含むアノテーションの名前を定義できます。

その結果、Kubernetes のサードパーティバックアップツールは、上記に依存して、PostgreSQL クラスターまたはそのセットと自分自身を調整できます。

リカバリーは、オペレーターが既存の PVC グループからクラスターを調整することに依存するだけです。

アダプターのカスタマイズ

前述のように、アダプターは 2 つの方法で構成でき、これにより、 `Cluster` リソースで使用する必要がある実際の `add-on` が決まります。

オペレーターによって管理されるすべての Postgres `Cluster` リソースに同じ動作を定義する予定がある場合は、`external-backup-adapter` アドオンを使用し、オペレーターの構成でアノテーション/ラベルを構成することをお勧めします。

所有する Postgres `Cluster` リソースのサブセットに異なる動作を予定している場合は、`external-backup-adapter-cluster` アドオンを使用することをお勧めします。

両方のアドオンは、カスタマイズという点で同じ機能を共有します。これは、次のキーを持つ YAML オブジェクトとして定義する必要があります。

- `electedResourcesDecorators`
- `excludedResourcesDecorators`
- `excludedResourcesSelector`
- `backupInstanceDecorators`
- `preBackupHookConfiguration`
- `postBackupHookConfiguration`

以下、各セクションについて説明します。さらに下には、2 つのアドオンをそれぞれカスタマイズする方法と、いくつかの例があります。

electedResourcesDecorators セクション

このセクションでは、選択されたすべての PVC グループに配置されるラベルおよび/またはアノテーションの配列を構成できます。

配列の各要素には、次のフィールドが必要です。

key

ラベルまたはアノテーションのキーの名前

metadataType

メタデータのタイプ "label" または"annotation"

value

ラベルまたは注釈に割り当てられる値

excludedResourcesDecorators セクション

このセクションでは、除外されたすべてのポッドおよび PVC に配置されるラベルおよび/またはアノテーションの配列を構成できます。

配列の各要素には、上記の electedResourcesDecorators セクションと同じフィールドが必要です。

excludedResourcesSelector セクション

このセクションでは、excludedResourcesDecorators に適用されるポッドと PVC を選択します。 label selector rule を受け入れます

値として。空の場合、すべてのポッドと選択されなかったすべての PVC が除外されます。

backupInstanceDecorators セクション

このセクションでは、オペレーターによってバックアップのために選択され、実行されるフックが含まれるインスタンスに配置されるラベルおよび/またはアノテーションの配列を構成できます。

配列の各要素には、上記の electedResourcesDecorators セクションと同じフィールドが必要です。

preBackupHookConfiguration セクション

このセクションでは、オペレーターがコンテナの名前、バックアップを取る前に実行するコマンド、およびサードパーティツール側でエラー/中止が発生した場合に実行するコマンドを配置するアノテーションの名前を制御できます。。このようなメタデータは、バックアップのためにオペレーターが選択したインスタンスに適用されます上記の backupInstanceDecorators を参照してください。

次のフィールドを提供する必要があります。

container

pre-backup コマンドを実行するコンテナに関する情報を配置する場所を指定します。コンテナ名は固定値であり、構成はできません。注釈に保存されます。フックでポッドを装飾するには、次を参照してください instanceWithHookDecorators

command

バックアップが取られる前に実行されるコマンドに関する情報を配置する場所を指定します。実行されるコマンドは固定値であり、設定はできません。注釈に保存されます。フックでポッドを装飾するには、次を参照してください instanceWithHookDecorators

onError

エラーが発生した場合に実行されるコマンドに関する情報を配置する場所を指定します。実行されるコマンドは固定値であり、設定はできません。注釈に保存されます。フックでポッドを装飾するには、次を参照してください instanceWithHookDecorators

postBackupHookConfiguration セクション

このセクションでは、オペレーターがコンテナの名前とバックアップを取った後に実行するコマンドを配置するアノテーションの名前を制御できます。このようなメタデータは、バックアップのためにオペレーターが選択したインスタンスに適用されます上記の backupInstanceDecorators を参照してください。

次のフィールドを提供する必要があります。

container

post-backup コマンドを実行するコンテナに関する情報を配置する場所を指定します。コンテナ名は固定値であり、構成はできません。注釈に保存されます。フックでポッドを装飾するには、次を参照してください instanceWithHookDecorators

command

バックアップが取られた後に実行されるコマンドに関する情報を配置する場所を指定します。実行されるコマンドは固定値であり、設定はできません。注釈に保存されます。フックでポッドを装飾するには、次を参照してください instanceWithHookDecorators

external-backup-adapter アドオン

external-backup-adapter アドオンは、オペレーターの ConfigMap /Secret の EXTERNAL_BACKUP_ADDON_CONFIGURATION フィールドを介して、オペレーターレベルで完全に設定できます。

詳細については、このセクションの最後に提供されているサンプルファイル、または以下の例を参照してください。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: postgresql-operator-controller-manager-config
  namespace: postgresql-operator-system
data:
  # ...
  EXTERNAL_BACKUP_ADDON_CONFIGURATION: |-
    electedResourcesDecorators:
      - key: "app.example.com/elected"
```

(次のページに続く)

(前のページからの続き)

```
      metadataType: "label"
      value: "true"
  excludedResourcesSelector: app=xyz,env=prod
  excludedResourcesDecorators:
    - key: "app.example.com/excluded"
      metadataType: "label"
      value: "true"
    - key: "app.example.com/excluded-reason"
      metadataType: "annotation"
      value: "Not necessary for backup"
  backupInstanceDecorators:
    - key: "app.example.com/hasHooks"
      metadataType: "label"
      value: "true"
  preBackupHookConfiguration:
    container:
      key: "app.example.com/pre-backup-container"
    command:
      key: "app.example.com/pre-backup-command"
    onError:
      key: "app.example.com/pre-backup-on-error"
  postBackupHookConfiguration:
    container:
      key: "app.example.com/post-backup-container"
    command:
      key: "app.example.com/post-backup-command"
```

アドオンは、次のアノテーションを Cluster リソースに追加することによりアクティブにできます。

```
k8s.enterprisedb.io/addons: ["external-backup-adapter"]
```

external-backup-adapter-cluster アドオン

external-backup-adapter-cluster アドオンは、次の例に示すように、k8s.enterprisedb.io/externalBackupAdapterClusterConfig アノテーション-コンテンツとして YAML オブジェクトを受け入れます-を介して使用する各 Cluster リソースで構成する必要があります。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
  annotations:
    "k8s.enterprisedb.io/addons": ["external-backup-adapter-cluster"]
    "k8s.enterprisedb.io/externalBackupAdapterClusterConfig": |-
      electedResourcesDecorators:
```

(次のページに続く)

(前のページからの続き)

```
        - key: "app.example.com/elected"
          metadataType: "label"
          value: "true"
      excludedResourcesSelector: app=xyz,env=prod
      excludedResourcesDecorators:
        - key: "app.example.com/excluded"
          metadataType: "label"
          value: "true"
        - key: "app.example.com/excluded-reason"
          metadataType: "annotation"
          value: "Not necessary for backup"
      backupInstanceDecorators:
        - key: "app.example.com/hasHooks"
          metadataType: "label"
          value: "true"
      preBackupHookConfiguration:
        container:
          key: "app.example.com/pre-backup-container"
        command:
          key: "app.example.com/pre-backup-command"
        onError:
          key: "app.example.com/pre-backup-on-error"
      postBackupHookConfiguration:
        container:
          key: "app.example.com/post-backup-container"
        command:
          key: "app.example.com/post-backup-command"
spec:
  instances: 3
  storage:
    size: 1Gi
```

フェンシングのアノテーションについて

構成された外部バックアップアダプタがアノテーションをバックアップする場合、フェンシングアノテーションはプレバックアップフックによって設定され、復元されたクラスターに持続します。クラスターを復元した後、これを修正するには、`Cluster` オブジェクトからフェンシングアノテーションを手動で削除する必要があります。

これは、kubectl の cnp プラグインを使用して行うことができます。

```
kubectl cnp fencing off <cluster-name>
```

または、 cnp プラグインがない場合は、次のコマンドを使用してフェンシングアノテーションを手動で削除できます。

```
kubectl annotate cluster <cluster-name> k8s.enterprisedb.io/fencedInstances-
```

詳細については、 *fencing documentation* を参照してください。

制限事項

バックアップ部分に関する限り、現在、Kubernetes の EDB Postgres と external-backup-adapter および external-backup-adapter-cluster との統合は、**コールドバックアップ** のみをサポートしています。これらは**オフラインバックアップ**とも呼ばれます。これは、選択したレプリカが一時的にフェンスされ、external-backup-adapter および external-backup-adapter-cluster が PVC グループ、つまり PGDATA ボリューム、および使用可能な場合は WAL ボリュームの物理的スナップショットを取得できることを意味します。

この短い時間フレームでは、スタンバイは読み取り専用の接続を受け入れることができません。使用可能なスタンバイがなく、通常は単一インスタンスのクラスターにいるため、アノテーション k8s.enterprisedb.io/snapshotAllowColdBackupOnPrimary が true に設定されている場合、external-backup-adapter および external-backup-adapter-cluster はプライマリを一時的にフェンスし、読み取り/書き込み操作の点でダウンタイムが発生します。このユースケースは、通常、開発環境に任されます。

YAML ファイルの完全な例

これは、次のいずれかに配置される YAML コンテンツの完全な例です。

- external-backup-adapter アドオンについて上記で説明したオペレーターの構成プロセスの一部としての EXTERNAL_BACKUP_ADDON_CONFIGURATION オプション、または
- external-backup-adapter-cluster アドオンの k8s.enterprisedb.io/externalBackupAdapterClusterConfig アノテーション内

```yaml
#  An array of labels and/or annotations that will be placed

#  on the elected PVC group

electedResourcesDecorators:
  - key: "backup.example.com/elected"
    metadataType: "label"
    value: "true"

#  An array of labels and/or annotations that will be placed

#  on every excluded pod and PVC

excludedResourcesDecorators:
  - key: "backup.example.com/excluded"
    metadataType: "label"
    value: "true"
  - key: "backup.example.com/excluded-reason"
```

(次のページに続く)

(前のページからの続き)

```
    metadataType: "annotation"
    value: "Not necessary for backup"

#  A LabelSelector containing the labels being used to filter Pods

#  and PVCs to decorate with excludedResourcesDecorators.

#  It accepts a label selector rule as value.

#  See https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/
↪#label-selectors

#  When empty, all the Pods and every PVC that is not elected will be excluded.

excludedResourcesSelector: app=xyz,env=prod

#  An array of labels and/or annotations that will be placed

#  on the instance pod thats been selected for the backup by

#  the operator and which contains the hooks.

#  At least one element is required

backupInstanceDecorators:
  - key: "backup.example.com/hasHooks"
    metadataType: "label"
    value: "true"

#  The pre-backup hook configuration allows you to control the names

#  of the annotations in which the operator will place the container

#  name, the command to run before taking the backup, and the command

#  to run in case of error/abort on the third-party tool side.

#  Such metadata will be applied on the instance thats been selected

#  by the operator for the backup (see `backupInstanceDecorators`)

preBackupHookConfiguration:
  # Where to place the information about the container that will run
  # the pre-backup command. The container name is a fixed value and
  # cannot be configured. Will be saved in the annotations.
```

(次のページに続く)

(前のページからの続き)

```
  # To decorate the pod with the hooks refer to: instanceWithHookDecorators
  container:
    key: "app.example.com/pre-backup-container"
  # Where to place the information about the command that will be
  # executed before the backup is taken. The command is a fixed value
  # and cannot be configured. Will be saved in the annotations.
  # To decorate the pod with the hooks refer to: instanceWithHookDecorators
  command:
    key: "app.example.com/pre-backup-command"
  # Where to place the information about the command that will be
  # executed in case of an error on the third-party tool side.
  # The command is a fixed value and cannot be configured.
  # Will be saved in the annotations.
  # To decorate the pod with the hooks refer to: instanceWithHookDecorators
  onError:
    key: "app.example.com/pre-backup-on-error"

#  The post-backup hook configuration allows you to control the names

#  of the annotations in which the operator will place the container

#  name and the command to run after taking the backup.

#  Such metadata will be applied on the instance thats been selected by

#  the operator for the backup (see `backupInstanceDecorators`).

postBackupHookConfiguration:
  # Where to place the information about the container that will run
  # the post-backup command. The container name is a fixed value and
  # cannot be configured. Will be saved in the annotations.
  # To decorate the pod with hooks refer to: instanceWithHookDecorators
  container:
    key: "app.example.com/post-backup-container"
  # Where to place the information about the command that will be
  # executed after the backup is taken. The command is a fixed value
  # and cannot be configured. Will be saved in the annotations.
  # To decorate the pod with hooks refer to: instanceWithHookDecorators
  command:
    key: "app.example.com/post-backup-command"
```

6.7.2 カステン

Kasten は、Kubernetes の非常に人気のあるデータ保護ツールであり、Kubernetes アプリケーションのバックアップと復元、ディザスターリカバリー、およびアプリケーションモビリティを可能にします。詳細については、Kastenwebsite を参照してください。

つまり、Kubernetes クラスターの EDB Postgres で Kasten との透過的な統合を有効にするには、`Cluster` 仕様の `k8s.enterprisedb.io/addons` アノテーションに `kasten` 値を追加するだけです。例

```yaml
kind: Cluster
metadata:
  name: one-instance
  annotations:
    k8s.enterprisedb.io/addons: ["kasten"]
    k8s.enterprisedb.io/snapshotAllowColdBackupOnPrimary: enabled
spec:
  instances: 1
  storage:
    size: 1Gi
  walStorage:
    size: 1Gi
```

クラスターが作成され正常になると、オペレーターは指定されたバックアップとして最も遠いアヘッドレプリカインスタンスを選択し、アノテーションとラベルを介して Kasten 固有のバックアップフックをそのインスタンスに追加します。

制限事項

バックアップ部分に関する限り、現在 Kasten との Kubernetes 統合の EDB Postgres は **コールドバックアップ** のみをサポートしています。これらは**オフラインバックアップ**とも呼ばれます。これは、選択したレプリカが一時的にフェンスされ、外部バックアップアダプターが PVC グループ、つまり `PGDATA` ボリューム、および使用可能な場合は WAL ボリュームの物理的スナップショットを取得できることを意味します。

この短い時間フレームでは、スタンバイは読み取り専用の接続を受け入れることができません。使用可能なスタンバイがなく、通常は単一インスタンスのクラスターにいるため、アノテーション `k8s.enterprisedb.io/snapshotAllowColdBackupOnPrimary` が true に設定されている場合、Kasten はプライマリを一時的にフェンスし、読み取り/書き込み操作の点でダウンタイムが発生します。このユースケースは、通常、開発環境に任されます。

リカバリーの面では、Kasten との統合はスナップショットリカバリーのみをサポートしています。現時点では Kasten アドオンでは利用可能なポイントインタイムリカバリー PITR はなく、RPO は Kasten 環境のスナップショットの頻度によって決まります。組織が Kasten に依存している場合、これは通常許容されますが、PITR が必要な場合は、オブジェクトストアのネイティブ継続バックアップ方法を検討することをお勧めします。

6.7.3 ヴェレロ

Velero は、Kubernetes クラスターリソースと永続ボリュームを安全にバックアップ、復元、ディザスターリカバリーを実行、および移行するためのオープンソースツールです。詳細については、Velero documentation を参照してください。 CloudNativePG クラスターの EDB Postgres AI との Velero 互換性を有効にするには、`velero` 値をクラスター仕様の `k8s.enterprisedb.io/addons` アノテーションに追加します。例

```
kind: Cluster
metadata:
  name: one-instance
  annotations:
    k8s.enterprisedb.io/addons: ["velero"]
    k8s.enterprisedb.io/snapshotAllowColdBackupOnPrimary: enabled
spec:
  instances: 1
  storage:
    size: 1Gi
  walStorage:
    size: 1Gi
```

クラスターが作成され正常になると、オペレーターは指定されたバックアップとして最も遠いアヘッドレプリカインスタンスを選択し、Velero 固有のバックアップフックをそのインスタンスにアノテーションとして追加します。

これらの annotations は、Velero がコマンドを実行して、バックアップする Postgres インスタンスを準備するために使用されます。

制限事項

バックアップ部分に関する限り、現在 Velero との Kubernetes 統合の EDB Postgres は **コールドバックアップ** のみをサポートしています。これらは**オフラインバックアップ**とも呼ばれます。これは、選択したレプリカが一時的にフェンスされ、外部バックアップアダプターが PVC グループ、つまり `PGDATA` ボリューム、および使用可能な場合は WAL ボリュームの物理的スナップショットを取得できることを意味します。

この短い時間フレームでは、スタンバイは読み取り専用の接続を受け入れることができません。使用可能なスタンバイがなく、通常は単一インスタンスのクラスターにいるため、アノテーション `k8s.enterprisedb.io/snapshotAllowColdBackupOnPrimary` が true に設定されている場合、Velero はプライマリを一時的にフェンスし、読み取り/書き込み操作の点でダウンタイムが発生します。このユースケースは、通常、開発環境に任されます。

リカバリーの面では、Velero との統合は、現時点ではスナップショットリカバリーのみをサポートしています。現時点では Velero アドオンでは利用可能なポイントインタイムリカバリー PITR はなく、RPO は Velero 環境のスナップショットの頻度によって決まります。組織が Velero に依存している場合、これは通常許容されますが、PITR が必要な場合は、オブジェクトストアのネイティブの継続的バックアップ方法を検討することをお勧めします。

バックアップ

デザイン、CloudNativePG クラスターの EDB Postgres AI は、前述のバックアップフックを利用できるようにすることを唯一の要件として、可能な限り多くのバックアップ機能を Velero にオフロードします。Kubernetes の EDB Postgres は、必要な構成をすべて透過的に設定し、残りは標準の Velero であるため、Velero を使用して Postgres クラスターをバックアップすることは、他のオブジェクトの場合と同様に簡単です。例

```
velero backup create mybackup \
  --include-namespaces mynamespace \
  -n velero-install-namespace
```

このコマンドは、構成されたオブジェクトストレージと構成されたスナップショット API を使用して標準の Velero バックアップを作成します。

リストア

バックアップと同様に、リカバリプロセスは標準の Velero 手順です。上記のパラメーターで作成されたバックアップから復元するコマンドは次のとおりです。

```
velero create restore myrestore \
  --from-backup mybackup \
  -n velero-install-namespace
```

第7章　REFERENCE

7.1 Operator capability levels

これらの機能は、CloudNativePG クラスターの EDB Postgres AI によって実装され、Operator SDK definition of Capability Levels を使用して分類されました

フレームワーク。

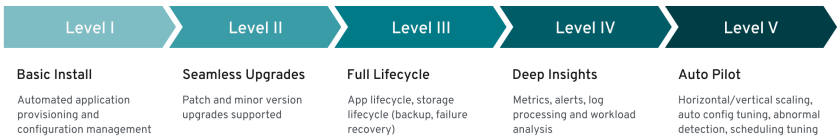


図 1: Operator Capability Levels

各機能レベルは、オペレーターが提供する特定の管理機能のセットに関連付けられています。

1. 基本的なインストール
2. シームレスなアップグレード
3. 完全なライフサイクル
4. 深い洞察
5. オートパイロット

> **注釈**
>
> 私たちは、このフレームワークを、オペレーターでの将来の作業と実装のガイドと考えます。

7.1.1 レベル 1 基本インストール

機能レベル 1 には、オペレーターのインストールと構成が含まれます。このカテゴリには、オペレーターや PostgreSQL クラスター構成との対話方法の改善など、使いやすさとユーザーエクスペリエンスの強化が含まれています。

宣言的構成を介した Operator の展開

オペレーターは、PostgreSQL ライフサイクルの管理に必要なコア `CustomResourceDefinition` オブジェクトを定義する Kubernetes マニフェストを使用して、宣言的な方法でインストールされます。これらには次のものが含まれます。

- **構成とトポロジ** `Cluster` 、`Pooler` 、`ImageCatalog` 、および `ClusterImageCatalog` 。
- **ID とスキーマ** `DatabaseRole` および `Database` 。
- **事業継続性** `Backup` 、`ScheduledBackup` 、`Publication` 、および `Subscription` 。
- **ランタイムオーケストレーション** `FailoverQuorum` 自動フェイルオーバー中にコンセンサスのためにオペレーターが使用します。

宣言的構成を介した PostgreSQL クラスターの展開

`Cluster` カスタムリソースを使用して、完全な宣言的方法で PostgreSQL クラスターオペランドを定義します。PostgreSQL バージョンは、CR で定義されたオペランドコンテナイメージによって決定されます。これは、要求されたレジストリから自動的に取得されます。

オペレーターは、標準の Kubernetes リソース `Pod` 、`Service` 、`Secret` 、`ConfigMap` 、`PersistentVolumeClaim` 、`PodDisruptionBudget` 、`ServiceAccount` 、`RoleBinding` 、および `Role` を作成および管理することにより展開をオーケストレーションし、`Database` や 11 などのユーザー定義の CNP リソースを調整してデータベース環境を edb_tran します目的の状態と一致します。

オプションで、 `Cluster` と `Pooler` リソースの両方に既存のサービスアカウントを提供できます。この共有 ServiceAccount サポートにより、クラウドネイティブの ID プロバイダーとのシームレスな統合が可能になり、クラスターごとではなくインフラストラクチャレベルで IAM ロールと権限を管理できます。

CRD を介したオペランドイメージのオーバーライド

オペレーターは、PostgreSQL を含むオペランドコンテナイメージをサポートしています。`docker.enterprisedb.com` では、最新のコミュニティでサポートされているメジャーバージョンの最新の安定したマイナーバージョンがデフォルトですが、`Cluster` リソースで `.spec.imageName` 属性を設定することにより、これをオーバーライドできます。このダイレクト方法は、プライベートレジストリの `imagePullSecrets` をサポートし、タグと SHA256 ダイジェストの両方を使用してコンテナの不変性を保証します。

または、*image catalogs* を活用して、PostgreSQL メジャーバージョンを参照するだけで、イメージをより効果的に管理できます。このアプローチは、イメージサプライチェーンの管理を一元化するため、実稼働環境に優れています。

コアの PostgreSQL エンジンを超えて、イメージカタログでオペランドイメージとともに *extension volumes* を定義できるようになりました。これにより、データベースとその関連モジュールが常に互換性があり、バージョンが調整され、インフラストラクチャ全体で単一の凝集ユニットとして扱われることが保証されます。同じカタログメカニズムが *PgBouncer* を使用した接続プーリング も管理します

プーラーイメージ、イメージカタログを、オペレーターが管理する PostgreSQL スタック全体にわたる統合されたサプライチェーンプリミティブにします。

ラベルとアノテーション

クラスターのメタデータで定義されたラベルとアノテーションの継承をサポートするようにオペレーターを構成できます。目標は、Kubernetes インフラストラクチャでの CloudNativePG クラスター展開の EDB Postgres AI の構成を改善することです。

自己完結型インスタンスマネージャー

Patroni や Stolon などの Kubernetes クラスターポッド内の PostgreSQL インスタンスを調整する外部ツールに依存する代わりに、オペレーターは各ポッド内の/controller/manager という名前のファイルにオペレーター実行可能ファイルを注入します。アプリケーションは、基になる PostgreSQL インスタンスを制御し、PostgreSQL クラスタートポロジに基づいてインスタンスとポッドステータスを調整するために使用されます。インスタンスマネージャーは、プローブの kubelet によって呼び出される Web サーバーも起動します。 kubelet によって呼び出される Unix シグナルは、インスタンスマネージャーによってフィルタリングされます。必要に応じて、それらは postgres プロセスに転送され、外部イベントに対する高速で制御された反応が行われます。インスタンスマネージャーは Go で書かれており、外部依存関係はありません。

ストレージ構成

ストレージは、データベースワークロードの重要なコンポーネントです。オペレーターは、ストレージに関する Kubernetes のネイティブ機能とリソースを利用して、基になる Kubernetes 環境が提供できるものに基づいて、ワークロード要件に適切なストレージを選択するための十分な柔軟性を提供します。これは、パブリッククラウド環境で特定のストレージクラスを選択するか、CR の storage パラメーターの PVC テンプレートを介して生成された PVC を微調整することを意味します。

パフォーマンスの向上とより詳細な制御のために、クラスターのログ先行書き込み WAL、pg_wal としても知られるを別のボリューム、できれば別のストレージでホストすることを選択することもできます。ドキュメントの *Benchmarking* セクションでは、運用前にストレージとデータベースの両方をベンチマークする方法の詳細な手順を提供します。cnp プラグインに依存して、最適なパフォーマンスと信頼性を保証します。

レプリカ構成

オペレーターは、instances と呼ばれる単一のパラメーターを介してクラスター内のレプリカを検出します。1 に設定されている場合、クラスターは、レプリカのない単一のプライマリ PostgreSQL インスタンスで構成されます。1 より高い場合、オペレーターは自動フェイルオーバーを介した高可用性 HA、スイッチオーバー操作を介したローリング更新を含む instances -1 レプリカを管理します。

CloudNativePG クラスターの EDB Postgres AI は、高可用性クラスター内のすべてのレプリカのレプリケーションスロットを管理します。また、プライマリでユーザー定義の物理レプリケーションスロットをサポートし、論理デコードフェイルオーバーを有効にします。PostgreSQL 17 以降ではネイティブに sync_replication_slots を使用し、以前のバージョンの場合は pg_failover_slots 拡張機能を介して。

サービス構成

デフォルトでは、CloudNativePG クラスターの EDB Postgres AI は 3 つの Kubernetes 管理対象サービス を作成します

アプリケーションがネットワークを介してクラスターにアクセスする場合

- 読み取り/書き込み操作のプライマリを指すもの。
- 読み取り専用クエリーのレプリカを指すもの。
- 読み取り操作のインスタンスを指す汎用的なもの。

構成を介して読み取り専用および読み取りサービスを無効にできます。さらに、サービステンプレート機能を活用して、ロードバランサーを含むカスタムサービスリソースを作成し、Kubernetes 外部の PostgreSQL にアクセスできます。これは、DBaaS の目的で特に役立ちます。

ポッドセレクターを介した動的なネットワークアクセス制御

CloudNativePG クラスターの EDB Postgres AI は、`podSelectorRefs` の宣言的定義をサポートして、`pg_hba.conf` ルールを動的に管理します。ラベルセレクターを使用してクライアントポッドを識別することにより、オペレーターは自動的にエフェメラル IP アドレスを解決し、それに応じて PostgreSQL ホストベースの認証ルールを更新します。これにより、同じ名前空間内の許可されたワークロードのみがデータベースに接続できるようになり、手動の IP 管理または静的 CIDR レンジが必要なくなります。

データベース構成

オペレーターは、単一のデータベースで PostgreSQL クラスターをブートストラップするように設計されています。オペレーターは、読み取り書き込み、読み取り、および読み取り専用のワークロード用にプロビジョニングおよび管理される 3 つの Kubernetes サービスを介して、クラスターへのネットワークアクセスを透過的に管理します。構成より規約のアプローチを使用して、オペレーターは `app` と呼ばれるデータベースを作成します。デフォルトでは、同じ名前の通常の Postgres ユーザーが所有します。必要に応じて、ブートストラップの一部としてデータベース名とユーザー名の両方を指定できます。

`Database` CRD を使用して、*declarative database management* を介して追加のデータベースを作成または管理できます。拡張機能、スキーマ、外部データラッパー FDW、および外部サーバーもサポートしています。

クラスターを実行するための構成は必要ありませんが、 CR の `postgresql` セクションで PostgreSQL ランタイム構成と PostgreSQL ホストベースの認証ルールの両方をカスタマイズできます。

Postgres のロール、ユーザー、およびグループの構成

CloudNativePG クラスターの EDB Postgres AI は *comprehensive management of PostgreSQL roles, users, and groups* をサポート

2 つの宣言的メソッドを介して

- `DatabaseRole` CRD 推奨きめ細かいライフサイクル管理のためのスタンドアロンリソース。これには、Kubernetes リソースが削除されたときにデータベースロールをドロップするかどうかを定義する `databaseRoleReclaimPolicy` `retain` または `delete` をサポートが含まれています。
- `managed` スタンザ より単純な要件の場合、`Cluster` リソースの `.spec.managed.roles` セクション内でロールをインラインで定義できます。

どちらの方法も、ロール属性たとえば `login`、`superuser`、`connectionLimit` などの自動調整と、Kubernetes Secret を介した安全なバージョン化されたパスワード管理を提供します。パスワードは PostgreSQL に送信される前にオペレーター側で SCRAM-SHA-256 エンコードされるため、クリアテキスト値がサーバーログまたは拡張機能に到達することはありません。`DatabaseRole` は、オペレーターが自動的に生成および更新する TLS クライアント証明書をさらに要求でき、パスワードなしの `cert` 認証を有効にします。

Postgres 拡張機能の構成

CloudNativePG クラスターの EDB Postgres AI は、各ポッドに読み取り専用の *extension volumes* としてマウントすることにより、PostgreSQL 拡張機能を管理するための宣言サポートを提供します。バージョン 1.29 以降、推奨されるアプローチは *ImageCatalogSpec* を活用することです。これにより、オペレーターは PostgreSQL バージョンに基づいてイメージ参照とディレクトリパスを自動的に解決できます。イメージボリュームには、`extension_control_path` パラメーター PostgreSQL 18 以降と `ImageVolume` 機能 Kubernetes 1.35 以降、またはフィーチャーゲート付き 1.33 以降が必要です。これらの要件が満たされない場合、拡張子をオペランドイメージに直接含める必要があります。システムで利用可能になると、*Postgres* **データベースの宣言的管理**

`CREATE EXTENSION` SQL ライフサイクルを自動化します。

ポッドのセキュリティ標準

InfoSec 要件の場合、オペレーターはコンテナに特権モードを必要としません。読み取り専用のルートファイルシステムを適用して、オペレーターとオペランドの両方のポッドのコンテナの不変性を保証します。また、必要なセキュリティコンテキストも明示的に設定します。

Red Hat OpenShift では、Cloud Native PostgreSQL は最も制限的な restricted で実行されます。ポッドの実行を名前空間が割り当てられた UID および SELinux コンテキストに制限することを目的としています。

アフィニティ

クラスターの `affinity` セクションでは、ポッドと永続ボリュームなどの関連リソースを Kubernetes クラスターのノード全体でスケジュールする方法を微調整できます。特に、オペレーターは以下をサポートしています。

- ポッドのアフィニティとアンチアフィニティ
- ノードセレクター
- テイントと許容範囲

トポロジスプレッドコンストレイン

クラスターの `topologySpreadConstraints` セクションにより、トポロジ全体でポッドのスケジューリングの追加制御が有効になり、アフィニティとアンチアフィニティが提供できる内容が強化されます。

ライセンスキー

オペレーターは、ライセンスキーのサポートが付属しており、キーが存在しない場合のデフォルト動作をプログラム的に定義する可能性があります。クラウドネイティブの PostgreSQL は、展開されたクラスターごとに暗黙の 30 日間の試用ライセンスを作成するようにプログラムされています。ライセンスキーは、オペレーターが非対称キー技術を使用して検証できる署名文字列です。コンテンツは、タイプ、製品、有効期限、および必要に応じて、クラスター識別子名前空間と名前、インスタンス数、プルするオペレーターがシークレットとして使用する資格情報を含む JSON オブジェクト保護されたコンテナレジストリからイメージをダウンします。有効期限を過ぎると、ライセンスキーが復元されるまで、オペレーターは調整プロセスを停止します。

コマンドラインインターフェイス

CloudNativePG クラスターの EDB Postgres AI には、独自のコマンドラインインターフェイスがありません。 cnp と呼ばれるプラグインを提供することにより、Kubernetes に最適なコマンドラインインターフェイス kubectl に依存しています。このプラグインは、PostgreSQL クラスター管理エクスペリエンスを強化および簡素化します。

クラスターの現在のステータス

オペレーターは、クラスターの観察されたステータスで CR のステータスセクションを継続的に更新します。PostgreSQL クラスター全体のステータスは、各ポッドで実行されているインスタンスマネージャーによって継続的に監視されます。インスタンスマネージャーは、制御される PostgreSQL インスタンスに必要な変更を適用して、クラスターの必要なステータスに収束します。たとえば、クラスターステータスがポッド-1 がプライマリであることをレポートする場合、ポッド -1 は自分自身を昇格させる必要がありますが、他のポッドはポッド -1 に従う必要があります。

オペレーターの認証局

オペレーターは、自分自身の認証局を作成します。Webhook サーバーが使用するリーフ証明書を作成し、オペレーター認証局で署名します。この証明書は、Kubernetes API サーバーとオペレーター間の安全な通信を保証します。

クラスターの認証局

オペレーターは、すべての PostgreSQL クラスターの証明機関を作成します。この認証局は、ストリーミングレプリケーションスタンバイサーバーパスワードの代わりにを含むクライアントの認証のための TLS 証明書を発行および更新するために使用されます。クライアント証明書のカスタム認証局のサポートは、シークレットを介して利用できます。これには、cert-manager との統合も含まれます。証明書は、kubectl の cnp プラグインを使用して発行できます。

TLS 接続

オペレーターは、TLS / SSL 接続を透過的にサポートし、クラスターの認証局を使用してクライアント/サーバー通信を暗号化し、セキュリティを向上させます。カスタムサーバー証明書のサポートは、シークレットを介して利用できます。これには、cert-manager との統合も含まれます。

ストリーミングレプリケーションの証明書認証

スタンバイサーバーからのストリーミングレプリケーション接続を承認するために、オペレーターは TLS クライアント証明書認証に依存します。この方法は、パスワードおよびシークレットに依存する代わりに使用されます。

継続的な構成管理

オペレーターを使用すると、PostgreSQL 構成の `Cluster` リソース YAML セクションに変更を適用できます。構成オプションによっては、すべてのインスタンスが適切にリロードまたは再起動されていることも確認します。

> **注釈**
>
> *ALTER SYSTEM* による変更は検出されません。つまり、クラスター状態が適用されません。

既存の PostgreSQL データベースのインポート

オペレーターは、オフライン移行を使用して、Kubernetes の CloudNativePG クラスタークラスターの新しい EDB Postgres AI に既存の Postgres データベースをインポートする宣言的方法を提供します。同じ機能は、PostgreSQL データベースのオフラインメジャーアップグレードもカバーしています。オフラインとは、データベースがインポートされるまで、アプリケーションがソースで書き込み操作を停止する必要があることを意味します。この機能は、`initdb` ブートストラップ方法を拡張して、別の PostgreSQL データベースで使用可能なデータの論理スナップショットを使用して、新しい PostgreSQL クラスターを作成します。このデータは、スーパーユーザー接続を介してネットワークを介してアクセスできます。インポートは、サポートされている Postgres のバージョンからです。これは、操作の一部であるすべてのデータベース、および要求に応じてロールの新しいクラスタープライマリから実行される `pg_dump` および `pg_restore` に依存しています。

PostGIS クラスター

CloudNativePG クラスターの EDB Postgres AI は、 *PostGIS*

地理データベースのオープンソース拡張機能。この拡張機能は、PostgreSQL の最も人気のある拡張機能の 1 つです。

PostgreSQL の基本的な LDAP 認証

オペレーターを使用すると、 LDAP authentication section of the PostgreSQL documentation で説明しているように、_simplebind_または_search+bind_モードを使用して、PostgreSQL クライアントの LDAP 認証を構成できます。

複数のインストール方法

オペレーターは、`kubectl apply` を介して Kubernetes マニフェストを介してインストールでき、パブリックおよびプライベートクラウド環境での従来の Kubernetes インストールで使用できます。さらに、Red Hat の OpenShift Container Platform を介して展開できます。オペレーターの Helm チャートも利用できます。

構成より規約

オペレーターは、規約より構成パラダイムをサポートし、標準のデフォルト値を決定しながら、オーバーライドおよびカスタマイズができます。数行の YAML コードで `Cluster` CRD を使用して、PostgreSQL クラスターのデプロイメントを指定できます。

7.1.2 レベル 2 シームレスなアップグレード

機能レベル 2 は、オペレーターと実際のワークロード、この場合 PostgreSQL サーバーの更新を有効にすることに関するものです。これには、PostgreSQL のマイナーリリース更新通常はセキュリティとバグ修正が含まれ、メジャーオンラインアップグレードが含まれます。

オペレーターのアップグレード

オペレーターのアップグレードはシームレスで、新しい展開として実行できます。コントローラをアップグレードした後、展開されたすべての PostgreSQL クラスターのローリング更新が開始されます。すべてのクラスターを同時に更新するか、時間の経過とともにアップグレードを分散することを選択できます。

インスタンスマネージャーのインジェクションのおかげで、オペレーターのアップグレードにオペランドの変更が必要ないため、オペレーターは古いバージョンを管理できます。

さらに、CloudNativePG クラスターの EDB Postgres AI は **インスタンスマネージャーのインプレース更新** をサポート

オペレーターのアップグレード後。インプレース更新では、ローリング更新またはその後のクラスターのスイッチオーバーは必要ありません。

管理対象ワークロードのアップグレード

オペランドは、`Cluster` リソースを更新することにより、宣言的アプローチを介してアップグレードされます。具体的には`.spec.imageName` パラメーターを介して、または**イメージカタログを使用する** のバージョン参照を更新します。このプロセスは、通常、セキュリティパッチまたは新しい PostgreSQL マイナーバージョンによってトリガーされます。スタンバイサーバーを使用するクラスターでは、オペレーターはレプリカから開始してローリング更新を実行します。既存のポッドを削除し、基になるストレージを再利用しながら、更新されたイメージを使用して新しいポッドに置き換えます。`primaryUpdateStrategy` によっては、オペレーターは元のプライマリ `unsupervised` を更新する前にスイッチオーバーを自動的に実行するか、`cnp` プラグイン `supervised` を介してユーザーが開始する手動スイッチオーバーを待機できます。この戦略により、組織は、スイッチオーバーに必要なワークロードに応じて数秒から分の範囲の短いダウンタイムと、ビジネス要件のバランスを保つことができます。

PostgreSQL のオフラインインプレースメジャーアップグレード

CloudNativePG クラスターの EDB Postgres AI は、より高い PostgreSQL メジャーバージョンを使用した新しいオペランドコンテナイメージが適用されるときに、宣言的オフラインインプレースメジャーアップグレードをサポートします。このアップグレードは、`.spec.imageName` を直接更新することにより、またはイメージカタログ内の新しいメジャーバージョンを選択することによりトリガーできます。イメージカタログの使用は、関連する *extension volumes* が新しい PostgreSQL メジャーバージョンと互換性のあるバージョンに同時に更新されるため、特に有益です。

プロセス中に、オペレーターはすべてのクラスターポッドをシャットダウンしてデータの一貫性を維持し、ジョブを開始してアップグレード条件を検証し、`pg_upgrade` を実行します。このジョブは、レプリカを再作成する前に、`PGDATA` 、WAL ファイル、およびテーブルスペースに必要な新しいディレクトリを作成します。この構造化されたワークフローは、メジャーバージョン移行の信頼できるパスを提供し、ユーザーが障害後にイメージを元に戻したときの自動ロールバッククリーンアップをサポートします。イメージボリューム拡張機能を使用するクラスターもサポートされています。ソースバージョンとターゲットバージョ

ンの拡張イメージは、アップグレードジョブ中にサイドバイサイドでマウントされるため、古いサーバーはライブラリを保持し、失敗したアップグレードはきれいに元に戻ります。

アップグレード中にクラスターの可用性ステータスを表示する

任意の時点で、クラスターの高可用性ステータスたとえば、`Setting up primary` 、`Creating a new replica`、`Cluster in healthy state`、`Switchover in progress`、`Failing over`、`Upgrading cluster`、および `Upgrading Postgres major version` などを伝えます。

7.1.3 レベル 3 完全なライフサイクル

機能レベル 3 では、オペレーターはビジネスの継続性とスケーラビリティの側面を管理する必要があります。

_ディザスターリカバリー_は、データベースのバックアップとリカバリーの両方が正常に動作することを必要とするビジネス継続コンポーネントです。出発点として、目標は リリース候補の目的<を達成することです。5 分、長期目標は、RPO=0 のバックアップソリューションを実装することです。_高可用性_は、ビジネス継続性のもう 1 つの重要なコンポーネントです。PostgreSQL ネイティブ物理レプリケーションとホットスタンバイレプリカを介して、オペレーターはフェールオーバーとスイッチオーバー操作を実行できます。この領域には、次の機能の強化が含まれます。

- 同期レプリケーション、カスケードレプリケーションクラスターなどの PostgreSQL 物理レプリケーションの制御
- 接続プーリング、pgBouncer を使用した接続プーリングレイヤーを介してパフォーマンスと制御を向上させます

PostgreSQL WAL アーカイブ

オペレーターは、オブジェクトストア AWS S3 および S3 互換、Azure Blob Storage、Google Cloud Storage、および MinIO のようなゲートウェイへの WAL ファイルの PostgreSQL 継続的アーカイブをサポートしています。

WAL アーカイブは、クラスター定義の `backup` パラメーターを介して、クラスターレベルで宣言的に定義されます。これは、S3 プロトコルの宛先 URL たとえば、AWS S3 バケットの特定のフォルダーをポイントするなど、オプションで汎用エンドポイント URL を指定することにより実行されます。

継続的バックアップの前提条件である WAL アーカイブには、それ以上のユーザーアクションは必要ありません。オペレーターは、`barman-cloud-wal-archive` に依存して WAL ファイルを定義されたエンドポイントに出荷するように `archive_command` を透過的に設定します。圧縮アルゴリズムと、アーカイブ内の WAL ファイルを同時にアップロードする並列ジョブの数を決定できます。さらに、`Instance Manager` は、WAL ファイルの最初のセットの出荷を開始する前に `barman-cloud-check-wal-archive` コマンドを実行して、アーカイブの宛先の正しさをチェックします。

PostgreSQL のバックアップ

CloudNativePG クラスターの EDB Postgres AI は、PostgreSQL のネイティブ物理バックアップメカニズムつまりベースバックアップと継続的 WAL アーカイブを使用してアプリケーションレベルのバックアップを管理するためのプラガブルインターフェイス CNPG-I を提供します。このデザインにより、一貫性とパフォーマンスを保証しながら、柔軟性と拡張性が可能になります。

CloudNativePG クラスターコミュニティの EDB Postgres AI は、Barman Cloud Plugin を公式にサポートしています。これにより、オブジェクトストアへの継続的な物理バックアップと、完全なポイントインタイムリカバリー PITR 機能が有効になります。

CNPG-I プラグインに加えて、CloudNativePG クラスターの EDB Postgres AI は、基になるストレージクラスと CSI ドライバーでサポートされている場合、Kubernetes ボリュームスナップショットを使用したバックアップもネイティブにサポートします。

ベースバックアップは 2 つの方法で開始できます。

- オンデマンド、`Backup` カスタムリソースを使用
- `ScheduledBackup` カスタムリソースを使用し、cron のようなスケジュール形式でスケジュール

ボリュームスナップショットは Kubernetes API を活用し、速度とストレージ効率の点で非常に大規模なデータベース VLDB に特に効果的です。

ボリュームスナップショットと CNPG-I ベースのバックアップは両方とも以下をサポートしています。

- ホットバックアップ PostgreSQL の実行中に取得され、中断を最小限に抑えます。
- コールドバックアップ必要に応じて、PostgreSQL を一時的に停止して、完全に一貫したスナップショットを保証することにより実行されます。

スタンバイからのバックアップ

オペレーターは、データベースの リリース候補の目的 に影響を与えることなく、スタンバイへのベースバックアップのオフロードをサポートします。これにより、標準のデータベース操作のためにプライマリにリソース、特に I/O を保存できます。

バックアップからの完全な復元

オペレーターを使用すると、ボリュームスナップショット、オブジェクトストア、またはプラグインを介して、既存のアクセス可能なバックアップから新しいクラスターとその設定をブートストラップできます。

ブートストラッププロセスが完了すると、オペレーターはリカバリモードでインスタンスを起動します。指定されたアーカイブから使用可能なすべての WAL ファイルを再生し、リカバリを終了してプライマリとして起動します。次に、オペレーターは、要求された数のスタンバイインスタンスをプライマリからクローン作成します。CloudNativePG クラスターの EDB Postgres AI は、アーカイブからの並列 WAL 取得をサポートしています。

バックアップからのポイント インタイム リカバリー PITR

オペレーターを使用すると、既存のバックアップをタイムスタンプ、ラベル、またはトランザクション ID で定義された特定の時点にリカバリーすることにより、新しい PostgreSQL クラスターを作成できます。この機能は、完全なリストアのものの上に構築され、PostgreSQL for PITR で使用可能なすべてのオプションをサポートしています。

同期レプリケーションを介したゼロデータ損失クラスター

CloudNativePG クラスターの EDB Postgres AI は、同期レプリケーションとオプションの *Failover Quorum* クォーラムベースのフェールオーバー を組み合わせることにより、_zero-data-lossRPO=0_高可用性クラスターを有効にします。任意の時点で使用可能な同期スタンバイレプリカの数を定義でき、フェールオーバーのみが保証されますデータが安全にレプリケートされると続行されます。クォーラムベースと優先度ベースの両方の同期レプリケーションがサポートされており、オペレーターは、耐久性とパフォーマンスの要件に一致するように `synchronous_standby_names` 設定を構成する際に完全な柔軟性を提供します。

レプリカクラスター

ネイティブストリーミングとカスケードレプリケーションの力を利用して、PostgreSQL クラスター用の堅牢なクロス Kubernetes クラスタートポロジを確立します。`replica` オプションを使用すると、同じメジャーバージョンの別の PostgreSQL ソースからデータを一貫してレプリケートするように自律クラスターを構成できます。WAL ファイルを取得するための WAL アーカイブ、または 2 つのエンドポイント間の TLS を介した直接ストリーミング接続にアクセスできる場合、このソースはどこにでもあります。

特に、ソースの PostgreSQL インスタンスは、物理的設定か仮想設定かにかかわらず、Kubernetes 環境の外部に存在できます。

レプリカクラスターは、ボリュームスナップショット、リカバリオブジェクトストア Barman Cloud バックアップ形式を使用、`pg_basebackup` を使用したストリーミングなどのさまざまな方法を介してインスタンス化できます。WAL ファイルシッピングと WAL ストリーミングの両方がサポートされています。レプリカクラスターの展開により、Kubernetes 内の PostgreSQL データベースのビジネス継続体制が大幅に向上し、マルチプルのデータセンターに拡張され、ハイブリッドおよびマルチクラウドのセットアップが促進されます。Kubernetes フェデレーションネイティブ機能を想定していますが、データセンター間の手動スイッチオーバーは引き続き必要です。

さらに、柔軟性は、プライマリクラスターから意図的に遅延して作成される遅延レプリカクラスターにまで及びます。この意図的な遅延は、誤った `DELETE` または `UPDATE` SQL 操作などの意図しないエラーが発生した場合の復旧時間目標 *RTO* を最小限に抑えることを目的としています。

分散データベーストポロジ

レプリカクラスターを活用して 分散データベーストポロジ を定義する

さまざまな Kubernetes クラスターにまたがる PostgreSQL に対応し、ハイブリッドおよびマルチクラウドの展開を促進します。CloudNativePG クラスターの EDB Postgres AI を使用すると、次のような強力な機能を獲得できます。

- **宣言的プライマリ制御**どの PostgreSQL クラスターがプライマリとして機能するかを簡単に指定します。
- **シームレスなプライマリスイッチオーバー**現在のプライマリを簡単に降格し、通常は別のリージョンにある別の PostgreSQL クラスターを昇格させます。以前のプライマリを再クローンする必要はありません。

このセットアップは、2 つ以上のリージョンにまたがって効率的に動作でき、レプリケーションをオブジェクトストアに完全に依存でき、最大 5 分の RPO 目標リカバリポイントを保証します。この高度な機能は、CloudNativePG クラスターの EDB Postgres AI が独自に提供し、多様な環境全体で堅牢なデータの完全性と継続性を保証します。

テーブルスペースのサポート

CloudNativePG クラスターの EDB Postgres AI は、個々の永続ボリュームの宣言的定義を容易にすることにより、PostgreSQL テーブルスペースの堅牢なサポートをシームレスに統合します。この革新的な機能により、さまざまなストレージデバイスに I/O 操作を効率的に分散できます。CloudNativePG クラスターの EDB Postgres AI は、テーブルスペースの透過的なオーケストレーションを介して、PostgreSQL データベースのパフォーマンスとスケーラビリティを強化し、クラウドネイティブ環境で大規模なデータストレージを管理するための合理化され最適化されたエクスペリエンスを保証します。一時テーブルスペースのサポートも含まれています。

カスタマイズ可能なスタートアップ、ライブネス、および準備状況プローブ

CloudNativePG クラスターの EDB Postgres AI は、Kubernetes kubelet によって管理される PostgreSQL コンテナのスタートアップ、ライブネス、および準備状況プローブを構成します。これらのプローブは、インスタンスマネージャーの Web サーバーによって公開される`/startupz`、`/healthz`、および`/readyz`エンドポイントと対話して、ポッドの状態と準備状況をモニタします。

すべてのプローブはデフォルト設定で構成されますが、特定のニーズを満たすように完全にカスタマイズでき、環境やワークロードに合わせて微調整が可能です。

詳細な構成オプションと高度な使用方法については、*Postgres Instance Manager* ドキュメントを参照してください。

ローリング展開

オペレーターは、ローリング展開をサポートして、ダウンタイムを最小限に抑えます。PostgreSQL クラスターが一般に公開されている場合、サービスは、初期化または更新中に使用可能なポッドにのみ読み取り専用トラフィックをロードバランシングします。

レプリカのスケールアップとスケールダウン

演算子を使用すると、PostgreSQL クラスター内のインスタンスの数をスケールアップおよびダウンできます。新しいレプリカはプライマリサーバーから起動され、クラスターの HA インフラストラクチャに参加します。CRD は、`kubectl scale`コマンドを使用できるようにする「scale」サブリソースを宣言します。また、scale サブリソースは、管理対象インスタンスポッドのラベルセレクターを公開し、オートスケーラーがそれらを検出できるようにします。*Vertical Pod Autoscaler integration* を参照してください。

推奨される使用および *Horizontal Pod Autoscaler integration*

HPA に適用される注意事項については、

Kubernetes ノードのメンテナンスウィンドウと PodDisruptionBudget

オペレーターは`PodDisruptionBudget`リソースを作成して、同時中断数を 1 つのプライマリインスタンスに制限します。この構成により、メンテナンス操作でクラスター内のすべてのポッドが削除されるのを防ぎ、指定された数のインスタンスを作成できます。PodDisruptionBudget は、ノードドレイン操作中に適用され、クラスターサービスの中断を防ぎます。

この戦略は、ストレージがすべてのワーカーノードで共有される Kubernetes クラスターには正しいですが、ローカルストレージを使用するクラスター、またはプライベートクラウドにインストールされたクラスター

には、最適なソリューションではない可能性があります。オペレーターを使用すると、メンテナンスウィンドウを指定し、基になるノードのエビクションへの反応を構成できます。メンテナンスウィンドウセクションの ReusePVC オプションを使用すると、使用するストラテジーを指定できます。削除されたインスタンスの別の PVC に新しいストレージを割り当てるか、基になるノードが再び利用できるようになるのを待ちます。

フェンシング

フェンシングは、PostgreSQL クラスターの 1 つ以上、またはすべてのインスタンスに障害があると思われる場合に、データを保護するプロセスです。インスタンスがフェンスされると、PostgreSQL サーバープロセスは保証的にシャットダウンされますが、ポッドは実行されたままです。これにより、フェンスが解除されるまで、ポッド上のデータは PostgreSQL によって変更されず、デバッグとトラブルシューティングの目的でファイルシステムを調査できます。

ハイバネーション

CloudNativePG クラスターの EDB Postgres AI は *hibernation of a running PostgreSQL cluster* をサポート

`k8s.enterprisedb.io/hibernation` アノテーションを介して宣言的な方法で。 `Hibernation` は、データベース PVC を維持しながらデータベースポッドを削除することにより、CPU パワーを節約できます。この機能は、0 インスタンスへのスケーリングをシミュレートします。

ポッドの永続ボリュームストレージの再利用

ユーザーによって削除されたポッド、または Kubernetes メンテナンスオペレーションによって削除されたポッドをオペレーターが作成する必要がある場合、`PersistentVolumeClaim` 可能な場合は再利用します。この機能により、プライマリからデータを再度クローン作成する必要がなくなります。

CPU とメモリの要求と制限

オペレーターを使用すると、管理者は、マニフェストの `resources` セクションでクラスターのポッドによるリソースの使用を制御および管理できます。特に、CPU と RAM の両方に `requests` および `limits` 値を設定できます。 `Cluster` は、scale サブリソースを介してラベルセレクターを公開するため、**垂直ポッドオートスケーラー *VPA* との統合** のターゲットとして使用することもできます。

推奨専用モードで、これらの値のサイズの提案を取得します。

PgBouncer を使用した接続プーリング

CloudNativePG クラスターの EDB Postgres AI は、PostgreSQL で最も人気のあるオープンソース接続プーラーの 1 つである *PgBouncer* **を使用した接続プーリング**を使用した接続プーリングのネイティブサポートを提供します。アーキテクチャの観点から、PgBouncer 接続プーラーのネイティブ実装は、データベースにアクセスするための新しいレイヤーを導入します。これにより、インスタンスへのクエリフローが最適化され、基になる PostgreSQL リソースの使用がより効率的になります。PostgreSQL サービスに直接接続する代わりに、アプリケーションは PgBouncer サービスに接続し、既存の接続の再利用を開始できるようになりました。PgBouncer イメージは、`ImageCatalog` または `ClusterImageCatalog` `spec.pgbouncer.imageCatalogRef` を介して一元管理でき、オプションで `Pooler` メトリックエンドポイントを TLS で提供できます。

論理レプリケーション

CloudNativePG クラスターの EDB Postgres AI は、 `Publication` および `Subscription` カスタムリソース定義を使用して、宣言的方法で PostgreSQL の論理レプリケーションをサポートします。

論理レプリケーションは、次の場合に特に役立ちます。

- オンラインデータの移行最小限のダウンタイムで外部の PostgreSQL インスタンスまたはパブリック DBaaS ソリューションからデータを移動します。
- PostgreSQL のメジャーアップグレードメジャーバージョン間のほぼゼロのダウンタイムのアップグレードを促進します。
- 選択的なデータ分散レポートまたはローカライズされたワークロードのために、さまざまなクラスターにわたって特定のテーブルまたはデータセットを複製します。

Kubernetes の外部バックアップツールとの統合

CloudNativePG クラスターの EDB Postgres AI は、以下と統合するためのアドオンを提供します。

- Kasten 、Kubernetes の非常に人気のあるデータ保護ツール、クラウドネイティブアプリケーションのバックアップと復元、ディザスターリカバリー、およびアプリケーションモビリティを可能にします
- Velero Kubernetes リソースと永続ボリュームをバックアップおよび復元するための非常に人気のオープンソースツールと

OpenShift API for Data Protection (OADP)

また、 *external backup adapter add-on*

は、CloudNativePG クラスターの EDB Postgres AI をバックアップ用のサードパーティツールに統合するための汎用インターフェイスを提供します。

7.1.4 レベル 4 深い洞察

機能レベル 4 は、_observability_モニタリング、アラート、トレンド、ログ処理に関するものです。これには、Prometheus、Grafana、Fluent Bit などの外部ツール、および JSON 形式で直接エラーログを出力するための PostgreSQL エンジンの拡張機能の使用が含まれる場合があります。

CloudNativePG クラスターの EDB Postgres AI は、柔軟なモニタリングとロギングのための業界標準のコミュニティで認められたツールと簡単に統合するために必要なすべてを提供するように設計されました。

構成可能なクエリを備えた Prometheus エクスポータ

インスタンスマネージャーは、プラグ可能なフレームワークを提供します。`metrics` ポート 9187 でリッスンする独自の Web サーバーを介して、 Prometheus モニタリングおよびアラートツールのメトリックをエクスポートするエンドポイントを公開します。オペレーターは、postgres_exporter と互換性のある構文を使用して、`ConfigMap` または `Secret` オブジェクトとして定義されたカスタムモニタリングクエリーをサポートします。CloudNativePG クラスターの EDB Postgres AI は、コンテキストに統合および適応できる PostgreSQL の基本的なモニタリングクエリーのセットを提供します。

Grafana ダッシュボード

CloudNativePG クラスターの EDB Postgres AI には、PostgreSQL クラスターのすべての重要な側面を監視し、カスタマイズするためのベースとして使用できる Grafana ダッシュボードが付属しています。

JSON 形式の PostgreSQL エラーメッセージの標準出力ログ

すべてのログメッセージは JSON 形式で標準出力に配信されます。最初のレベルは、タイムスタンプ、ログレベル、および正規の PostgreSQL エラーメッセージチャネルの `postgres` などのログエントリのタイプの定義です。その結果、CloudNativePG クラスターの EDB Postgres AI によって管理されるすべてのポッドは、ソースデータ型として JSON をサポートするダウンストリームログ処理スタックと簡単かつ直接統合できます。

リアルタイムクエリモニタリング

CloudNativePG クラスターの EDB Postgres AI は、以下を透過的かつネイティブにサポートします。

- 必須の pg_stat_statements 、PostgreSQL サーバーによって実行されるすべての SQL ステートメントの計画と実行統計の追跡を有効にします
- auto_explain は、 `EXPLAIN` を手動で実行せずに、遅いステートメントの実行プランを自動的にログに記録する手段を提供します最適化されていないクエリーを追跡するのに役立ちます

監査

CloudNativePG クラスターの EDB Postgres AI を使用すると、データベースとセキュリティの管理者、監査人、およびオペレーターは、PostgreSQL の PGAudit と EDB Audit Logging 機能 EDB Postgres Advanced の場合を使用して、データベースアクティビティを追跡および分析できます。このようなアクティビティは JSON ログで直接フローし、Fluentd のような一般的なログブローカーを使用して正しいダウンストリームターゲットに適切にルーティングできます。

Kubernetes イベント

リソースの作成、ノードの削除、アップグレードなど、Kubernetes API で想定されているメジャーイベントを記録します。 `kubectl describe` 、`kubectl get events` コマンドを使用してイベントを表示できます。

7.1.5 レベル 5 自動操縦

機能レベル 5 は、オブザーバビリティレイヤーから現れた異常と洞察の発見を介した自動スケーリング、修復、および調整に焦点を当てています。

自己修復のための自動フェイルオーバー

プライマリで障害が検出された場合、オペレーターは、最も調整されたレプリカを新しいターゲットプライマリとして設定することにより、クラスターのステータスを変更します。その結果、各生きているポッドのインスタンスマネージャーは、必要なプロシージャを開始して、クラスターの要求されたステータスに自分自身を調整します。これは、新しいプライマリになるか、フォローすることにより実行されます。元のプライマリが復旧した場合、同じメカニズムにより、アプリケーションがそれに到達しないようにし、

サーバーで pg_rewind を実行し、スタンバイとして再起動することにより、スプリットブレインを回避します。オペレーターはさらに、ミューテックスとして機能するクラスターごとの Kubernetes Lease を介してプライマリプロモーションを調整し、任意の時点で最大 1 つのインスタンスがプロモートするように保証します。インスタンスはプライマリとして機能する前にリースを保持し、クリーンシャットダウンでリリースする必要があるため、レプリカ完全な TTL を待たずに昇格できます。同期フェイルオーバークォーラムが有効になっている場合、オペレーターの「オートパイロット」ロジックはさらに洗練されたものになります。レプリカのクォーラムがトランザクション状態を検証できない場合、プロモーションをアクティブにブロックします。これにより、プライマリとその同期スタンバイの両方に影響を与えるネットワークパーティションなど、複雑な障害シナリオ中の偶発的なデータの損失が防止されます。

スタンバイの自動再作成

スタンバイをホスティングするポッドが削除された場合、オペレーターはスタンバイサーバーを再作成する手順を開始します。

7.2 Frequently Asked Questions (FAQ)

7.2.1 Kubernetes での PostgreSQL の実行

**PostgreSQL のようなステートフルワークロードは Kubernetes では実行できないことは誰もが知っています。なぜ逆に言うのですか？*

2021 年 9 月の _independent research survey commissioned by the Data on KubernetesCommunity_ では、回答者の半数が運用ワークロードのほとんどを Kubernetes で実行していることが明らかになりました。90 % が Kubernetes がステートフルワークロードに対応する準備ができていると考えており、70 %が本番環境でデータベースを実行しています。Postgres のようなデータベース。しかし、彼らによると、知識ギャップ一般に Kubernetes とクラウドネイティブは急な学習曲線を持っていますや、Kubernetes オペレーターの品質など、重要な課題が残っています。後者は、CloudNativePG クラスターの EDB Postgres AI のようなオペレーターがプロジェクトの成功に大きく貢献すると考えられる理由です。

私たちのようなデータベース愛好家にとって、真のゲームチェンジャーは、_Kubernetes 1.14 in April 2019_ でのローカル永続ボリュームのサポートの導入でした。

CloudNativePG クラスターの EDB Postgres AI は、不変のアプリケーションコンテナに基づいて構築されています。それはどういう意味ですか？

マイクロサービスアーキテクチャパターンによると、コンテナは単一のアプリケーションまたはプロセスを実行するように設計されています。その結果、このようなコンテナイメージは、メインアプリケーションを単一のエントリポイントいわゆる PID 1 プロセスとして実行するようにビルドされます。

Kubernetes の用語では、アプリケーションはワークロードと呼ばれます。ワークロードは、Web アプリケーションサーバーのようなステートレスまたはデータベースのようなステートフルにすることができます。この概念を PostgreSQL にマッピングすると、不変アプリケーションコンテナは、実行され、単一の特定のバージョン不変コンテナイメージ内のバージョンに関連付けられている単一の「postgres」プロセスです。

SSH、systemd、または syslog などの他のプロセスは許可されません。

不変のアプリケーションコンテナは、依然としてコンテナを解釈して使用する非常に一般的な方法である可変システムコンテナとは対照的です。

不変とは、コンテナがその存続期間中に変更されないことを意味します。更新、パッチ、構成変更はありません。アプリケーションコードを更新するか、パッチを適用する必要がある場合は、新しいイメージをビルドして再展開します。不変性により、展開がより安全で反復可能になります。

詳細については、 _"Why EDB chose immutable application containers"_ を参照してください。

クラウドネイティブとはどういう意味ですか?

Cloud Native Computing Foundation は、用語「 _Cloud Native_ 」を定義しています。しかし、2ndQuadrant で CloudNativePG Cluster オペレーターの Cloud Native PostgreSQL / EDB Postgres AI を開始して以来、開発チームはクラウドネイティブを 3 つの主要コンセプトとして解釈してきました。

1. 人、および原則とプロセスに基づいた、既存の健全で、本物の、豊かな DevOps 文化。これにより、チームと組織がチームのチームとして継続的に変化し、結果と成果の提供を革新および加速できます。より安全、より効率、そしてより魅力的な方法でビジネスの価値を引き出します
2. 不変のアプリケーションコンテナに基づいたマイクロサービスアーキテクチャ
3. Kubernetes などのこれらのコンテナを管理およびオーケストレーションする方法

現在、コンテナオーケストレーションのデファクト標準は Kubernetes であり、クラウドネイティブアプリケーションの展開、管理、およびスケーラビリティを自動化します。

私たちの共鳴をもたらすクラウドネイティブのもう 1 つの定義は、_"Kubernetes Patterns", published by O'Reilly_ で Ibryam と Huß によって定義されたものです。

> コンテナ化されたマイクロサービスを大規模に自動化するための原則、パターン、ツール

ベアメタル Kubernetes で CloudNativePG クラスターの EDB Postgres AI を実行できますか?

はい、確かに。ベアメタルで Kubernetes を実行できます。また、ローカルに接続されたストレージを使用する 1 つ以上の物理ワーカーノードを PostgreSQL ワークロード専用にすると、最大かつ予測可能な I/O パフォーマンスを実現できます。

CloudNativePG クラスターの EDB Postgres AI の元になった実際のクラウドネイティブ PostgreSQL プロジェクトは、同じベアメタルサーバーでストレージと PostgreSQL を、最初に Linux で直接、次に Kubernetes 内でベンチマークした 2019 年のパイロットプロジェクトの後に生まれました。予想されたように、実験では、ローカル永続ボリュームを介して Kubernetes で実行されているコンテナによってもたらされたパフォーマンスへのごくわずかな影響のみが示されたため、クラウドネイティブイニシアチブを続行できます。

ファイルシステムレプリケーションの代わりに PostgreSQL レプリケーションを使用する必要があるのはなぜですか?

Architecture: Synchronizing the state をお読みください

セクション。

PostgreSQL をコンテナとして実行する代わりに、オペレーターを使用する必要があるのはなぜですか?

Kubernetes で PostgreSQL を実行するための最も基本的なアプローチは、Kubernetes での展開の最小単位であるポッドを使用し、レプリカのない Postgres コンテナを実行することです。Postgres データディレクトリをホスティングするボリュームはポッドにマウントされ、通常はネットワークストレージにあります。この場合、Kubernetes は問題が発生した場合にポッドを再起動するか、別の Kubernetes ノードに移動します。

最も洗練されたアプローチは、演算子を使用して PostgreSQL を実行することです。オペレーターは Kubernetes コントローラーの拡張機能であり、ビジネス継続性のコンテキストで複雑なアプリケーションがどのよう

に動作するかを定義します。演算子パターンは、この目的のための Kubernetes の現在最先端の技術です。オペレーターは、自動化されたプログラム的な方法で人間のオペレーターの作業をシミュレートします。

Postgres は複雑なアプリケーションであり、オペレーターはクラスターを展開する最初のステップだけでなく、予期しないイベントの後に適切に対応する必要があります。典型的な例は、フェイルオーバーの例です。

オペレーターは、自己修復、スケーラビリティ、レプリケーション、高可用性、バックアップ、リカバリー、更新、アクセス、リソース制御、ストレージ管理などの機能で Kubernetes に依存しています。また、ログ管理および監視インフラストラクチャへの PostgreSQL クラスターの統合も促進します。

CloudNativePG クラスターの EDB Postgres AI は、宣言的構成を介して PostgreSQL クラスターの目的の状態を定義できます。Kubernetes は、Kubernetes コントローラによって開始される調整ループを介して、インフラストラクチャの現在の状態が目的の状態と一致することを継続的に確認します。目的の状態と実際の状態が一致しない場合、調整ループは自己修復プロシージャーをトリガーします。そこで、CloudNativePG クラスターの EDB Postgres AI のようなオペレーターが登場します。

Postgres の他の演算子はありますか?

はい、もちろんです。そして、私たちのアドバイスは、決定を行う前に、それらをすべて見て、CloudNativePG クラスターの EDB Postgres AI と比較することです。これらのオペレーターのほとんどは、外部フェールオーバー管理ツール Patroni または同様のツールを使用し、StatefulSet に依存していることがわかります。

これは、GitHub での公開からの時系列での非完全なリストです。

- Crunchy Data Postgres Operator 2017
- Zalando Postgres Operator 2017
- Stackgres 2020
- Percona Operator for PostgreSQL 2021
- Kubegres 2021

関連する欠落エントリを PR としてご自由に報告してください。

> **注釈**
>
> Data on Kubernetes Community

メンテナーの一部を含むは、 Operator Feature Matrix と呼ばれるオペレーターをリストする独立したベンダー中立プロジェクトに取り組んでいます。

CloudNativePG クラスターの EDB Postgres AI は完全に宣言的演算子であると言います。それはどういう意味ですか？

最も簡単な方法は、命令的構成との違いを強調する例を介して宣言的構成を説明することです。命令的コンテキストでは、状態は順番に実行される一連のタスクとして定義されます。したがって、最初のインスタンスを作成し、レプリケーションの構成、2 番目のインスタンス、3 番目のインスタンスのクローンを作成することにより、3 ノードの PostgreSQL クラスターを取得できます。

宣言的アプローチでは、システムの状態は構成を使用して定義されます。つまり、2 つのレプリカを持つ PostgreSQL 18 クラスターがあります。このアプローチは、変更管理操作を非常に簡素化し、これらが Git の

ようなソース制御システムに保存されると、Infrascriptor as Code 機能が有効になります。そして、Kubernetes は、私たちの要求が常に満たされることを保証するため、展開よりもさらに進んでいます。

Kubernetes で PostgreSQL を実行するために必要なスキルは何ですか?

Kubernetes で PostgreSQL を実行するには、DevOps チームに PostgreSQL と Kubernetes の両方のスキルが必要です。最高のエクスペリエンスは、データベース管理者が Kubernetes のコア概念を理解し、Kubernetes 管理者と対話できる場合です。

私たちのアドバイスは、クラウドネイティブ PostgreSQL を完全に活用して、CNCF 認定プログラムから「Certified Kubernetes AdministratorCKA」ステータスを取得したいと考えているすべての人を対象としています。

CloudNativePG クラスターの EDB Postgres AI はステートフルセットを使用しないのはなぜですか?

CloudNativePG クラスターの EDB Postgres AI は、`StatefulSet` リソースに依存せず、代わりに、動的プロビジョニング用に選択したストレージクラスを活用して、基になる PVC を直接管理します。この決定の背後にある理由と詳細については、*Custom Pod Controller* セクションを参照してください。

7.2.2 高可用性

オペレーターポッドが停止するか、特定の時間使用できない場合、PostgreSQL クラスターはどうなりますか?

とりわけ、CloudNativePG Cluster オペレーターの EDB Postgres AI は、自己修復機能を担当します。そのため、オペレーターの停止中は利用できない場合があります。

ただし、停止が PostgreSQL クラスターが実行されているノードに影響を与えないと仮定すると、データベースは、関連する Kubernetes サービスを介して通常の操作を提供し続けます。さらに、各 PostgreSQL ポッド内で実行される **インスタンスマネージャーのインプレース更新** は引き続き動作し、ロギング、メトリックのエクスポート、WAL ファイルの継続的アーカイブなどのアクセサリサービスを含む、データベースサーバーがアップしていることを確認します。

要約すると

オペレーターの停止は、必ずしも PostgreSQL データベースの停止を意味するわけではありません。それは、DBA またはシステム管理者なしでデータベースを実行するようなものです。

CloudNativePG クラスターの EDB Postgres AI が、Patroni、repmgr、Stolon のようなフェールオーバー管理ツールに依存しないのは、なぜですか?

CloudNativePG クラスターの EDB Postgres AI を開発するチームの一部は、過去に repmgr に深く関わっていましたが、私たちは別のアプローチを取って Kubernetes コントローラーを直接拡張し、Kubernetes API サーバーに依存して Postgres のステータスを保持することにしました。クラスターを作成し、それを真実の唯一のソースとして使用します。

- 主に自動フェイルオーバーとスイッチオーバーを介して Postgres クラスターの高可用性を制御し、**インスタンスマネージャーのインプレース更新** と調整します
- Kubernetes サービス、つまりアプリケーションのエントリポイントを制御します

フェールオーバーに続いて、元のプライマリを新しいプライマリと手動で再同期する必要がありますか?

いいえ。オペレーターはそれを自動的に行い、 pg_rewind に依存して元のプライマリと新しいプライマリを同期します。

7.2.3 データベース管理

PostgreSQL を使用する必要があるのはなぜですか?

私たちは、PostgreSQL がデータベース領域で Linux がオペレーティングシステム領域で表すものと同等であると考えています。Postgres の現在の最新メジャーバージョンはバージョン 16 で、そのまま出荷されます。

- ネイティブストリーミングレプリケーション、物理的および論理的の両方
- 継続的なホットバックアップとポイント インタイム リカバリー
- 水平テーブルパーティショニングの宣言的パーティショニング。これは、単一のインスタンスの垂直スケーラビリティを向上させるデータベース分野で非常によく知られた技術です。
- 拡張性、地理データベース用の *PostGIS* のような拡張機能を使用
- 垂直スケーラビリティのためのパラレル クエリー
- JSON サポート、標準の SQL を介して照会される構造化データと非構造化データの両方のマルチモデルハイブリッドデータベースを解放します

など…

単一の PostgreSQL インスタンスでホストする必要があるデータベースはいくつありますか?

私たちをお勧めするのは、単一の PostgreSQL クラスタープライマリサーバーと複数のスタンバイサーバーとしてを目的とした単一のデータベース専用であり、単一のマイクロサービスアプリケーションによって完全に管理されることです。ただし、「postgres」スーパーユーザーを活用することにより、使用可能なリソースに応じて、必要な数のユーザーとデータベースを作成できます。

この推奨の理由は、マイクロサービスに基づいたクラウドネイティブの概念にあります。純粋なマイクロサービスアーキテクチャでは、マイクロサービス自分自身が排他的に管理するデータを所有する必要があります。これらはフラットファイル、キュー、キーバリューストア、またはこの場合、構造化データと非構造化データの両方を含む PostgreSQL リレーショナルデータベースです。一般的な考え方は、マイクロサービスのみがスキーマ管理と移行を含むデータベースにアクセスできるということです。

CloudNativePG クラスターの EDB Postgres AI は、そのままでこの方法で動作するように設計されており、デフォルトでアプリケーションユーザーと、前述のアプリケーションユーザーが所有するアプリケーションデータベースを作成します。

PostgreSQL インスタンスを単一のマイクロサービスが所有するデータベースに予約すると、以下が強化されます。

- リソース管理 PostgreSQL では、CPU、メモリの制限されたリソースは、一般にデータベースレベルではなくインスタンスレベルで処理されるため、ポッドレベルで Kubernetes リソース管理ポリシーと統合することが簡単になります
- 物理的継続バックアップとポイントインタイムリカバリー PITR PostgreSQL がインスタンスレベルで継続的バックアップとリカバリーを処理することを考えると、インスタンスごとに 1 つのデータ

ベースがあることにより、PITR 操作が簡素化され、リテンションポリシー管理が差別化され、バックアップのデータ保護が強化されます

- アプリケーションの更新さまざまなアプリケーションが所有する他のデータベースに影響を与えることなく、各アプリケーションが更新ポリシーを決定できるようにします
- データベースの更新各アプリケーションは、どの PostgreSQL バージョンを使用するか、および独立して、PostgreSQL の別のメジャーバージョンにアップグレードする時期、およびどのような条件カットオーバー時間などを決定できます。

Kubernetes を考慮しないためのデータベースサイズの上限はありますか?

いいえ、これに関する限り、Kubernetes は仮想マシンやベアメタルと変わらないためです。ただし、実際には、Kubernetes クラスターの使用可能なリソースに依存します。非常に大規模なデータベース VLDB に関するアドバイスは、Kubernetes ワーカーノードが専用のストレージを備えた単一の Postgres インスタンス専用であるシェアードナッシングアーキテクチャを検討することです。running PostgreSQL on bare metal Kubernetes with local persistentvolumes をベンチマークしたときに、この極端なアーキテクチャパターンが機能することが証明されました。テーブルスペースと水平パーティショニングは、データベースの垂直スケーラビリティを向上させるために使用できるデータモデリング手法です。

PostgreSQL クラスターでタイムゾーンを指定するにはどうすればよいですか?

公式ドキュメントで説明されているように、PostgreSQL はタイムゾーンを広範にサポートしています。

- Date time data types
- Client connections config options

タイムゾーンはセッション、トランザクション、および PostgreSQL のクエリの一部として使用することもできますが、非常に一般的な方法はそれらをグローバルに設定することです。CloudNativePG クラスターの EDB Postgres AI を使用すると、次の例のように`.spec.postgresql.parameters` セクションでクラスターレベルのタイムゾーンを構成できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: pg-italy
spec:
  instances: 1

  postgresql:
    parameters:
      timezone: "Europe/Rome"

  storage:
    size: 1Gi
```

タイムゾーンは次の方法で確認できます。

```
$ kubectl exec -ti pg-italy-1 -c postgres -- psql -x -c "SHOW timezone"
- [ RECORD 1 ]---------
```

(次のページに続く)

(前のページからの続き)

```
TimeZone | Europe/Rome
```

最高のビジネス継続性の成果を得るための推奨アーキテクチャは何ですか?

Architecture セクションで説明しているように、主な推奨事項は、Kubernetes が単一のクラスターで提供するネイティブ機能とリソースを活用して、可能な限りシェアードナッシングアーキテクチャを採用することです。

- アベイラビリティーゾーン同じ Kubernetes クラスター内の異なるアベイラビリティーゾーンにインスタンスを分散します。
- ワーカーノード その結果、Postgres インスタンスが別の Kubernetes ワーカーノードにあることを確認します
- ストレージ Postgres を実行している各ワーカーノードに専用のストレージを使用します

少なくとも 1 つのスタンバイ、できれば少なくとも 2 つを使用して、クラスターで同期レプリケーションを構成し、高可用性を実現するために **リリース候補の目的**=0 を導入できます。

アベイラビリティーゾーンがない場合-通常、オンプレミスインストールの場合-ワーカーノードとストレージに分離します。

ローカル/リージョンオブジェクトストアで継続的バックアップを適切に設定します。

単一の Kubernetes クラスター内にある同じアーキテクチャは、**レプリカクラスター** 機能を介して別の Kubernetes クラスター通常は別の地理的エリアまたはリージョンにレプリケートでき、世界規模でのディザスターリカバリーと高可用性を提供します。

デフォルトの最大 RPO5 分で十分な場合は、ストリーミング接続を提供せずに、プライマリオブジェクトストアの WAL アーカイブを使用して、他のリージョンのレプリカにフィードできます。

インスタンスはどのように停止または起動できますか?

フェンシングのアノテーションについて または **ハイバネーション** を参照してください。

CloudNativePG クラスターの EDB Postgres AI によって自動的に作成されるロールやデータベースなどのグローバルオブジェクトとは何ですか?

オペレーターは、アプリケーションのユーザーデフォルトで app と呼ばれます、と前述のユーザーが所有するアプリケーションのデータベースデフォルトで app と呼ばれます。

このようにして、開発者は_superuser_アクセスを必要とせずに app ユーザーを使用して移行を制御できるため、データベースはマイクロサービスを採用する準備が整います。

Teams は、:ref:`DatabaseRole` リソースを使用した宣言的ロール管理 <`DatabaseRole` リソースを使用した宣言的ロール管理>` 機能を介して読み取り書き込み操作用の別のユーザーを作成し、必要な\ ``GRANT`` をテーブルに割り当てることができます。

7.3 Troubleshooting

このページでは、Kubernetes クラスター展開の CloudNativePG クラスターの EDB Postgres AI をトラブルシューティングする方法に関する基本情報を見つけることができます。

7.3.1 始める前に

Kubernetes 環境

トラブルシューティングアクティビティに違いをもたらすのは、基になる Kubernetes システムに関する明確な情報を提供することです。

次のことを確認してください

- 使用している Kubernetes ディストリビューションとバージョン
- PostgreSQL が実行されているノードの仕様
- 運用に入る前に実行したストレージクラスとベンチマークを含む、実際の**その他の *S3* 互換オブジェクトストレージプロバイダー** について可能な限り。
- クラスターで使用している関連する Kubernetes アプリケーションつまり Prometheus、Grafana、Istio、Certmanager など。
- 継続的なバックアップの状況、特にそれが配置され、正常に動作している場合。そうでない場合、必ず **緊急バックアップ** を取ってください

潜在的な破壊的な操作を実行する前に

便利なユーティリティ

トラブルシューティングの為、必須の `kubectl` ユーティリティに加えて、次のプラグイン/ユーティリティをシステムで使用できることをお勧めします。

- `kubectl` の場合 *CNPG-I*
- jq 、軽量で柔軟なコマンドライン JSON プロセッサー
- grep 、1 つ以上の入力ファイルで、指定されたパターンとの一致を含む行を検索します。これは、ほとんどの*nix ディストリビューションで既に利用可能です。Windows OS を使用している場合は、`grep` の代替として findstr を使用するか、wsl を直接使用できます。

そして、優先する*nix ディストリビューションをインストールし、上記のツールを使用します。

7.3.2 最初の一歩

クラスターまたはインストールの概要をすばやく取得するために、`kubectl` プラグインは使用する主なツールです。

1. *status subcommand* は、クラスターの概要を提供します
2. *report subcommand* は、クラスターとオペレーター展開のマニフェストを提供します。`--logs` オプションを使用してログを含めることもできます。プラグインを介して生成されたレポートには、完全なクラスターマニフェストが含まれます。

プラグインは、パッケージを介してエアギャップシステムにインストールできます。完全な手順については、*plugin document* を参照してください。

7.3.3 バックアップはありますか？

プラグインでクラスターマニフェストを取得した後、バックアップがセットアップされ動作しているかどうかを確認する必要があります。

トラブルシューティング操作を続行する前に、バックアップに関する結果によっては、緊急バックアップを実行することをお勧めします。手順については、次のセクションを参照してください。

定期的なバックアップを保持せずに運用データベースを操作することは、**非常にリスク** です。

7.3.4 緊急バックアップ

一部の緊急の状況では、メインの app データベースの緊急論理バックアップを作成する必要がある場合があります。

次の例は、`cluster-example-1` ポッドから `cluster-example` Postgres クラスターの app データベースの論理バックアップを取得する方法を示しています。

```
kubectl exec cluster-example-1 -c postgres \
  -- pg_dump -Fc -d app > app.dump
```

> **注釈**
>
> 環境で使用したオブジェクトの名前を提供することにより、上記のコマンドをクラスターのバックアップに簡単に適合させることができます。

上記のコマンドは、カスタム形式で `pg_dump` コマンドを発行します。これは、logical backups in PostgreSQL を取得する最も汎用性の高い方法です。

次の手順は、データベースを復元することです。初期化されたばかりの新しい PostgreSQL クラスターで操作していることを前提としていますので、app データベースは空です。

次の例は、プライマリ `new-cluster-example-1` ポッドに接続することにより、`new-cluster-example` Postgres クラスターの app データベースに上記の論理バックアップを復元する方法を示しています。

```
kubectl exec -i new-cluster-example-1 -c postgres \
  -- pg_restore --no-owner --role=app -d app --verbose < app.dump
```

上記の手順は、将来のある段階で cnp プラグインに統合される可能性があります。

7.3.5 ログ

CloudNativePG クラスターログの EDB Postgres AI によって作成および管理されるすべてのリソースは、*JSON 形式の PostgreSQL エラーメッセージの標準出力ログ* を使用して、Kubernetes 規約に従って標準出力に記録されます。

ログは通常インフラレベルで処理され、CloudNativePG クラスターの EDB Postgres AI からのログが含まれますが、トラブルシューティング中にコマンドラインインターフェイスからログに直接アクセスすることが重要です。これには、3 つの主なオプションがあります。

- `kubectl logs` コマンドを使用して特定のリソースからログを取得し、読みやすいように `jq` を適用します。
- CloudNativePG クラスター固有のロギングの EDB Postgres AI の *kubectl cnp logs* を使用します。
- stern のような専用のオープンソースツールを活用し、複数のリソースたとえば、`k8s.enterprisedb.io/clusterName` ラベルを選択して PostgreSQL クラスター内のすべてのポッドからログを集計し、ログエントリをフィルタリング、出力形式をカスタマイズできます。

> **注釈**
>
> 以下のセクションでは、CloudNativePG クラスターの EDB Postgres AI のトラブルシューティングを行うときに、さまざまなリソースのログを取得する方法の例を提供します。

7.3.6 オペレーター情報

デフォルトでは、CloudNativePG クラスターオペレーターの EDB Postgres AI は、Kubernetes の `postgresql-operator-system` 名前空間に `Deployment` としてインストールされます *Details about the deployment* を参照してください

詳細についてはこちらをご覧ください。

次のコマンドを実行して、オペレーターポッドのリストを取得できます。

```
kubectl get pods -n postgresql-operator-system
```

> **注釈**
>
> 通常、オペレーターが実行されているポッドが 1 つあり、 *postgresql-operator-controller-manager-* で始まる名前で識別されます。高可用性用にオペレーターを設定している場合、より多くのエントリがあるはずです。これらのポッドは、 *postgresql-operator-controller-manager* という名前の展開によって管理されます。

次の方法を使用して、ポッド <POD> で実行されているオペレーターに関する関連情報を収集します。

```
kubectl describe pod -n postgresql-operator-system <POD>
```

次に、次のコマンドを実行して同じポッドからログを取得します。

```
kubectl logs -n postgresql-operator-system <POD>
```

オペレーターに関する詳細情報を収集する

次を実行して、CloudNativePG クラスターオペレーター展開マルチオペレーター展開の場合の EDB Postgres AI のすべてのポッドからログを取得します。

```
kubectl logs -n postgresql-operator-system \
  deployment/postgresql-operator-controller-manager --all-containers=true
```

> **Tip**
>
> *-f* フラグを上記のコマンドに追加して、リアルタイムでログを追跡できます。

次のコマンドを実行して、ログを JSON ファイルに保存します。

```
kubectl logs -n postgresql-operator-system \
  deployment/postgresql-operator-controller-manager --all-containers=true | \
  jq -r . > cnp_logs.json
```

kubectl-cnp プラグインを使用して、CloudNativePG クラスターオペレーターバージョンの EDB Postgres AI を取得します。

```
kubectl-cnp status <CLUSTER>
```

出力

```
Cluster in healthy state
Name:               cluster-example
Namespace:          default
System ID:          7044925089871458324
PostgreSQL Image:   docker.enterprisedb.com/k8s/postgresql:18.4-standard-ubi9
Primary instance:   cluster-example-1
Instances:          3
Ready instances:    3
Current Write LSN:  0/5000000 (Timeline: 1 - WAL File: 000000010000000000000004)

Continuous Backup status
Not configured

Streaming Replication status
Name               Sent LSN   Write LSN  Flush LSN  Replay LSN  Write Lag       Flush␣
↪Lag       Replay Lag      State      Sync State  Sync Priority
- ---               --------   ---------  ---------  ----------  ---------       -----
↪----       ----------      -----      ----------  -------------
cluster-example-2  0/5000000  0/5000000  0/5000000  0/5000000   00:00:00       ␣
↪00:00:00        00:00:00        streaming  async       0
cluster-example-3  0/5000000  0/5000000  0/5000000  0/5000000   00:00:00.10033 ␣
```

(次のページに続く)

(前のページからの続き)

```
↪00:00:00.10033  00:00:00.10033  streaming  async       0

Instances status
Name               Database Size  Current LSN  Replication role  Status  QoS        ␣
↪Manager Version
- ---               -------------  -----------  ----------------  ------  ---        ␣
↪---------------
cluster-example-1  33 MB          0/5000000    Primary           OK      BestEffort ␣
↪1.12.0
cluster-example-2  33 MB          0/5000000    Standby (async)   OK      BestEffort ␣
↪1.12.0
cluster-example-3  33 MB          0/5000060    Standby (async)   OK      BestEffort ␣
↪1.12.0
```

7.3.7 クラスター情報

NAMESPACE 名前空間の<CLUSTER> クラスターのステータスは、次の方法で確認できます。

```
kubectl get cluster -n <NAMESPACE> <CLUSTER>
```

出力

```
NAME        AGE        INSTANCES   READY   STATUS                     PRIMARY
<CLUSTER>   10d4h3m    3           3       Cluster in healthy state   <CLUSTER>-1
```

上記の例では、すべてが_ready_状態で、<CLUSTER>-1 がプライマリである 3 つのインスタンスの正常な PostgreSQL クラスターをレポートします。

異常な状態の場合、 Cluster リソースのマニフェストを取得することにより、詳細を検出できます。

```
kubectl get cluster -o yaml -n <NAMESPACE> <CLUSTER>
```

収集するもう 1 つの重要なコマンドは、 cnp プラグインが提供する status コマンドです。

```
kubectl cnp status -n <NAMESPACE> <CLUSTER>
```

> **Tip**
>
> *--verbose* オプションを追加することにより、詳細情報を印刷できます。

EDB PostgreSQL Advanced Server / PostgreSQL コンテナイメージバージョンを取得します。

```
kubectl describe cluster <CLUSTER_NAME> -n <NAMESPACE> | grep "Image Name"
```

出力

```
Image Name:    docker.enterprisedb.com/k8s/postgresql:18.4-standard-ubi9
```

> **注釈**
>
> また、 *kubectl-cnp status -n <NAMESPACE> <CLUSTER_NAME>* を使用して同じ情報を取得できます。

7.3.8 ポッド情報

次の方法を使用して、特定の PostgreSQL クラスターに属するインスタンスのリストを取得できます。

```
#  using labels available from CNP 1.12.0

kubectl get pod -l k8s.enterprisedb.io/cluster=<CLUSTER> -L role -n <NAMESPACE>

#  using legacy labels

kubectl get pod -l postgresql=<CLUSTER> -L role -n <NAMESPACE>
```

出力

```
NAME          READY   STATUS    RESTARTS   AGE       ROLE
<CLUSTER>-1   1/1     Running   0          10d4h5m   primary
<CLUSTER>-2   1/1     Running   0          10d4h4m   replica
<CLUSTER>-3   1/1     Running   0          10d4h4m   replica
```

次を実行して、ポッドに障害が発生しているかどうか、およびどのように障害が発生しているかを確認できます。

```
kubectl get pod -n <NAMESPACE> -o yaml <CLUSTER>-<N>
```

次の使用で、特定の PostgreSQL インスタンスのすべてのログを取得できます。

```
kubectl logs -n <NAMESPACE> <CLUSTER>-<N>
```

検索を PostgreSQL プロセスのみに制限したい場合は、次のコマンドを実行できます。

```
kubectl logs -n <NAMESPACE> <CLUSTER>-<N> | \
  jq select(.logger=="postgres") | .record.message
```

次の例では、タイムスタンプも追加します。

```
kubectl logs -n <NAMESPACE> <CLUSTER>-<N> | \
  jq -r select(.logger=="postgres") | [.ts, .record.message] | @csv
```

タイムスタンプが Unix エポック時間で表示されている場合、使いやすい形式に変換できます。

```
kubectl logs -n <NAMESPACE> <CLUSTER>-<N> | \
  jq -r select(.logger=="postgres") | [(.ts|strflocaltime("%Y-%m-%dT%H:%M:%S %Z")), .
↪record.message] | @csv
```

PostgreSQL ポッドに関する追加情報を収集およびフィルタリングする

クラッシュした特定のポッドからログを確認します。

```
kubectl logs -n <NAMESPACE> --previous <CLUSTER>-<N>
```

特定の PostgreSQL ポッドから FATAL エラーを取得します。

```
kubectl logs -n <NAMESPACE> <CLUSTER>-<N> | \
  jq -r .record | select(.error_severity == "FATAL")
```

出力

```
{
  "log_time": "2021-11-08 14:07:44.520 UTC",
  "user_name": "streaming_replica",
  "process_id": "68",
  "connection_from": "10.244.0.10:60616",
  "session_id": "61892f30.44",
  "session_line_num": "1",
  "command_tag": "startup",
  "session_start_time": "2021-11-08 14:07:44 UTC",
  "virtual_transaction_id": "3/75",
  "transaction_id": "0",
  "error_severity": "FATAL",
  "sql_state_code": "28000",
  "message": "role \"streaming_replica\" does not exist",
  "backend_type": "walsender"
}
```

特定のポッドのログにある PostgreSQL DB エラーメッセージをフィルタリングします。

```
kubectl logs -n <NAMESPACE> <CLUSTER>-<N> | jq -r .err | select(. != null)
```

出力

```
dial unix /controller/run/.s.PGSQL.5432: connect: no such file or directory
```

err ワードと一致するメッセージを特定のポッドから取得します。

```
kubectl logs -n <NAMESPACE> <CLUSTER>-<N> | jq -r .msg | grep "err"
```

出力

```
2021-11-08 14:07:39.610 UTC [15] LOG:  ending log output to stderr
```

特定のポッドから PostgreSQL プロセスからすべてのログを取得します。

```
kubectl logs -n <NAMESPACE> <CLUSTER>-<N> | \
  jq -r . | select(.logger == "postgres") | select(.msg != "record") | .msg
```

出力

```
2021-11-08 14:07:52.591 UTC [16] LOG:  redirecting log output to logging collector␣
↪process
2021-11-08 14:07:52.591 UTC [16] HINT:  Future log output will appear in directory "/
↪controller/log".
2021-11-08 14:07:52.591 UTC [16] LOG:  ending log output to stderr
2021-11-08 14:07:52.591 UTC [16] HINT:  Future log output will go to log destination
↪"csvlog".
```

値を含むフィールドでフィルタリングされた Pod ログを取得し、実行している| で区切って結合します。

```
kubectl logs -n <NAMESPACE> <CLUSTER>-<N> | \
  jq -r [.level, .ts, .logger, .msg] | join(" | ")
```

出力

```
info | 1636380469.5728037 | wal-archive | Backup not configured, skip WAL archiving
info | 1636383566.0664876 | postgres | record
```

7.3.9 バックアップ情報

次の使用で、名前付けクラスターに対して作成されたバックアップをリストできます。

```
kubectl get backup -l k8s.enterprisedb.io/cluster=<CLUSTER>
```

7.3.10 ストレージ情報

次のように、クラスターが使用する StorageClass をダブルチェックして、調査またはトラブルシューティング中により多くのコンテキストを理解すると役立つ場合があります。

```
STORAGECLASS=$(kubectl get pvc <POD> -o jsonpath={.spec.storageClassName})
kubectl get storageclasses $STORAGECLASS -o yaml
```

多くの場合、クラスターはデフォルトの StorageClass を使用して作成されるため、ここではクラスターポッドの 1 つから StorageClass を取得します。

7.3.11 ノード情報

Kubernetes ノードは、最終的に PostgreSQL ポッドが実行される場所です。それらについてできるだけ多くを知ることが戦略的に重要です。

次の使用で、Kubernetes クラスター内のノードのリストを取得できます。

```
#  look at the worker nodes and their status

kubectl get nodes -o wide
```

さらに、特定のクラスターのポッドが実行されているノードのリストを収集できます。

```
kubectl get pod -l k8s.enterprisedb.io/cluster=<CLUSTER> \
  -L role -n <NAMESPACE> -o wide
```

後者は、ポッドがどこに分散されているかを理解するために重要です。*affinity/anti-affinity rules and/or tolerations* を使用している場合に非常に役立ちます。

7.3.12 条件

多くのネイティブ kubernetes オブジェクト like here と同様に、クラスターは `status.conditions` も公開します。これにより、クラスター全体の状態に依存するのではなく、特定のイベントが発生するを「待機」できます。現在利用可能な条件は次のとおりです。

- LastBackupSucceeded
- 継続的アーカイブ
- 準備完了

`LastBackupSucceeded` は、最新のバックアップのステータスを報告しています。 `True` に設定されている場合、最後のバックアップは正しく取得されており、それ以外の場合は `False` に設定されます。

`ContinuousArchiving` は、WAL アーカイブのステータスを報告しています。`True` に設定されている場合、最後の WAL アーカイブプロセスは正常に終了しており、それ以外の場合は `False` に設定されます。

クラスターにユーザーが指定した数のインスタンスがあり、プライマリインスタンスの準備ができている場合、`Ready` は `True` です。この条件は、クラスターが作成されるのを待機するスクリプトで使用できます。

特定の条件を待機する方法

- バックアップ

```
$ kubectl wait --for=condition=LastBackupSucceeded cluster/<CLUSTER-NAME> -n
↪<NAMESPACE>
```

- 継続的アーカイブ

```
$ kubectl wait --for=condition=ContinuousArchiving cluster/<CLUSTER-NAME> -n
↪<NAMESPACE>
```

- Ready クラスターの準備ができているかどうか。

```
$ kubectl wait --for=condition=Ready cluster/<CLUSTER-NAME> -n <NAMESPACE>
```

以下は、失敗条件を含む cluster.status のスニペットです。

```
$ kubectl get cluster/<cluster-name> -o yaml
.
.
.
  status:
    conditions:
    - message: unexpected failure invoking barman-cloud-wal-archive: exit status
        2
      reason: ContinuousArchivingFailing
      status: "False"
      type: ContinuousArchiving

    - message: exit status 2
      reason: LastBackupFailed
      status: "False"
      type: LastBackupSucceeded

    - message: Cluster Is Not Ready
      reason: ClusterIsNotReady
      status: "False"
      type: Ready
```

7.3.13 ネットワーキング

CloudNativePG クラスターの EDB Postgres AI には、基本的なネットワーキングと接続が必要です。詳細については、ネットワーキング セクションをご覧ください。

既存の環境に CloudNativePG クラスターの EDB Postgres AI をインストールする場合、ネットワークポリシーが設定されている場合、またはクラスター専用に作成されたその他のネットワーク構成が存在する可能性があります、オペレーターとクラスターポッドおよび/またはそれらの間の必要な接続に影響を与える可能性があります。ポッドの間。

次のコマンドを使用して、既存のネットワークポリシーを検索できます。

```
kubectl get networkpolicies
```

Kubernetes ネットワーク管理者によって設定された複数のネットワークポリシーが存在する場合があります。

```
$ kubectl get networkpolicies
NAME                    POD-SELECTOR                    AGE
```

(次のページに続く)

(前のページからの続き)

```
allow-prometheus       k8s.enterprisedb.io/cluster=cluster-example   47m
default-deny-ingress   <none>                                        57m
```

7.3.14 PostgreSQL コアダンプ

まれに、PostgreSQL はクラッシュし、PGDATA フォルダーにコアダンプを生成する場合があります。それが発生した場合、通常、それは PostgreSQL のバグであり、ほとんどの場合、既に解決されています。これが、常に PostgreSQL の最新のマイナーバージョンを実行することが重要である理由です。

CloudNativePG クラスターの EDB Postgres AI を使用すると、k8s.enterprisedb.io/coredumpFilter アノテーションを介してコアダンプに含めるものを制御できます。

> **注釈**
>
> ラベルとアノテーション を参照してください。

CloudNativePG クラスターの EDB Postgres AI が提供する標準のアノテーションの詳細については、 をご覧ください。

デフォルトでは、これがほとんどの場合最も安全なアプローチであるため、ダンプから共有メモリセグメントを除外するために、k8s.enterprisedb.io/coredumpFilter は 0x31 に設定されます。

> **注釈**
>
> Core dump filtering settings を参照してください。コアダンプフィルタを制御するビットマスクを設定する方法の詳細については、

あなたはコアダンプの検査に個人的に関わっていないかもしれませんが、Postgres の専門家がそれらを調べることができるようにそれらの提供を求められる場合があります。まず、次のコマンドを使用して、PGDATA ディレクトリにコアダンプがあることを確認します。Postgres インスタンスが実行されている正しいポッドに対して実行してください。

```
kubectl exec -ti POD -c postgres \
  -- find /var/lib/postgresql/data/pgdata -name core.*
```

通常、これは空のセットを返すはずです。例、コアダンプファイルがあるとします。

```
/var/lib/postgresql/data/pgdata/core.14177
```

ディスク上の領域が十分であることを確認したら、次のように kubectl cp を介してマシンでコアダンプを収集できます。

```
kubectl cp POD:/var/lib/postgresql/data/pgdata/core.14177 core.14177
```

これで、ファイルができました。コアダンプを削除して、サーバー上の領域を解放してください。

7.3.15 プロファイリングデータの視覚化と分析

CloudNativePG クラスターの EDB Postgres AI は、pprof と統合して、2 つのレベルでプロファイリングデータを収集および分析します。

- **オペレーターレベル** – `--pprof-server=true` オプションをオペレーター展開に追加することにより有効にします *Operator configuration* を参照してください。
- **Postgres クラスターレベル** – `alpha.k8s.enterprisedb.io/enableInstancePprof` アノテーションを `Cluster` リソース以下に説明を追加することにより有効にします。

`alpha.k8s.enterprisedb.io/enableInstancePprof` アノテーションが`"true"` に設定されている場合、各インスタンスポッドは、インスタンスマネージャーが提供する Go pprof HTTP サーバーを公開します。

- サーバーはポッド内の `0.0.0.0:6060` でリッスンします。
- `pprof 6060/TCP` という名前のコンテナポートがポッド仕様に自動的に追加されます。

アノテーションを削除するか、`"false"` に設定することにより、いつでも pprof を無効にできます。オペレーターは変更を自動的にロールアウトして、pprof ポートとフラグを削除します。

例

次のアノテーションを追加して、クラスターで pprof を有効にします。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
  annotations:
    alpha.k8s.enterprisedb.io/enableInstancePprof: "true"
spec:
  instances: 3
  # ...
```

このアノテーションを変更すると、インスタンスポッド仕様が更新されポート `6060` および対応するフラグが追加され、ローリング更新がトリガーされます。

> **⚠ 警告**
>
> 次の例では、ローカルテストのみに`kubectl port-forward` を使用します。これは、運用環境で機能を公開するための意図された方法ではありません**。 pprof を機密デバッグインターフェイスとして扱い、決して公開しないでください。リモートでアクセスする必要がある場合は、適切なネットワークポリシーとアクセス制御で保護してください。

ポート転送を使用して pprof エンドポイントにアクセスします。

```
kubectl port-forward -n <namespace> pod/<instance-pod> 6060
curl -sS http://localhost:6060/debug/pprof/
go tool pprof http://localhost:6060/debug/pprof/profile?seconds=30
```

ブラウザーを使用して pprof にアクセスすることもできます。http://localhost:6060/debug/pprof/。

障害対応

最初に、クラスターに `alpha.k8s.enterprisedb.io/enableInstancePprof: "true"` アノテーションセットがあることを確認します。

次に、instance manager コマンドに`--pprof-server` フラグが含まれ、ポート `6060/TCP` が公開されていることを確認します。これは、次を実行して行うことができます。

```
kubectl -n <namespace> describe pod <instance-pod>
```

次に、出力の **Command** および **Ports** セクションを確認します。

最後に、ポート転送を使用していない場合は、ネットワークポリシーがポート `6060/TCP` へのアクセスを許可していることを確認します。

7.3.16 いくつかの既知の問題

ストレージがいっぱいです

ストレージがいっぱいの場合、PostgreSQL ポッドは新しいデータを書き込むことができません。または、WAL セグメントを含むディスクがいっぱいである場合、PostgreSQL はシャットダウンします。

ディスクがいっぱいであることに関するメッセージがログにある場合は、影響を受ける PVC のサイズを増やす必要があります。これは、PVC を編集し、`spec.resources.requests.storage` フィールドを変更することにより行うことができます。その後、新しいサイズでクラスターリソースを更新して、同じ変更をすべてのポッドに適用する必要があります。ドキュメントの **ボリュームの拡張** を参照してください。

WAL セグメントの領域が使い果たされると、ポッドはクラッシュループし、クラスターステータスは `Not enough disk space` と報告されます。PVC、次にクラスターリソースのサイズを増やすと、問題が解決されます。**ディスクフル障害** も参照してください。

ポッドが `Pending` 状態でスタックしている

クラスターのインスタンスが `Pending` フェーズでスタックしている場合、ポッドの `Events` セクションを確認して、この背後にある理由を理解する必要があります。

```
kubectl describe pod -n <NAMESPACE> <POD>
```

この考えられる原因の一部は次のとおりです。

- `nodeSelector` と一致するノードはありません
- Toleration がノードのテイントと一致するように正しく構成されていません
- 利用可能なノードがまったくありません。これは、`cluster-autoscaler` がいくつかの制限に到達したこと、または一時的な問題が発生したことに関連している可能性があります。

この場合、名前空間内のイベントをチェックすることも役立つ場合があります。

```
kubectl get events -n <NAMESPACE>

#  list events in chronological order

kubectl get events -n <NAMESPACE> --sort-by=.metadata.creationTimestamp
```

バックアップが構成されていない場合、レプリカが同期していない

場合によっては、メンテナンスのため、レプリカが少しの間オフになる場合があります Kubernetes ノードがドレインされたときを考えてください。クラスターにバックアップが構成されていない場合、レプリカがバックアップしたときに、プライマリにはもう存在しない WAL ファイル:ref:``postgresql` セクション <*postgresql* セクション>` で述べているように、WAL 管理ポリシーに従って既にリサイクルされており、フォールアウトする場合があります同期のこと。

同様に、`pg_rewind` が以前のプライマリに存在しない WAL ファイルを必要とする場合、`pg_rewind: error: could not open file` をレポートします。

これらの場合、ポッドの準備ができなくなるため、PVC を削除し、オペレーターにレプリカを再構築してもらう必要があります。

動的にプロビジョニングされた永続ボリュームに依存し、PV 自分自身を削除することに自信がある場合は、次の方法で削除できます。

```
PODNAME=<POD>
VOLNAME=$(kubectl get pv -o json | \
  jq -r .items[]|select(.spec.claimRef.name==\"$PODNAME\")|.metadata.name)

kubectl delete pod/$PODNAME pvc/$PODNAME pvc/$PODNAME-wal pv/$VOLNAME
```

クラスターが `Creating new replica` でスタックしている

クラスターは「新しいレプリカの作成」でスタックしていますが、ポッドログには関連する問題は表示されません。これは、次の問題**インストールされたネットワークポリシーにより、ネットワーキングが障害されます**に関連していることがわかりました。ネットワークの問題は、次のようにステータス列に反映されます。

```
Instance Status Extraction Error: HTTP communication issue
```

インストールされたネットワークポリシーにより、ネットワーキングが障害されます

ネットワーキング で指摘されたように、ローカルネットワークポリシーは、必要な接続の一部を妨げる可能性があります。

接続が障害されていることを示す兆候は、オペレーターログに次のようなメッセージが存在することです。

```
"Cannot extract Pod status", [···snipped···] "Get \"http://<pod IP>:8000/pg/status\":␣
↪dial tcp <pod IP>:8000: i/o timeout"
```

ネットワークポリシーをリストし、接続を制限するポリシーを探す必要があります。

```
$ kubectl get networkpolicies
NAME                   POD-SELECTOR                          AGE
allow-prometheus       k8s.enterprisedb.io/cluster=cluster-example   47m
default-deny-ingress   <none>                                57m
```

たとえば、上記のリストでは、 `default-deny-ingress` が犯人である可能性が高いと思われます。それをドリルダウンできます。

```
$ kubectl get networkpolicies default-deny-ingress -o yaml
<…snipped…>
spec:
  podSelector: {}
  policyTypes:
  - Ingress
```

networking page には、オペレーターがクロスネームスペースをクラスターポッドに接続できるようにする `NetworkPolicy` を作成するようにカスタマイズできるネットワークポリシーファイルがあります。

データディレクトリのブートストラップ中のエラー

クラスターの初期化ジョブが「バスエラーコアダンプされた子プロセスが終了コード 135 で終了しました」でクラッシュした場合、クラスター hugepages 設定を修正する必要がある可能性があります。

その理由は、v2 で修正する必要がある cgroup v1 の huge ページのサポートが不完全であるためです。詳細については、PostgreSQL BUG #17757: Nothonoring huge_pages setting during initdb causes DB crash inKubernetes を確認してください。

huge ページが有効になっているかどうかを確認するには、Kubernetes ノードで `grep HugePages /proc/meminfo` を実行し、hugepage が存在するかどうか、そのサイズ、および空き数を確認します。

huge ページが存在する場合、すべての PostgreSQL ポッドで使用可能な hugepage メモリの量を構成する必要があります。

例

```
postgresql:
  parameters:
    shared_buffers: "128MB"

resources:
  requests:
    memory: "512Mi"
  limits:
    hugepages-2Mi: "512Mi"
```

クラスター内のすべてのポッドをスケジュールするには、十分な hugepages メモリが必要であることに注意してください。上記の例では、ポッドごとに少なくとも 512MiB を解放する必要があります。

ブートストラップジョブが実行中ステータスでハングする

Running ステータスにあるときにクラスターの初期化ジョブがハングし、メッセージ「API サーバーが到達可能になるのを待機しているときにエラーが発生した場合」は、おそらくネットワークに問題があり、Kubernetes API サーバーとの通信を妨げています。初期化ジョブほとんどのジョブと同様に、Kubernetes API にアクセスする必要があります。ネットワークを確認してください。

もう 1 つの考えられる原因は、サイドカーインジェクションを構成している場合です。Istio などのサイドカーは、起動中にネットワークを一時的に利用できなくなる場合があります。サイドカーインジェクションを有効にしている場合は、インジェクションを無効にして再試行します。

レプリカは、フェールオーバー後の再接続に 2 分以上かかります

プライマリインスタンスに障害が発生すると、オペレーターは最も高度なスタンバイをプライマリロールに昇格させます。他のスタンバイインスタンスは、レプリケーションのために-rw サービスに再接続しようとします。ただし、この再接続プロセス中に、kube-proxy はルーティング情報をまだ更新していない場合があります。その結果、スタンバイインスタンスによって送信された最初の SYN パケットは、目的の宛先に到達しない場合があります。

ネットワークがパケットを拒否するのではなく、サイレントにドロップするように構成されている場合、スタンバイインスタンスは応答を受信せず、指数バックオフ期間の後に接続を再試行します。Linux システムでは、 tcp_syn_retries カーネルパラメーターのデフォルト値は 6 です。これは、システムが断念するまでに約 127 秒間の接続の確立を試行することを意味します。この再試行期間が長くなると、再接続プロセスが大幅に遅延する可能性があります。詳細については、

tcp_syn_retries documentation を参照してください。

Operator configuration で STANDBY_TCP_USER_TIMEOUT を設定することにより、この問題をワークアラウンドできます。これにより、最初の SYN パケットが指定されたタイムアウト内に確認されない場合、スタンバイインスタンスは TCP 接続を閉じ、より迅速に接続を再試行できます。

7.4 API Reference - v1.30.0

7.4.1 Packages

- *postgresql.k8s.enterprisedb.io/v1*

7.4.2 postgresql.k8s.enterprisedb.io/v1

パッケージ v1 には、postgresql v1 API グループの API スキーマ定義が含まれています

Resource Types

- *Appendix B - Backup on object stores*
- *cluster-example-projected-volume.yaml*
- *ClusterImageCatalog*
- *Postgres* データベースの宣言的管理

- :ref:``DatabaseRole` リソースを使用した宣言的ロール管理 <*DatabaseRole* リソースを使用した宣言的ロール管理>`
- *DatabaseRoleList*
- *FailoverQuorum*
- *ClusterImageCatalog*
- **プーラー構成**
- **宣言的なパブリケーションおよびサブスクリプションオブジェクトを介した論理レプリケーション**
- *ScheduledBackup*
- **宣言的なパブリケーションおよびサブスクリプションオブジェクトを介した論理レプリケーション**

AffinityConfiguration

AffinityConfiguration には、ポッドのアフィニティルールを作成するために必要な情報が含まれています

登場場所:

- *ClusterSpec*

Field	Description	Required	Default	Validation
enablePodAntiAffinity _boolean_	ポッドのアンチアフィニティをアクティブにします。このフィールドが明示的に false に設定されていない限り、オペレーターはポッド アンチアフィニティを定義します			
topologyKey _string_	アンチアフィニティ構成に使用するトポロジキー。詳細については、k8s ドキュメント を参照してください。			
nodeSelector _object (keys:string, values:string)_	NodeSelector は、ポッドを実行できるノードを定義するために使用されるキーと値のペアのマップです。 詳細情報 <https://kubernetes.io/docs/concepts/configuration/assign-pod- node/>			
nodeAffinity _ NodeAffinity _	NodeAffinity は、ポッドのノードアフィニティスケジューリングルールについて説明します。 詳細情報 <https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#node-affinity>			

次のページに続く

表 1 – 前のページからの続き

Field	Description	Required	Default	Validation
tolerations _ 許容範囲 **array**_	Tolerations は、汚染されたノードで実行できるようにするために、すべてのポッドに設定する必要がある許容範囲のリストです。 詳細情報 <https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/>			
podAntiAffinityType _string_	PodAntiAffinityType を使用すると、ユーザーはクラスターインスタンス間のポッドアンチアフィニティを スケジュール中に強い要件と見なすかどうかを決定できます。使用できる値は「preferred」空の場合のデフォルトまたは 「required」です。 「required」に設定すると、新しい kubernetes ノードが 追加されるまでインスタンスが保留されたままになる可能性があります。既存のすべてのノードが必要なポッドのアンチアフィニティルールと一致しない場合。 詳細情報 <https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#inter-pod-affinity-and-anti-affinity>			
additionalPodAntiAffinity _ PodAntiAffinity _	AdditionalPodAntiAffinity では、EnablePodAntiAffinity が true デフォルトに設定されている場合にオペレーターによって生成された用語に追加するポッドアンチアフィニティ用語を指定できます、または false に設定されている場合に排他的に使用されます。			
additionalPodAffinity _ PodAffinity _	AdditionalPodAffinity では、すべてのクラスターのポッドに渡されるポッドアフィニティ用語を指定できます。			

AvailableArchitecture

AvailableArchitecture は、クラスターのアーキテクチャの状態を表します

登場場所:

- *ClusterStatus*

Field	Description	Required	Default	Validation
goArch _string_	GoArch は実行可能アーキテクチャの名前です	True		
hash _string_	ハッシュは実行可能ファイルのハッシュです	True		

Backup

バックアップリソースは、ユーザーによる PostgreSQL バックアップの要求です。

Field	Description	Required	Default	Validation
apiVersion _string_	*postgresql.k8s.enterprisedb.io/v1*	True		
kind _string_	*Backup*	True		
metadata _ *EmbeddedObjectMetadata* _	`metadata`のフィールドについては、Kubernetes API ドキュメントを参照してください。	True		
spec _ *BackupSpec* _	バックアップの目的の動作の仕様。 詳細情報 <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>	True		
status _ *BackupStatus* _	最近観察されたバックアップのステータス。このデータは最新でない可能性があります 。システムによって設定されます。読み取り専用。 詳細情報 <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>			

BackupConfiguration

BackupConfiguration は、クラスターのバックアップを取得する方法を定義します。サポートされているバックアップ方法は、BarmanObjectStore と VolumeSnapshot です。詳細と例については、ドキュメントのバックアップとリカバリーセクションを参照してください。

登場場所:

- *ClusterSpec*

Field	Description	Required	Default	Validation
volumeSnapshot _ *VolumeSnapshot-Configuration* _	VolumeSnapshot は、ボリュームスナップショットバックアップを実行するための構成を提供します。			
barmanObjectStore _ BarmanObject-StoreConfiguration _	barman-cloud ツールスイートの構成			
retentionPolicy _string_	RetentionPolicy は、バックアップ および WAL つまり '60d' に使用される保持ポリシーです。保持ポリシーは`XXu`のフォームで表現されます。ここで、*XX`は正の整数、`u`は `[dwm]* - 日、週、月 現在、BarmanObjectStore メソッドを使用する場合にのみ適用されます。			Pattern: *^[1-9][0-9]*[dwm]$*
target _ *BackupTarget* _	バックアップを実行するインスタンスを決定するポリシー。使用可能な オプションは空の文字列で、デフォルトで`prefer-standby`ポリシー、 プライマリインスタンスでバックアップを常に実行するように `primary`、 `prefer-standby` 利用可能な場合は、できれば最新のスタンバイでバックアップを実行します。			Enum: [primary prefer-standby]

BackupMethod

Underlying type: string

BackupMethod は、選択した PostgreSQL インスタンスの物理ベースのバックアップを実行する方法を定義します

登場場所:

- *BackupSpec*
- *BackupStatus*
- *ClusterStatus*

- *ScheduledBackupSpec*

Field	Description
volumeSnapshot	BackupMethod VolumeSnapshot とは、ボリュームスナップショットを使用することを意味します Kubernetes 機能
barmanObjectStore	BackupMethodBarmanObjectStore は、barman を使用してバックアップすることを意味します PostgreSQL クラスター
plugin	BackupMethodPlugin は、このバックアップを プラグイン によって処理する必要があることを意味します

BackupPhase

Underlying type: string

BackupPhase はバックアップのフェーズです

登場場所:

- *BackupStatus*

BackupPluginConfiguration

BackupPluginConfiguration には、バックアッププラグインが使用するバックアップ構成が含まれます

登場場所:

- *BackupSpec*
- *ScheduledBackupSpec*

Field	Description	Required	Default	Validation
name _string_	Name は、このバックアップを管理するプラグインの名前です	True		
parameters _object (keys:string, values:string)_	パラメーターは、このバックアップのバックアップ プラグインに渡される構成パラメーターです。			

BackupSnapshotElementStatus

BackupSnapshotElementStatus は、ボリュームスナップショット方式バックアップの一部であるボリュームスナップショットです

登場場所:

- *BackupSnapshotStatus*

Field	Description	Required	Default	Validation
name _string_	Name はスナップショットリソース名です	True		
type _string_	タイプは、クラスター内のスナップショットのロールです。PG_DATA、PG_WAL、PG_TABLESPACE など	True		
tablespaceName _string_	TablespaceName は、スナップショットされたテーブルスペースの名前です。タイプが PG_TABLESPACE の場合にのみ設定されます			

BackupSnapshotStatus

BackupSnapshotStatus volumeSnapshot 方法のバックアップ専用のフィールド

登場場所:

- *BackupStatus*

Field	Description	Required	Default	Validation
elements _ *BackupSnapshotElementStatus* **array**_	収集されたボリュームスナップショットが設定された要素リスト			

BackupSource

BackupSource には、復元する必要があるバックアップと、それを正しく復元するために必要な情報が含まれています。

登場場所:

- *BootstrapRecovery*

Field	Description	Required	Default	Validation
name _string_	参照先の名前。	True		
endpointCA _ SecretKeySelector _	EndpointCA は、barman エンドポイントの CA バンドルを保存します。自己署名証明書を使用して 証明書発行者と barman-cloud-wal-archive での エラーを回避する場合に役立ちます。			

BackupSpec

BackupSpec は、バックアップの望ましい状態を定義します

登場場所:

- *Appendix B - Backup on object stores*

Field	Description	Required	Default	Validation
cluster _ LocalObjectReference _	バックアップするクラスター	True		
target _ *BackupTarget* _	このバックアップを実行するインスタンスを決定するポリシー。空の場合 デフォルトの `cluster.spec.backup.target`. 使用可能なオプションは空の文字列、`primary`および `prefer-standby`です。 `primary`は常にプライマリインスタンスで実行されます。 `prefer-standby`はバックアップを優先して実行します最新の スタンバイで利用可能な場合。			Enum: [primary prefer-standby]
method _ *BackupMethod* _	使用するバックアップ方法、可能なオプションは `barmanObjectStore`、 `volumeSnapshot`または `plugin`です。デフォルト `barmanObjectStore`		barmanObjectStore	Enum: [barmanObjectStore volumeSnapshot plugin]
pluginConfiguration _ *BackupPluginConfiguration* _	このバックアップを管理するプラグインに渡される構成パラメーター			
online _boolean_	ボリュームスナップショットを使用したバックアップのデフォルトのタイプが オンライン/ホット *true*、デフォルトかオフライン/コールド `false`かどうか クラスターフィールド '.spec.backup.volumeSnapshot で指定されたデフォルト設定をオーバーライドします。オンライン』			

次のページに続く

表 10 – 前のページからの続き

Field	Description	Required	Default	Validation
onlineConfiguration _ *OnlineConfiguration* _	ボリュームスナップショットを使用してオンライン/ホットバックアップを制御する構成パラメーター クラスター'.backup.volumeSnapshot.onlineConスタンザで指定されたデフォルト設定をオーバーライドします			

BackupStatus

BackupStatus は、バックアップの監視された状態を定義します

登場場所:

- *Appendix B - Backup on object stores*

Field	Description	Required	Default	Validation
googleCredentials _ GoogleCredentials _	Google Cloud Storage にデータをアップロードするために使用する資格情報			
s3Credentials _ S3Credentials _	S3 にデータをアップロードするために使用する資格情報			
azureCredentials _ AzureCredentials _	Azure Blob Storage にデータをアップロードするために使用する資格情報			
majorVersion _integer_	バックアップが作成されたときに実行されていた PostgreSQL メジャーバージョン バックアップ。	True		
endpointCA _ SecretKeySelector _	EndpointCA は、barman エンドポイントの CA バンドルを保存します。自己署名証明書を使用して 証明書発行者と barman-cloud-wal-archive での エラーを回避する場合に役立ちます。			
endpointURL _string_	クラウドにデータをアップロードするために使用されるエンドポイント 自動エンドポイント検出をオーバーライド			

次のページに続く

表 11 – 前のページからの続き

Field	Description	Required	Default	Validation
destinationPath _string_	バックアップを保存するパスつまり s3://bucket/path/to/folder このパスは、さまざまな宛先フォルダーで、WAL およびデータに使用されます。これは、エラーが発生した場合に入力されない場合があります。			
serverName _string_	この パラメータが省略された場合、S3 のサーバー名、クラスター名が使用されます			
encryption _string_	S3 API に必要な暗号化方法			
backupId _string_	Barman バックアップの ID			
backupName _string_	Barman のバックアップの名前			
phase _ *BackupPhase* _	前回のバックアップステータス			
startedAt _ バックアップからのポイントインタイム リカバリー *PITR* _	バックアップツールによってバックアップの実行が開始されたとき			
stoppedAt _ バックアップからのポイントインタイム リカバリー *PITR* _	バックアップツールによってバックアップの実行が終了されたとき			
reconciliationStartedAt _ バックアップからのポイントインタイム リカバリー *PITR* _	オペレーターによってバックアッププロセスが開始されたとき			
reconciliationTerminatedAt _ バックアップからのポイントインタイム リカバリー *PITR* _	調整がオペレーターによって終了されたとき成功または失敗			
beginWal _string_	開始する WAL			
endWal _string_	エンディング WAL			
beginLSN _string_	スタート xlog			
endLSN _string_	エンディング xlog			
error _string_	検出されたエラー			
commandOutput _string_	未使用。古いバージョンとの互換性のために保持されています。			
commandError _string_	エラーが発生した場合の backup コマンド出力			

次のページに続く

表 11 – 前のページからの続き

Field	Description	Required	Default	Validation
backupLabelFile _integer **array**_	オンラインホットバックアップの場合に Postgres から返されたラベルファイルのコンテンツをバックアップします			
tablespaceMapFile _integer **array**_	オンラインホットバックアップの場合に Postgres によって返されるテーブルスペースマップファイルのコンテンツ			
instanceID _ *InstanceID* _	バックアップが取得されたインスタンスを識別する情報			
snapshotBackupStatus _ *BackupSnapshotStatus* _	ボリュームのステータススナップショットバックアップ			
method _ *BackupMethod* _	使用されているバックアップ方法			
online _boolean_	バックアップがオンライン/ホット *true* またはオフライン/コールド *false*			
pluginMetadata _object (keys:string, values:string)_	プラグインのメタデータを含むマップ			

BackupTarget

Underlying type: string

BackupTarget は、バックアップの優先ターゲットを説明します

登場場所:

- *BackupConfiguration*
- *BackupSpec*
- *ScheduledBackupSpec*

BootstrapConfiguration

BootstrapConfiguration には、PostgreSQL クラスターの作成方法に関する情報が含まれています。サポートされているブートストラップ方法のうち、1 つのブートストラップ方法のみを定義できます。`initdb` は、指定しない場合、ブートストラップ方法として使用されます。詳細については、ドキュメントの Bootstrap ページを参照してください。

登場場所:

- *ClusterSpec*

Field	Description	Required	Default	Validation
initdb _ *BootstrapInitDB* _	initdb を介してクラスターをブートストラップする			
recovery _ *BootstrapRecovery* _	バックアップからクラスターをブートストラップする			
pg_basebackup _ *BootstrapPgBaseBackup* _	別の互換性のある PostgreSQL インスタンスの物理バックアップを取得してクラスターをブートストラップする			

BootstrapInitDB

BootstrapInitDB は、initdb が使用される場合のブートストラッププロセスの構成です。詳細については、ドキュメントの Bootstrap ページを参照してください。

登場場所:

- *BootstrapConfiguration*

Field	Description	Required	Default	Validation
database _string_	アプリケーションが使用するデータベースの名前。デフォルト`app`			
owner _string_	アプリケーションが使用するインスタンスのデータベースの所有者の名前 。`database`キーの値のデフォルト。			
secret _ LocalObjectReference _	ユーザーデータベースの 所有者の初期資格情報を含むシークレットの名前。空の場合、新しいシークレットが 最初から作成されます			
redwood _boolean_	Redwood の互換性を有効/無効にする必要がある場合。必要 EPAS および EPAS の場合 デフォルトで true			

次のページに続く

表 13 – 前のページからの続き

Field	Description	Required	Default	Validation
options _string **array**_	クラスターの作成時に initdb に渡す必要があるオプションのリスト。 非推奨これは一貫性のない構成につながる可能性があります。 代わりに明示的に提供されたパラメーターを使用してください。 定義されている場合、明示的な値は無視されます。			
dataChecksums _boolean_	*-k* `オプションを *initdb* に渡し、* * データページのチェックサムを有効にするかどうかデフォルト *false*			
encoding _string_	initdb のオプション `--encoding`として渡される値デフォルト`UTF8`			
localeCollate _string_	initdb のオプション `--lc-collate`として渡される値デフォルト`C`			
localeCType _string_	initdb のオプション `--lc-ctype`として渡される値デフォルト`C`			
locale _string_	新しいデータベースにデフォルトの照合順序と文字分類を設定します。			
localeProvider _string_	このオプションは、新しいクラスターで作成されるデータベースのロケールプロバイダーを設定します。 PostgreSQL 16 から利用できます。			
icuLocale _string_	ICU プロバイダーを使用するときに ICU ロケールを指定します。 このオプションには、`localeProvider`を `icu`に設定する必要があります。 PostgreSQL 15 から利用できます。			
icuRules _string_	追加の照合ルールを指定して、デフォルトの照合の動作をカスタマイズします。 このオプションでは、 `localeProvider`を`icu`に設定する必要があります。 PostgreSQL 16 から利用できます。			

次のページに続く

表 13 – 前のページからの続き

Field	Description	Required	Default	Validation
builtinLocale _string_	ビルトインプロバイダーを使用する場合にロケール名を指定します。 このオプションには、`localeProvider`を`builtin`に設定する必要があります。 PostgreSQL 17 から利用できます。			
walSegmentSize _integer_	initdb の `--wal-segsize` オプションに渡されるメガバイト単位の値 1〜1024 デフォルト空、PostgreSQL のデフォルト 16MB になります			Maximum: 1024 Minimum: 1
postInitSQL _string **array**_	クラスターが作成された直後に`postgres` データベースでスーパーユーザーとして実行される SQL クエリーのリスト - 細心の注意を払って使用します デフォルトでは空			
postInitApplicationSQL _string **array**_	クラスターが作成された直後にアプリケーション データベースでスーパーユーザーとして実行される SQL クエリのリスト - 細心の注意を払って使用します デフォルトでは空			
postInitTemplateSQL _string **array**_	クラスターが作成された直後に`template1` データベースでスーパーユーザーとして実行される SQL クエリーのリスト - 細心の注意を払って使用します デフォルトでは空			
import _ *OpenShift の重要な概念* _	論理バックアップ `pg_dump`および`pg_restore`を使用して、既存の PostgreSQL インスタンスからデータをインポートすることにより、新しいクラスターをブートストラップします			

次のページに続く

表 13 – 前のページからの続き

Field	Description	Required	Default	Validation
postInitApplicationSQLRefs _ *SQLRefs* _	SQL ファイルを含む ConfigMap またはシークレットへの参照のリスト クラスターが作成された直後にアプリケーションデータベースでスーパーユーザーとして実行されます クラスタが作成された後。参照は特定の順序で処理されます。 最初に、すべてのシークレットが処理され、次にすべての ConfigMap が処理されます。 各グループ内で、処理順序はそれぞれの配列で指定されたシーケンスに従います 。 (デフォルトでは空)			
postInitTemplateSQLRefs _ *SQLRefs* _	クラスターが作成された直後に`template1`データベースでスーパーユーザーとして実行される SQL ファイルを含む ConfigMaps またはシークレットへの参照 リスト。参照は特定の順序で処理されます。 最初に、すべてのシークレットが処理され、次にすべての ConfigMap が処理されます。 各グループ内で、処理順序はそれぞれの配列で指定されたシーケンスに従います 。 (デフォルトでは空)			
postInitSQLRefs _ *SQLRefs* _	クラスターが作成された直後に`postgres`データベースでスーパーユーザーとして実行される SQL ファイルを含む ConfigMaps またはシークレットへの参照 リスト。参照は特定の順序で処理されます。 最初に、すべてのシークレットが処理され、次にすべての ConfigMap が処理されます。 各グループ内で、処理順序はそれぞれの配列で指定されたシーケンスに従います 。 (デフォルトでは空)			

BootstrapPgBaseBackup

BootstrapPgBaseBackup には、既存の PostgreSQL クラスターの物理バックアップを取得するために必要な構成が含まれています

登場場所:

- *BootstrapConfiguration*

Field	Description	Required	Default	Validation
source _string_	物理的バックアップを取得する必要があるサーバーの名前	True		MinLength: 1
database _string_	アプリケーションが使用するデータベースの名前。デフォルト `app`			
owner _string_	アプリケーションが使用するインスタンスのデータベースの所有者の名前 。`database`キーの値のデフォルト。			
secret _ LocalObjectReference _	ユーザーデータベースの 所有者の初期資格情報を含むシークレットの名前。空の場合、新しいシークレットが 最初から作成されます			

BootstrapRecovery

BootstrapRecovery には、外部クラスター、ボリュームスナップショット、またはバックアップオブジェクトの 3 つの方法論を使用して既存のクラスターから復元するために必要な構成が含まれています。フルリカバリーとポイントインタイムリカバリーがサポートされています。この方法は、継続的リカバリーのクラスターレプリカクラスターを作成するために使用することもでき、次の場合にカスケードレプリケーションをサポートします `instances` >

1. クラスターがリカバリーを終了すると、スーパーユーザーのパスワードは、提供されたシークレットを介して変更されます。詳細については、ドキュメントの Bootstrap ページを参照してください。

登場場所:

- *BootstrapConfiguration*

Field	Description	Required	Default	Validation
backup _ *BackupSource* _	リカバリ手順を開始する物理ベースバックアップを含むバックアップオブジェクト。 `source`および `volumeSnapshots`と相互排他的。			
source _string_	バックアップを復元する外部クラスター。これは バックアップが保存されるフォルダーの名前としても使用される ため、ソースクラスターの名前に設定する必要があります `backup`と相互排他的。			
volumeSnapshots _ *DataSource* _	リカバリ手順を開始する静的 PVC データソース リカバリ手順。現在サポートしている `VolumeSnapshot` および `PersistentVolumeClaim`リソースは、既存の PVC グループをマッピングし、CloudNativePG クラスターの EDB Postgres AI と互換性があり、フェンスされた Postgres インスタンスのコールドバックアップコピーで取得されます制限 将来的にオンラインバックアップが 実装されると削除されます)。 `backup`と相互排他的。			

次のページに続く

表 15 – 前のページからの続き

Field	Description	Required	Default	Validation
recoveryTarget _ *RecoveryTarget* _	デフォルトでは、復旧プロセスはアーカイブ内の使用可能なすべての WAL ファイルを適用しますフルリカバリー。ただし、 PostgreSQL で想定されているように、 一貫した状態に到達するとすぐにリカバリーを終了するか、`RecoveryTarget`オブジェクトを指定してポイントインタイム PITR にリカバリーすることもできます。つまり、タイムスタンプ、トランザクション ID、LSN など。 詳細情報<https://www.postgresql.org/docs/current/runtime-config-wal.html#RUNTIME-CONFIG-WAL-RECOVERY-TARGET>			
database _string_	アプリケーションが使用するデータベースの名前。デフォルト`app`			
owner _string_	アプリケーションが使用するインスタンスのデータベースの所有者の名前 。`database`キーの値のデフォルト。			
secret _ LocalObjectReference _	ユーザーデータベースの 所有者の初期資格情報を含むシークレットの名前。空の場合、新しいシークレットが 最初から作成されます			

CatalogComponentImage

CatalogComponentImage は、非 PostgreSQL コンポーネントの名前付け画像エントリです。

登場場所:

- *ImageCatalogSpec*

Field	Description	Required	Default	Validation
key _string_	Key は、カタログ内のこの画像の一意の識別子です。	True		MaxLength: 63 Pattern: ^ *:ref: `a-z0-9 <[-a-z0-9]*[a-z0-9]> ?$`
image _string_	Image はコンテナイメージ参照です。	True		

CatalogImage

CatalogImage はイメージとメジャーバージョンを定義します

登場場所:

- *ImageCatalogSpec*

Field	Description	Required	Default	Validation
image _string_	画像参照	True		
major _integer_	イメージの PostgreSQL メジャーバージョン。カタログ内で一意である必要があります。	True		Minimum: 10
extensions _ *ExtensionConfiguration* **array**_	追加する拡張機能の構成			

CertificatesConfiguration

CertificatesConfiguration には、サーバー証明書を処理するために必要な構成が含まれています。

登場場所:

- *CertificatesStatus*
- *ClusterSpec*

Field	Description	Required	Default	Validation
serverCASecret _string_	Server CA 証明書を含むシークレット。定義されていない場合、新しいシークレットが自己署名 CA で作成され、TLS 証明書 ServerTLSSecret の生成に使用されます。 含まれるもの - *ca.crt* サーバー証明書の検証に使用する CA。 クライアント接続文字列で `sslrootcert`として使用します。 - *ca.key* サーバー SSL 証明書の生成に使用されるキー ServerTLSSecret が提供されている場合、 これ省略できます。 			
serverTLSSecret _string_	サーバー TLS 証明書とキーを含む kubernetes.io/tls タイプのシークレット。クライアントが postgres に安全に接続できるように `ssl_cert_file`および `ssl_key_file`として設定されるキー。 定義されていない場合、ServerCASecret は`ca.key`も提供する必要があります。そして新しいシークレットが 提供された CA を使用して作成されます。			
replicationTLSSecret _string_	`streaming_replica`ユーザーとして認証するためのクライアント証明書を含む kubernetes.io/tls タイプのシークレット。定義されていない場合、ClientCASecret は`ca.key`も提供する必要があり、新しいシークレットが 次を使用して作成されます。提供された CA。			

次のページに続く

表 18 – 前のページからの続き

Field	Description	Required	Default	Validation
clientCASecret _string_	クライアント CA 証明書を含むシークレット。定義されていない場合、新しいシークレットが自己署名 CA で作成され、すべてのクライアント証明書の生成に使用されます。 含まれるもの - *ca.crt* クライアント証明書の検証に使用する CA。 すべてのインスタンスの`ssl_ca_file`として使用します。 - *ca.key* クライアント証明書の生成に使用されるキー。ReplicationTLSSecret が提供されている場合、これはできます。省略 します。			
serverAltDNSNames _string **array**_	必要に応じて、生成されたサーバー TLS 証明書に追加されるサーバー代替 DNS 名のリスト。			

CertificatesStatus

CertificatesStatus には、構成証明書と関連する有効期限が含まれています。

登場場所:

- *ClusterStatus*

Field	Description	Required	Default	Validation
serverCASecret _string_	Server CA 証明書を含むシークレット。定義されていない場合、新しいシークレットが自己署名 CA で作成され、TLS 証明書 ServerTLSSecret の生成に使用されます。 含まれるもの - *ca.crt* サーバー証明書の検証に使用する CA。 クライアント接続文字列で `sslrootcert`として使用します。 - *ca.key* サーバー SSL 証明書の生成に使用されるキー ServerTLSSecret が提供されている場合、 これ省略できます。 			
serverTLSSecret _string_	サーバー TLS 証明書とキーを含む kubernetes.io/tls タイプのシークレット。クライアントが postgres に安全に接続できるように `ssl_cert_file`および `ssl_key_file`として設定されるキー。 定義されていない場合、ServerCASecret は`ca.key`も提供する必要があります。そして新しいシークレットが 提供された CA を使用して作成されます。			
replicationTLSSecret _string_	`streaming_replica`ユーザーとして認証するためのクライアント証明書を含む kubernetes.io/tls タイプのシークレット。定義されていない場合、ClientCASecret は`ca.key`も提供する必要があり、新しいシークレットが 次を使用して作成されます。提供された CA。			

次のページに続く

表 19 – 前のページからの続き

Field	Description	Required	Default	Validation
clientCASecret _string_	クライアント CA 証明書を含むシークレット。定義されていない場合、新しいシークレットが自己署名 CA で作成され、すべてのクライアント証明書の生成に使用されます。 含まれるもの - *ca.crt* クライアント証明書の検証に使用する CA。 すべてのインスタンスの`ssl_ca_file`として使用します。 - *ca.key* クライアント証明書の生成に使用されるキー。ReplicationTLSSecret が提供されている場合、これはできます。省略 します。			
serverAltDNSNames _string **array**_	必要に応じて、生成されたサーバー TLS 証明書に追加されるサーバー代替 DNS 名のリスト。			
expirations _object (keys:string, values:string)_	すべての証明書の有効期限。			

ClientCertificateConfiguration

ClientCertificateConfiguration は、データベースロールの TLS クライアント証明書のオペレーター管理の発行を構成します。

登場場所:

- *DatabaseRoleSpec*

Field	Description	Required	Default	Validation
enabled _boolean_	Enabled は、このロールのクライアント証明書の発行をオンにします。 true の場合、ロールでログインを有効にする必要があります。ブロックが存在する場合、デフォルトは true です。		true	

ClientCertificateState

ClientCertificateState は、生成された TLS クライアント証明書の観察された状態を保持します。

登場場所:

- *DatabaseRoleStatus*

Field	Description	Required	Default	Validation
expiration _string_	Expiration は、RFC3339 形式で、生成されたクライアント証明書の有効期限です。			
message _string_	メッセージには、現在の証明書のステータスに関する人間が判読できる説明が含まれています。発行がスキップされた理由、既存のシークレットが変更されない理由など。			

Cluster

クラスターは、CloudNativePG クラスターの EDB Postgres AI によって管理される高可用性の PostgreSQL データベースクラスターの API スキーマを定義します。

Field	Description	Required	Default	Validation
apiVersion _string_	*postgresql.k8s.enterprisedb.io/v1*	True		
kind _string_	*Cluster*	True		
metadata _ *EmbeddedObjectMetadata* _	`metadata`のフィールドについては、Kubernetes API ドキュメントを参照してください。	True		
spec _ *ClusterSpec* _	クラスターの目的の動作の仕様。 詳細情報 <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>	True		

次のページに続く

表 22 – 前のページからの続き

Field	Description	Required	Default	Validation
status _ *ClusterStatus* _	最近観察されたクラスターのステータス。このデータは最新でない可能性があります 。システムによって設定されます。読み取り専用。 詳細情報 <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>			

ClusterImageCatalog

ClusterImageCatalog は、 clusterimagecatalogs API のスキーマです

Field	Description	Required	Default	Validation
apiVersion _string_	*postgresql.k8s.enterprisedb.io/v1*	True		
kind _string_	*ClusterImageCatalog*	True		
metadata _ *EmbeddedObjectMetadata* _	`metadata`のフィールドについては、Kubernetes API ドキュメントを参照してください。	True		
spec _ *ImageCatalogSpec* _	クラスターイメージカタログの目的の動作の仕様。 詳細情報 <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>	True		

ClusterMonitoringTLSConfiguration

ClusterMonitoringTLSConfiguration は、クラスターのモニタリングの TLS 構成を含むタイプです

登場場所:

- *MonitoringConfiguration*

Field	Description	Required	Default	Validation
enabled _boolean_	モニタリングエンドポイントの TLS を有効にします。 このオプションを変更すると、すべてのインスタンスが強制的にロールアウトされます。		false	

ClusterSpec

ClusterSpec は、CloudNativePG クラスターの EDB Postgres AI によって管理される PostgreSQL クラスターの望ましい状態を定義します。

登場場所:

- *cluster-example-projected-volume.yaml*

Field	Description	Required	Default	Validation
description _string_	この PostgreSQL クラスターの説明			
inheritedMetadata _ *EmbeddedObjectMetadata* _	クラスターに関連するすべてのオブジェクトが継承するメタデータ			
imageName _string_	タグ `<image>:<tag>` と決定的で反復可能な展開のためのダイジェストの両方をサポートするコンテナイメージの名前 (*<image>:<tag>@sha256:<digestValu*			
imageCatalogRef _ *ImageCatalogRef* _	イメージカタログ内で使用する PostgreSQL のメジャーバージョンを定義します			
imagePullPolicy _ PullPolicy _	イメージプルポリシー。 `Always`、*Never*、または `IfNotPresent`のいずれか。定義されていない場合、デフォルトの `IfNotPresent`になります。 更新できません。 詳細情報 <https://kubernetes.io/docs/concepts/containers/images>			
schedulerName _string_	指定した場合、ポッドは指定された Kubernetes スケジューラーによってディスパッチされます。指定しない場合、ポッドはデフォルトの スケジューラーによってディスパッチされます。詳細情報 <https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/>			
postgresUID _integer_	画像内の `postgres`ユーザーの UID、デフォルトは `26`		26	

次のページに続く

表 25 – 前のページからの続き

Field	Description	Required	Default	Validation
postgresGID _integer_	画像内の `postgres`ユーザーのGID、デフォルトは`26`		26	
instances _integer_	クラスターで必要なインスタンスの数	True	1	Minimum: 1
minSyncReplicas _integer_	プライマリとの同期レプリケーションで必要なインスタンスの最小数 プライマリ。未定義または0では、スタンバイが 使用できない場合でも書き込みを完了できます。		0	Minimum: 0
maxSyncReplicas _integer_	同期レプリケーションクォーラムのターゲット値。準備ができているスタンバイの数がこれより少ない場合、減らすことができます。 未定義または0は同期レプリケーションを無効にします。		0	Minimum: 0
postgresql _ *PostgresConfiguration* _	PostgreSQL サーバーの構成			
podSelectorRefs _ 動的 pg_hba アドレス解決のための名前付けポッドラベルセレクターを定義します。 オペレーターは、一致するポッド IP を解決し、 ${podselector:NAME} pg_hba ルールの参照を/32 IPv4 または/128 IPv6 マスクを使用して IP ごとに 1 行に展開します。 podSelectorRefs: - name: app-pods selector: matchLabels: app: myapp - name: モニタリングセレクター:matchLabels:role: モニタリング **array_**	PodSelectorRefs は、アドレスフィールドで${podselector:NAME}構文を使用して pg_hba ルールで参照できる名前付けポッドラベルセレクターを定義します。 オペレーターは一致するポッド IP を解決し、インスタンスマネージャーは展開します。 それに応じて pg_hba 行。クラスター自分自身の名前空間のポッドのみが考慮されます。			

次のページに続く

表 25 – 前のページからの続き

Field	Description	Required	Default	Validation
replicationSlots _ *ReplicationSlotsConfiguration* _	レプリケーションスロット管理構成			
bootstrap _ *BootstrapConfiguration* _	このクラスターをブートストラップする手順			
replica _ *ReplicaClusterConfiguration* _	レプリカクラスター構成			
superuserSecret _ LocalObjectReference _	スーパーユーザーのパスワードを含むシークレット。定義されていない場合、ランダムに生成されたパスワードで新しい シークレットが作成されます			
enableSuperuserAccess _boolean_	このオプションが有効になっている場合、オペレーターは`SuperuserSecret`を使用して `postgres`ユーザーパスワードを更新します。シークレットが 存在しない場合、オペレーターは自動的に作成します。この オプションが無効になっている場合、オペレーターは`SuperuserSecret`コンテンツを無視し、自動的に作成されたら削除 し、`NULL`に設定して`postgres` ユーザーのパスワードを空白にします。デフォルトでは無効になっています。			
certificates _ *CertificatesConfiguration* _	CA および関連する証明書の構成			
imagePullSecrets _ LocalObjectReference **array**_	イメージをプルするために使用されるプルシークレットのリスト。ライセンスキー プルシークレットが含まれている場合、そのシークレットは自動的に含まれます。			
storage _ *StorageConfiguration* _	インスタンスのストレージの構成			
serviceAccountTemplate _ *ServiceAccountTemplate* _	サービスアカウントの世代を構成する			

次のページに続く

表 25 – 前のページからの続き

Field	Description	Required	Default	Validation
serviceAccountName _string_	クラスターに使用する同じ名前空間内の既存のサービスアカウントの名前。指定されると、オペレーターは新しいサービスアカウントを作成しません 提供されたものを使用します。これは、複数のクラスター全体で単一の サービスアカウントを共有する場合に役立ちますクラウド IAM 構成の場合。 指定しない場合、サービスアカウントはクラスター名で作成されます。 ServiceAccountTemplate と相互排他的。			MaxLength: 253 Pattern: ^ *:ref: `a-z0-9 <[-a-z0-9]*[a-z0-9]> ?$`
walStorage _ *StorageConfiguration* _	PostgreSQL WAL のストレージの構成書き込み先行ログ			
ephemeralVolumeSource _ EphemeralVolumeSource _	Ephemeral VolumeSource を使用すると、ユーザーはエフェメラルボリュームのソースを構成できます。			
startDelay _integer_	PostgreSQL インスタンスが 正常に起動するまでの秒単位の時間デフォルト 3600。 起動プローブの失敗しきい値は、次の式を使用してこの値から導き出されます。 ceiling(startDelay / 10)。		3600	
stopDelay _integer_	PostgreSQL インスタンスが 正常にシャットダウンできる秒単位の時間デフォルト 1800		1800	
smartStopDelay _integer_	非推奨: 代わりに SmartShutdownTimeout を使用してください			
smartShutdownTimeout _integer_	Postgres のスマートシャットダウンが完了するために予約された時間枠を制御する秒単位の時間。 オペレータが Postgres の高速シャットダウンを要求するための十分な時間を予約してください つまり *stopDelay* - *smartShutdownTimeout*)。デフォルトは 180 秒です。		180	

次のページに続く

表 25 – 前のページからの続き

Field	Description	Required	Default	Validation
switchoverDelay _integer_	プライマリ PostgreSQL インスタンスが許可される時間秒 スイッチオーバー中に正常にシャットダウンします。デフォルト値は 3600 秒 1 時間です。		3600	
failoverDelay _integer_	クラスター内のプライマリ PostgreSQL インスタンスが異常であることが検出された後、フェールオーバーをトリガーするまでに待機する時間秒 		0	
livenessProbeTimeout _integer_	LivenessProbeTimeout は、PostgreSQL インスタンスが許可される時間秒単位であります。ライブネスプローブデフォルト 30 に正常に応答します。 ライブネスプローブの失敗しきい値は、次の式を使用してこの値から導き出されます。 br />ceiling(livenessProbe / 10)。			
affinity _ *AffinityConfiguration* _	ポッドのアフィニティ/アンチアフィニティルール			
topologySpreadConstraints _ TopologySpreadConstraint **array**_	TopologySpreadConstraints は、指定されたトポロジ間で一致するポッドを分散する方法を指定します。 詳細情報 <https://kubernetes.io/docs/concepts/scheduling-eviction/topology-spread-constraints/>			
resources _ ResourceRequirements _	生成されたすべてのポッドのリソース要件。詳細については、 <https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/> を参照してください。			
ephemeralVolumesSizeLimit _ *EphemeralVolumesSizeLimitConfiguration* _	Ephemeral VolumesSizeLimit を使用すると、ユーザーはエフェメラル ボリュームの制限を設定できます			

次のページに続く

表 25 – 前のページからの続き

Field	Description	Required	Default	Validation
priorityClassName _string_	生成されたすべてのポッドで使用される優先度クラスの名前。指定された PriorityClass が存在しない場合、ポッドはスケジュールできません。詳細については、 <https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption/#priorityclass> を参照してください。			
primaryUpdateStrategy _ ローリング更新戦略の例 - unsupervised すべての場合のプライマリの自動更新 レプリカがアップグレードされたデフォルト - supervised を実行するには手動の監視が必要です プライマリのスイッチオーバー primaryUpdateStrategy: unsupervised _	すべてのレプリカが正常に更新された後、ローリング 更新手順中にプライマリサーバーをアップグレードするために従う展開戦略。 自動 *unsupervised* - デフォルト または手動 *supervised* が可能です。		unsupervised	Enum: [unsupervised supervised]
primaryUpdateMethod _ *PrimaryUpdateMethod* _	すべてのレプリカが正常に更新された後、ローリング 更新手順中にプライマリサーバーをアップグレードするための方法。 スイッチオーバー *switchover* またはインプレース *restart* - デフォルト。 注 `switchover`を使用する場合、オペレーターはイメージ名と PostgreSQL 構成パラメーターの両方を同時に変更する更新を拒否して、スイッチオーバープロセス中の 構成の不一致を回避します。			Enum: [switchover restart]
backup _ *BackupConfiguration* _	バックアップに使用される構成			

次のページに続く

表 25 – 前のページからの続き

Field	Description	Required	Default	Validation
nodeMaintenanceWindow _ *NodeMaintenanceWindow* _	Kubernetes ノードのメンテナンスウィンドウを定義する			
licenseKey _string_	クラスターのライセンスキー。空の場合、クラスターは トライアルモードで動作し、有効期限デフォルト 30 日後、オペレーターは 調整試行を停止します。詳細については、オペレーターに付属するライセンス契約を参照してください。			
licenseKeySecret _ SecretKeySelector _	ライセンスキーへの参照。これが設定されている場合、LicenseKey よりも優先されます。			
monitoring _ *MonitoringConfiguration* _	このクラスターの監視インフラストラクチャの構成			
externalClusters _ *ExternalCluster* **array**_	構成で使用される外部クラスターのリスト			
logLevel _string_	インスタンスのログレベル、次の値のいずれか Error、Warning、info デフォルト、debug、trace		info	Enum: [error warning info debug trace]
projectedVolumeTemplate _ ProjectedVolumeSource _	投影ボリュームを定義するために使用されるテンプレート。投影ボリュームは`/projected`ベースフォルダーの下にマウントされます 			
env _ *EnvironmentEnvVar* **array**_	Env は、 Env 形式に従って、環境変数をクラスター内に作成されたポッドに 渡します			
envFrom _ EnvFromSource **array**_	EnvFrom は、EnvFrom フォーマットに従って、環境変数 ソースを Env が使用するポッドに渡します			
managed _ *ManagedConfiguration* _	インスタンスマネージャーによって管理される PostgreSQL の部分で使用される構成			

次のページに続く

表 25 – 前のページからの続き

Field	Description	Required	Default	Validation
seccompProfile _ SeccompProfile _	すべてのポッドとコンテナに適用される Seccomp プロファイル。 デフォルト `RuntimeDefault`			
podSecurityContext _ *#10743* _	クラスターのすべてのポッドに適用される PodSecurityContext をオーバーライドします。 設定されている場合、クラスターのオペレーターのデフォルトの PodSecurityContext をオーバーライドします。 省略すると、オペレーターのデフォルトが使用されます。 このフィールドはありません。 SecurityContextConstraints が存在する場合、効果はありません。			
securityContext _ *#10743* _	クラスターのポッド内のすべてのコンテナに適用されるセキュリティコンテキストをオーバーライドします。 設定されている場合、これはオペレーターのデフォルトのコンテナセキュリティコンテキストをオーバーライドします。 省略すると、オペレーターのデフォルトが使用されます。			
tablespaces _ *TablespaceConfiguration* **array**_	テーブルスペース構成			

次のページに続く

表 25 – 前のページからの続き

Field	Description	Required	Default	Validation
enablePDB _boolean_	クラスター内の `PodDisruptionBudget`リソースを管理します。`true`デフォルト設定として構成されている場合、ポッド中断バジェットは プライマリノードが終了されないように保護します。逆に、`false`に設定すると `PodDisruptionBudget`リソースがなくなり、PostgreSQL クラスターをホスティングするすべてのノード のシャットダウンが許可されます。この後者の構成は、開発/ステージングの目的で使用されるPostgreSQL クラスターに お勧めします。		true	
plugins _ *BackupPluginConfiguration* **array**_	プラグイン構成。対応する構成でロードされるプラグインを含む 			
probes _ *ProbesConfiguration* _	PostgreSQL ポッドに 注入されるプローブの構成。			
primaryLease _ *PrimaryLeaseConfiguration* _	クラスター内の安全なプライマリ 選択を調整するために使用される Kubernetes `Lease`の構成。省略すると、演算子はビルトインの デフォルトを適用します。これらの値は、 フェイルオーバーのタイミングの影響を理解している場合にのみ調整します。			

ClusterStatus

ClusterStatus は、CloudNativePG クラスターの EDB Postgres AI によって管理される PostgreSQL クラスターの観察された状態を定義します。

登場場所:

- *cluster-example-projected-volume.yaml*

Field	Description	Required	Default	Validation
instances _integer_	クラスター内で検出された PVC グループの合計数。既存のインスタンスポッドの数とは異なる場合があります。			
readyInstances _integer_	クラスター内の準備ができているインスタンスの合計数。これは、準備ができているインスタンスポッドの数と同じです。			
selector _string_	Selector は、このクラスターによって管理されるポッドを識別するラベルセレクターのシリアル化された形式であり ます。オペレーターによって設定され、スケールサブリソースを介して公開されますので、オートスケーラー HPA または VPA など 管理対象インスタンスポッドを検出できます。			
instancesStatus _object (keys: *PodStatus*, values:string array)_	InstancesStatus は、インスタンスがどのステータスを示します			
instancesReportedState _object (keys: *PodName*, values: *InstanceReportedState*)_	最後の調整ループ中に報告されたインスタンスの状態			
managedRolesStatus _ *ManagedRoles* _	ManchedRolesStatus は、クラスター内の管理対象ロールの状態を報告します			
tablespacesStatus _ *TablespaceState* **array**_	TablespacesStatus は、クラスター内の宣言的表領域の状態をレポートします			
podSelectorRefs _ *PodSelectorRefStatus* **array**_	PodSelectorRefs には、spec.podSelectorRefs で定義された各名前付けセレクター の解決されたポッド IP が含まれています。			
timelineID _integer_	Postgres クラスターのタイムライン			
topology _ **トポロジスプレッドコンストレイン** _	インスタンスのトポロジ。			

次のページに続く

表 26 – 前のページからの続き

Field	Description	Required	Default	Validation
latestGeneratedNode _integer_	最後に生成されたノードの ID ノード名の衝突を回避するために使用されます 非推奨: このフィールドはもう設定されていません			
currentPrimary _string_	現在のプライマリインスタンス			
targetPrimary _string_	ターゲットプライマリインスタンス、これは以前のものとは異なります スイッチオーバーまたはフェイルオーバー中に			
lastPromotionToken _string_	LastPromotionToken は、レプリカクラスターをプロモーションするために使用された最後の検証されたプロモーショントークンです 			
pvcCount _integer_	このクラスターによって作成された PVC の数			
jobCount _integer_	このクラスターによって作成されたジョブの数			
danglingPVC _string **array**_	このクラスターによって作成され、まだ使用可能な ポッドに接続されていないすべての PVC のリスト			
resizingPVC _string **array**_	ResizingPVC 条件を持つすべての PVC のリスト。			
initializingPVC _string **array**_	このクラスターによって初期化されているすべての PVC のリスト			
healthyPVC _string **array**_	ハングリングも初期化もしていないすべての PVC のリスト			
unusablePVC _string **array**_	別の PVC が欠落しているため使用できないすべての PVC のリスト			
licenseStatus _Status_	ライセンスのステータス			
writeService _string_	現在の書き込みポッド			
readService _string_	読み取りポッドの現在のリスト			
phase _string_	クラスターの現在のフェーズ			
phaseReason _string_	現在のフェーズの理由			

次のページに続く

表 26 – 前のページからの続き

Field	Description	Required	Default	Validation
secretsResourceVersion _ *SecretsResourceVersion* _	シークレットのリソースバージョンのリスト オペレーターによって管理されています。ここでのすべての変更は、インスタンスマネージャーの利益のために行われ、 シークレットデータが更新されます			
configMapResourceVersion _ *ConfigMapResourceVersion* _	構成マップのリソースバージョンのリスト オペレーターによって管理されます。ここでのすべての変更は、インスタンスマネージャーの利益のために行われ、 構成マップデータが更新されます			
certificates _ *CertificatesStatus* _	CA および関連する証明書の構成。デフォルトで初期化されます。			
firstRecoverabilityPoint _string_	最初の復旧可能ポイント、RFC3339 形式の日付として保存されます。 このフィールドは、FirstRecoverabilityPointByMethod のコンテンツから計算されます。 非推奨: フィールドはバックアッププラグインに設定されていません。			
firstRecoverabilityPointByMethod _object (keys: *BackupMethod* , values: **バックアップからのポイント インタイム リカバリー** *PITR*)_	バックアップ方法タイプごとに、RFC3339 形式の日付として保存される最初の復旧可能ポイント。 非推奨: フィールドはバックアッププラグインに設定されていません。			
lastSuccessfulBackup _string_	最後に成功したバックアップ、RFC3339 形式の日付として保存されます。 このフィールドは、LastSuccessfulBackupByMethod の内容から計算されます。 非推奨: フィールドはバックアッププラグインに設定されていません。			

次のページに続く

表 26 – 前のページからの続き

Field	Description	Required	Default	Validation
lastSuccessfulBackupByMethod _object (keys: *BackupMethod*, values: バックアップからのポイント インタイム リカバリー *PITR*)_	最後に成功したバックアップ、バックアップ方法タイプごとにRFC3339 形式の日付として保存されます。 非推奨: フィールドはバックアッププラグインに設定されていません。			
lastFailedBackup _string_	最後に失敗したバックアップ、RFC3339 形式の日付として保存されます。 非推奨: フィールドはバックアッププラグインに設定されていません。			
cloudNativePostgresqlCommitHash _string_	このオペレーターが実行しているコミットハッシュ番号			
currentPrimaryTimestamp _string_	最後にプライマリへの実際の昇格が発生したタイムスタンプ			
currentPrimaryFailingSinceTimestamp _string_	プライマリが異常であることが検出されたときのタイムスタンプ このフィールドは、`.spec.failoverDelay`が設定されるとき、またはオンラインアップグレード中に報告されます			
targetPrimaryTimestamp _string_	新しいプライマリの最後の要求が発生したタイムスタンプ			
poolerIntegrations _*PoolerIntegrations*_	クラスターを参照するプーラーに必要な統合			
cloudNativePostgresqlOperatorHash _string_	オペレーターのバイナリのハッシュ			
operatorCertificateFingerprint _string_	OperatorCertificateFingerprint は、オペレーターの メモリ内クライアント証明書公開キーの SHA256 フィンガープリントです。インスタンスマネージャーはこの フィンガープリントをピン留めして、オペレーターからの要求を認証します。			

次のページに続く

表 26 – 前のページからの続き

Field	Description	Required	Default	Validation
availableArchitectures _ *AvailableArchitecture* **array**_	AvailableArchitectures は、クラスターの使用可能なアーキテクチャをレポートします			
conditions _ **条件** **array**_	クラスターオブジェクトの条件			
instanceNames _string **array**_	クラスター内のインスタンス名のリスト			
onlineUpdateEnabled _boolean_	OnlineUpdateEnabled は、クラスター内でオンラインアップグレードが有効になっているかどうかを示します			
image _string_	イメージには、ポッドが使用するイメージ名が含まれています			
pgDataImageInfo _ *ImageInfo* _	PGDataImageInfo には、現在のデータディレクトリで実行された最新のイメージの詳細が含まれています。			
targetPgDataImageInfo _ *ImageInfo* _	TargetPGDataImageInfo には、進行中のメジャーアップグレードの ターゲットイメージの詳細が含まれています。これは、アップグレードジョブが作成される前に設定され、 正常に完了すると、またはアップグレードがロールバックされるとクリアされます。			
pluginStatus _ *PluginStatus* **array**_	PluginStatus は、ロードされたプラグインのステータスです			
switchReplicaClusterStatus _ *SwitchReplicaClusterStatus* _	SwitchReplicaClusterStatus は、レプリカクラスターへのスイッチのステータスです			
demotionToken _string_	DemotionToken は、pg_controldata からの情報を含む JSON トークンであり、データベースシステム識別子、最新のチェックポイントの タイムライン ID、最新のチェックポイントの REDO 場所、最新のチェックポイントの REDO WAL ファイル、最新のチェックポイントの時間			

次のページに続く

表 26 – 前のページからの続き

Field	Description	Required	Default	Validation
systemID _string_	SystemID は、最後に検出された PostgreSQL のシステム ID です			

ConfigMapResourceVersion

ConfigMapResourceVersion は、オペレーターが管理するシークレットのリソースバージョンです

登場場所:

- *ClusterStatus*

Field	Description	Required	Default	Validation
metrics _object (keys:string, values:string)_	メトリックを渡すために使用されるすべての構成マップのバージョンを含むマップ。 マップキーは構成マップ名であり、マップ値はバージョンです。			

DataDurabilityLevel

Underlying type: string

DataDurabilityLevel は、クラスターインスタンスが使用できないときに同期レプリケーションをどれだけ厳密に適用するかを指定します。オプションは `required` または `preferred` です。

登場場所:

- *SynchronousReplicaConfiguration*

Field	Description
required	DataDurabilityLevelRequired は、データの耐久性が厳密に適用されることを意味します
preferred	DataDurabilityLevelPreferred は、正常なレプリカが利用可能な場合にのみデータの耐久性が適用されることを意味します

DataSource

データソースには、既存のストレージから PostgreSQL クラスターをブートストラップするために必要な構成が含まれています

登場場所:

- *BootstrapRecovery*

Field	Description	Required	Default	Validation
storage _ TypedLocalObjectReference _	インスタンスのストレージの構成	True		
walStorage _ TypedLocalObjectReference _	PostgreSQL WAL のストレージの構成書き込み先行ログ			
tablespaceStorage _object (keys:string, values: TypedLocalObjectReference)_	PostgreSQL テーブルスペースのストレージの構成			

Database

データベースはデータベース API のスキーマです

Field	Description	Required	Default	Validation
apiVersion _string_	*postgresql.k8s.enterprisedb.io/v1*	True		
kind _string_	*Database*	True		
metadata _ *EmbeddedObjectMetadata* _	`metadata`のフィールドについては、Kubernetes API ドキュメントを参照してください。	True		
spec _ *DatabaseSpec* _	目的のデータベースの仕様。 詳細情報 <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>	True		
status _ *DatabaseStatus* _	最近観察されたデータベースのステータス。このデータは最新でない可能性があります 。システムによって設定されます。読み取り専用。 詳細情報 <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>			

DatabaseObjectSpec

DatabaseObjectSpec には、すべてのデータベースオブジェクトに共通のフィールドが含まれています

登場場所:

- *ExtensionSpec*

- *FDWSpec*
- *SchemaSpec*
- *ServerSpec*

Field	Description	Required	Default	Validation
name _string_	オブジェクトの名前拡張子、スキーマ、FDW、サーバー	True		
ensure _ *EnsureOption* _	データベースにオブジェクトスキーマなどを存在するかどうかを指定します 。`present`に設定されている場合、オブジェクトは 存在しない場合に作成されます。`absent`に設定している場合、拡張機能/スキーマは 存在する場合は削除されます。		present	Enum: [present absent]

DatabaseObjectStatus

DatabaseObjectStatus は、管理対象データベースオブジェクトのステータスです

登場場所:

- *DatabaseStatus*

Field	Description	Required	Default	Validation
name _string_	オブジェクトの名前	True		
applied _boolean_	オブジェクトの True はデータベースに正常にインストールされました 	True		
message _string_	メッセージはオブジェクトリコンシリエーションメッセージです			

DatabaseReclaimPolicy

Underlying type: string

DatabaseReclaimPolicy は、データベースのサポート終了のメンテナンスのポリシーを説明します。

登場場所:

- *DatabaseSpec*

Field	Description
delete	DatabaseReclaimDelete は、データベースがそのクレームから 解放されると PostgreSQL クラスターから削除されることを意味します。
retain	DatabaseReclaimRetain は、データベースが管理者による手動 再生利用のために現在のフェーズに残されることを意味します。デフォルトポリシーは Retain です。

DatabaseRole

DatabaseRole は、データベースロール API のスキーマです

登場場所:

- *DatabaseRoleList*

Field	Description	Required	Default	Validation
apiVersion _string_	*postgresql.k8s.enterprisedb.io/v1*	True		
kind _string_	*DatabaseRole*	True		
metadata _*EmbeddedObjectMetadata* _	`metadata`のフィールドについては、Kubernetes API ドキュメントを参照してください。	True		
spec _*DatabaseRoleSpec* _	目的のデータベースロールの仕様。 詳細情報 <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>	True		
status _*DatabaseRoleStatus* _	データベースロールの最近観察されたステータス。このデータは最新でない可能性があります 。システムによって設定されます。読み取り専用。 詳細情報 <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>			

DatabaseRoleList

DatabaseRoleList には、データベースロールのリストが含まれます

Field	Description	Required	Default	Validation
apiVersion _string_	*postgresql.k8s.enterprisedb.io/v1*	True		
kind _string_	*DatabaseRoleList*	True		
metadata _ ListMeta _	`metadata`のフィールドについては、Kubernetes API ドキュメントを参照してください。	True		
items _ :ref:``DatabaseRole`リソースを使用した宣言的ロール管理<*DatabaseRole* リソースを使用した宣言的ロール管理>` **array**_	NA	True		

DatabaseRoleReclaimPolicy

Underlying type: string

DatabaseRoleReclaimPolicy は、ロールのサポートが終了したメンテナンスのポリシーを説明します。

登場場所:

- *DatabaseRoleSpec*

Field	Description
delete	DatabaseRoleReclaimDelete は、ロールがそのクレームからリリース されるときに Kubernetes から削除されることを意味します。
retain	DatabaseRoleReclaimRetain は、管理者による手動での 再利用のためにロールが現在のフェーズに残されることを意味します。デフォルトポリシーは Retain です。

DatabaseRoleRef

DatabaseRoleRef は、PostgreSQL 内で使用可能なロールの参照です

登場場所:

- *TablespaceConfiguration*

Field	Description	Required	Default	Validation
name _string_	NA			

DatabaseRoleSpec

DatabaseRoleSpec は Postgres のロールを表します

登場場所:

- :ref:``DatabaseRole` リソースを使用した宣言的ロール管理 <*DatabaseRole* リソースを使用した宣言的ロール管理>`

Field	Description	Required	Default	Validation
name _string_	ロールの名前	True		
comment _string_	ロールの説明			
ensure _ *EnsureOption* _	ロールが `present`または `absent`であることを確認します-デフォルトは「present」です		present	Enum: [present absent]
passwordSecret _ LocalObjectReference _	ロールのパスワードが含まれるシークレット存在する場合。 ヌルの場合、パスワードは DisablePassword が設定されていない限り無視されます。 設定されている場合、シークレットは`kubernetes.io/basic-auth`フォーマットに従います そして両方を含む必要があります`username`および`password`フィールド。			
connectionLimit _integer_	ロールがログインできる場合、これはロールが作成できる同時 接続の数を指定します。 `-1`デフォルトは制限なしを意味します。		-1	
validUntil _ バックアップからのポイントインタイム リカバリー *PITR* _	ロールのパスワードが有効でなくなる日時。 省略すると、パスワードの有効期限はありませんデフォルト。			
inRoles _string **array**_	このロールが 新しいメンバーとしてすぐに追加される 1 つ以上の既存のロールのリスト。デフォルト空。 リストの変更は、ロールの作成時だけでなく、 `GRANT`および`REVOKE`ステートメントを介して既存のロールに適用されます。			

次のページに続く

表 38 – 前のページからの続き

Field	Description	Required	Default	Validation
inherit _boolean_	ロールが、メンバーであるロールの特権を「継承」するかどうか。 デフォルトは`true`です。		true	
disablePassword _boolean_	DisablePassword は、Postgres でロールのパスワードを NULL に設定する必要があることを示します			
superuser _boolean_	ロールが`superuser`であるかどうか データベース内のすべてのアクセス 制限をオーバーライドできる人 - スーパーユーザーステータスは危険であり、 本当に必要な場合にのみ使用する必要があります。新しいスーパーユーザーを作成するには、自分自身が スーパーユーザーである必要があります。デフォルトは`false`です。			
createdb _boolean_	`true`に設定されている場合、定義されるロールは新しいデータベースを作成できます 。`false`デフォルトを指定すると、ロールがデータベースを作成する 機能が拒否されます。			
createrole _boolean_	ロールが他のロールの作成、変更、ドロップ、コメント セキュリティラベルの変更、およびメンバーシップの付与または取り消しを許可されるかどうか ほかのロール。デフォルトは`false`です。			
login _boolean_	ロールにログインを許可するかどうか。 `login` 属性を持つロールはユーザーと考えられます。この属性のないロール はデータベース特権の管理に役立ちますが、通常の意味でのユーザーではありません 。デフォルトは`false`です。			

次のページに続く

表 38 – 前のページからの続き

Field	Description	Required	Default	Validation
replication _boolean_	ロールがレプリケーションロールであるかどうか。レプリケーションモード物理または論理レプリケーションで サーバーに接続し、次のことができるようにするには、ロールがこの 属性を持つか、スーパーユーザーである必要があります。レプリケーションスロットを作成またはドロップします。 `replication`属性を持つロールは非常に高い特権ロールであり、 レプリケーションに実際に使用されるロールでのみ使用する必要があります。デフォルト は`false`です。			
bypassrls _boolean_	ロールがすべての行レベルセキュリティ RLS ポリシーをバイパスするかどうか。 デフォルトは`false`です。			
cluster _ LocalObjectReference _	対応するクラスター	True		
databaseRoleReclaimPolicy _ *DatabaseRoleReclaimPolicy* _	このロールのサポートが終了したメンテナンスのポリシー		retain	Enum: [delete retain]
clientCertificate _ *ClientCertificateConfiguration* _	ClientCertificate は、クラスターのクライアント CA によって署名された、このロールの TLS クライアント 証明書を生成および更新するようにオペレーターを構成します。証明書は `<databaserolename>-client-cert`という名前のシークレットに保存されます。 ログインが true である必要があります。			

DatabaseRoleStatus

DatabaseRoleStatus は、データベースロールの観察された状態を定義します

登場場所:

- :ref:``DatabaseRole` リソースを使用した宣言的ロール管理 <*DatabaseRole* リソースを使用した宣言的

ロール管理>`

Field	Description	Required	Default	Validation
observedGeneration _integer_	同期された最新の 目的の状態を表すシーケンス番号			
applied _boolean_	ロールが正しくリコンサイルされた場合、Apply は true です			
message _string_	メッセージはリコンシリエーションエラーメッセージです			
secretResourceVersion _string_	SecretResourceVersion は、パスワードシークレットのリソースバージョンです 最後にロールに適用されました。それを変更すると、調整がトリガーされます。			
clientCertificate _ *ClientCertificateState* _	ClientCertificate は、クライアント証明書の発行が有効になっているときに、生成された TLS クライアント 証明書の観察された状態を保持します。			
conditions _ 条件 **array**_	DatabaseRole オブジェクトの条件			

DatabaseSpec

DatabaseSpec は、Postgresql データベースの仕様であり、PostgreSQL の `CREATE DATABASE`、`ALTER DATABASE`、および `DROP DATABASE` SQL コマンドを中心に構築されています。

登場場所:

- *Postgres* データベースの宣言的管理

Field	Description	Required	Default	Validation
cluster _ LocalObjectReference _	データベースをホストする PostgreSQL クラスターの名前。	True		
ensure _ *EnsureOption* _	PostgreSQL データベースが `present`または `absent`であることを確認します。デフォルトは「present」です。		present	Enum: [present absent]
name _string_	PostgreSQL 内に作成するデータベースの名前。この設定は変更できません。	True		

次のページに続く

表 40 – 前のページからの続き

Field	Description	Required	Default	Validation
owner _string_	`CREATE DATABASE`の`OWNER`パラメーターにマップします。 `ALTER DATABASE`の`OWNER TO`コマンドにマップします。 PostgreSQL 内のデータベースを所有するユーザーのロール名。	True		
template _string_	`CREATE DATABASE`の`TEMPLATE`パラメーターにマッピングされます。この設定は 変更できません。このデータベースを作成するテンプレートの名前 。			
encoding _string_	`CREATE DATABASE`の`ENCODING`パラメーターにマッピングされます。この設定は 変更できません。データベースで使用する文字セットエンコーディング。			
locale _string_	`CREATE DATABASE`の`LOCALE`パラメーターにマッピングされます。この設定は 変更できません。新しいデータベースにデフォルトの照合順序と文字 分類を設定します。			
localeProvider _string_	`CREATE DATABASE`の`LOCALE_PROVIDER`パラメーターにマッピングされます。この 設定は変更できません。このオプションは、新しいクラスターで作成された データベースのロケールプロバイダーを設定します。 PostgreSQL 16 から利用できます。			
localeCollate _string_	`CREATE DATABASE`の`LC_COLLATE`パラメーターにマッピングされます。この 設定は変更できません。			
localeCType _string_	`CREATE DATABASE`の`LC_CTYPE`パラメーターにマッピングされます。この設定は 変更できません。			

次のページに続く

表 40 – 前のページからの続き

Field	Description	Required	Default	Validation
icuLocale _string_	`CREATE DATABASE`の`ICU_LOCALE`パラメーターにマッピングされます。この 設定は変更できません。ICU プロバイダーを使用する場合に ICU ロケールを指定します。このオプションでは、 `localeProvider`を `icu`に設定する必要があります。 PostgreSQL 15 から利用できます。			
icuRules _string_	`CREATE DATABASE`の`ICU_RULES`パラメーターにマッピングされます。この設定は 変更できません。デフォルトの照合動作をカスタマイズする追加の照合ルールを指定します デフォルトの照合順序。このオプションでは `localeProvider`を`icu`に設定する必要があります。 PostgreSQL 16 から利用できます。			
builtinLocale _string_	`CREATE DATABASE`の`BUILTIN_LOCALE`パラメーターにマッピングされます。この 設定は変更できません。ビルトインプロバイダーを使用する場合にロケール名を指定します ビルトインプロバイダー。このオプションでは、 `localeProvider`を `builtin`に設定する必要があります。 PostgreSQL 17 から利用できます。			
collationVersion _string_	`CREATE DATABASE`の`COLLATION_VERSION`パラメーターにマッピングされます。この 設定は変更できません。			

次のページに続く

表 40 – 前のページからの続き

Field	Description	Required	Default	Validation
isTemplate _boolean_	`CREATE DATABASE`および `ALTER DATABASE`の `IS_TEMPLATE`パラメーターにマッピングされます。 trueの場合、このデータベースはテンプレートとみなされ、`CREATEDB`権限を持つユーザーが クローンを作成できます。			
allowConnections _boolean_	`CREATE DATABASE`および `ALTER DATABASE`の `ALLOW_CONNECTIONS`パラメーターにマッピングされます。 falseの場合、誰もこのデータベースに接続できません。			
connectionLimit _integer_	`CREATE DATABASE`および `ALTER DATABASE`の `CONNECTION LIMIT`句にマッピングされます。このデータベースへの 同時接続の数。 -1 デフォルトは制限なしを意味します。			
tablespace _string_	`CREATE DATABASE`の `TABLESPACE`パラメーターにマップします。 `ALTER DATABASE`の `SET TABLESPACE`コマンドにマップします。 新しいデータベースに関連付けられるテーブルスペースの名前 PostgreSQLの名前。このテーブルスペースは、このデータベースで作成されるオブジェクトに使用されるデフォルトの テーブルスペースになります。			
databaseReclaimPolicy _ *DatabaseReclaimPolicy* _	このデータベースのサポートが終了したメンテナンスのポリシー。		retain	Enum: [delete retain]
schemas _ *SchemaSpec* **array**_	データベースで管理するスキーマのリスト			
extensions _ *ExtensionSpec* **array**_	データベースで管理する拡張機能のリスト			

次のページに続く

表 40 – 前のページからの続き

Field	Description	Required	Default	Validation
fdws _ *FDWSpec* **array**_	データベースで管理する外部データラッパーのリスト			
servers _ *ServerSpec* **array**_	データベースで管理する外部サーバーのリスト			

DatabaseStatus

DatabaseStatus は、データベースの観察された状態を定義します

登場場所:

- *Postgres* データベースの宣言的管理

Field	Description	Required	Default	Validation
observedGeneration _integer_	同期された最新の 目的の状態を表すシーケンス番号			
applied _boolean_	データベースが正しくリコンサイルされた場合、Applied は true です			
message _string_	Message はリコンシリエーション出力メッセージです			
schemas _ *DatabaseObjectStatus* **array**_	Schemas は、管理対象スキーマのステータスです			
extensions _ *DatabaseObjectStatus* **array**_	Extensions は、管理対象拡張機能のステータスです			
fdws _ *DatabaseObjectStatus* **array**_	FDWs は、管理対象 FDW のステータスです			
servers _ *DatabaseObjectStatus* **array**_	Servers は、管理対象サーバーのステータスです			

EPASConfiguration

EPASConfiguration には、 EDB Postgres Advanced Server 固有の構成が含まれています

登場場所:

- *PostgresConfiguration*

Field	Description	Required	Default	Validation
audit _boolean_	true の場合 edb_audit ロギングを有効にします			
tde _ *TDEConfiguration* _	TDE 構成			

EmbeddedObjectMetadata

EmbeddedObjectMetadata には、クラスターに関連するすべてのリソースによって継承されるメタデータが含まれています

登場場所:

- *ClusterSpec*

Field	Description	Required	Default	Validation
labels _object (keys:string, values:string)_	NA			
annotations _object (keys:string, values:string)_	NA			

EnsureOption

Underlying type: string

EnsureOption は、PostgreSQL インスタンスにロールの存在または不存在を強制するかどうかを表します

登場場所:

- *DatabaseObjectSpec*
- *DatabaseRoleSpec*
- *DatabaseSpec*
- *ExtensionSpec*
- *FDWSpec*
- *OptionSpec*
- *RoleConfiguration*
- *SchemaSpec*
- *ServerSpec*

Field	Description
present	NA
absent	NA

EphemeralVolumesSizeLimitConfiguration

Ephemeral VolumesSizeLimitConfiguration には、エフェメラルストレージの構成が含まれます

登場場所:

- *ClusterSpec*

Field	Description	Required	Default	Validation
shm _ Quantity _	Shm は、共有メモリボリュームのサイズ制限です			
temporaryData _ Quantity _	TemporaryData は、一時データボリュームのサイズ制限です			

ExtensionConfiguration

ExtensionConfiguration は、PostgreSQL 拡張機能をクラスターに追加するために使用される構成です。

登場場所:

- *CatalogImage*
- *ImageInfo*
- *PostgresConfiguration*

Field	Description	Required	Default	Validation
name _string_	拡張機能の名前。必須。 59 文字の制限により、 拡張機能の Kubernetes ボリューム名 63 文字制限を取得するときにオペレーターが追加するプレフィックスのための余地が残ります。	True		MaxLength: 59 MinLength: 1 Pattern: ^ *:ref: `a-z0-9 <[-a-z0-9_]*[a-z0-9]> ?$` <br /*
image _ ImageVolumeSource _	拡張機能を含む画像。			

次のページに続く

表 46 – 前のページからの続き

Field	Description	Required	Default	Validation
extension_control_path _string **array**_	extension_control_path に追加するイメージ内のディレクトリのリスト。 定義されていない場合、デフォルトの「/share」。			
dynamic_library_path _string **array**_	Dynamic_library_path に追加するイメージ内のディレクトリのリスト。 定義されていない場合、デフォルトの「/lib」。			
ld_library_path _string **array**_	ld_library_path に追加するイメージ内のディレクトリのリスト。			
bin_path _string **array**_	PostgreSQL プロセスの`PATH`環境変数に追加するイメージ内のディレクトリのリスト。			
env _ *ExtensionEnvVar* **array**_	Env は、この拡張機能の PostgreSQL プロセスで設定されるカスタム環境変数のリストです。変数が特定の拡張機能に対して正しいことを確認するのは クラスタ管理者の責任です。これらの変数の変更を有効にするには、手動でクラスターを再起動する必要があることに注意してください。			

ExtensionEnvVar

ExtensionEnvVar は、特定の拡張イメージボリュームの環境変数を定義します。

登場場所:

- *ExtensionConfiguration*

Field	Description	Required	Default	Validation
name _string_	PostgreSQL プロセスに注入される環境変数の名前。	True		MinLength: 1 Pattern: *^[a-zA-Z_][a-zA-Z0-9_]*$*

次のページに続く

表 47 – 前のページからの続き

Field	Description	Required	Default	Validation
value _string_	環境変数の値。 CloudNativePG クラスターの EDB Postgres AI は、プレースホルダー展開のサポートを使用して、この値の直接 置換を実行します。 ${*image_root*}プレースホルダーは、拡張機能のボリュームの絶対マウントパス たとえば、*/extensions/my-extension*)。これは アプリケーションまたはライブラリが マウントされたイメージ内の特定のディレクトリを見つけられるようにする場合に特に役立ちます。 認識されないプレースホルダーは拒否されます。値にリテラル${...} を含めるには、$${...}としてエスケープします。	True		MinLength: 1

ExtensionSpec

ExtensionSpec はデータベースに拡張機能を構成します

登場場所:

- *DatabaseSpec*

Field	Description	Required	Default	Validation
name _string_	オブジェクトの名前拡張子、スキーマ、FDW、サーバー	True		
ensure _ *EnsureOption* _	データベースにオブジェクトスキーマなどを存在するかどうかを指定します 。`present`に設定されている場合、オブジェクトは 存在しない場合に作成されます。`absent`に設定している場合、拡張機能/スキーマは 存在する場合は削除されます。		present	Enum: [present absent]

次のページに続く

表 48 – 前のページからの続き

Field	Description	Required	Default	Validation
version _string_	インストールする拡張機能のバージョン。空の場合、オペレーターは デフォルトバージョン拡張機能の制御ファイルで指定されたものをインストールします。	True		
schema _string_	拡張機能のオブジェクトをインストールするスキーマの名前 拡張機能がコンテンツを再配置できる場合。指定されずデフォルトデフォルト、および拡張機能の制御ファイルがスキーマを指定しない場合、現在のデフォルトのオブジェクト作成スキーマ が使用されます。	True		

ExternalCluster

外部クラスターは、構成の他のセクションで使用される外部クラスターへの接続パラメーターを表します

登場場所:

- *ClusterSpec*

Field	Description	Required	Default	Validation
name _string_	サーバー名必須	True		
connectionParameters _object (keys:string, values:string)_	dbname、ホスト、ユーザー名などの接続パラメーターのリスト			
sslCert _ SecretKeySelector _	この インスタンスへの接続に使用される SSL 証明書への参照			
sslKey _ SecretKeySelector _	この インスタンスへの接続に使用される SSL プライベートキーへの参照			
sslRootCert _ SecretKeySelector _	この インスタンスへの接続に使用される SSL CA 公開キーへの参照			

次のページに続く

表 49 – 前のページからの続き

Field	Description	Required	Default	Validation
password _ SecretKeySelector _	サーバーへの接続に使用されるパスワードへの参照。パスワードが提供される場合、CloudNativePG クラスターの EDB Postgres AI は `/controller/external/NAME/pass`に PostgreSQL の パスファイルを作成します。ここで「NAME」は クラスターの名前)。このパスファイルは、現在の PostgreSQL `Cluster`からリモート PostgreSQL サーバーへの接続を確立するときに 接続文字列で自動的に参照されます。これにより、外部クラスターの 安全で効率的なパスワード管理が保証されます。			
barmanObjectStore _ BarmanObjectStoreConfiguration _	barman-cloud ツールスイートの構成			
plugin _ *BackupPluginConfiguration* _	この外部クラスターの WAL アーカイブとバックアップを処理するプラグインの構成 	True		

FDWSpec

FDWSpec はデータベースに外部データラッパーを構成します

登場場所:

- *DatabaseSpec*

Field	Description	Required	Default	Validation
name _string_	オブジェクトの名前拡張子、スキーマ、FDW、サーバー	True		

次のページに続く

表 50 – 前のページからの続き

Field	Description	Required	Default	Validation
ensure _ *EnsureOption* _	データベースにオブジェクトスキーマなどを存在するかどうかを指定します 。`present`に設定されている場合、オブジェクトは 存在しない場合に作成されます。`absent`に設定している場合、拡張機能/スキーマは 存在する場合は削除されます。		present	Enum: [present absent]
handler _string_	ハンドラーファンクションの名前たとえば、「postgres_fdw_handler」。 ハンドラーが指定されない場合、これは空になります。その場合、 デフォルトハンドラーは、FDW 拡張機能の作成時に登録されます。			
validator _string_	バリデーターファンクションの名前たとえば、「postgres_fdw_validator」。 バリデーターが指定されていない場合、これは空になります。その場合、 デフォルトのバリデーターは、FDW 拡張機能の作成時に登録されます。			
owner _string_	所有者は、外部データラッパーを所有するデータベースロールを指定します。 ロールには、ターゲットデータベースのスーパーユーザー権限が必要です。			
options _ *OptionSpec* **array**_	Options は、FDW の構成オプションを指定します。			
usage _ *UsageSpec* **array**_	FDW の `USAGE`特権が付与または取り消されるロールのリスト。			

FailoverQuorum

FailoverQuorum には、PG クラスターの現在のフェールオーバークォーラムステータスに関する情報が含まれています。これは、プライマリノードのインスタンスマネージャーによって更新され、オペレーターによってゼロにリセットされて更新をトリガーします。

Field	Description	Required	Default	Validation
apiVersion _string_	*postgresql.k8s.enterprisedb.io/v1*	True		
kind _string_	*FailoverQuorum*	True		
metadata _ *EmbeddedObjectMetadata* _	`metadata`のフィールドについては、Kubernetes API ドキュメントを参照してください。	True		
status _ *FailoverQuorumStatus* _	最近観察されたフェールオーバークォーラムのステータス。			

FailoverQuorumStatus

FailoverQuorumStatus は、PG クラスターのフェールオーバークォーラムの最新の観察されたステータスです。

登場場所:

- *FailoverQuorum*

Field	Description	Required	Default	Validation
method _string_	最新に報告されたメソッド値が含まれています。			
standbyNames _string **array**_	StandbyNames は、同期する可能性のある インスタンス名のリストです。			
standbyNumber _integer_	StandbyNumber は、トランザクションが 応答を待機する必要がある同期スタンバイの数です。			
primary _string_	Primary は、このオブジェクトを最後に更新したプライマリインスタンスの名前です 。			

ImageCatalog

ImageCatalog は、imagecatalogs API のスキーマです

Field	Description	Required	Default	Validation
apiVersion _string_	*postgresql.k8s.enterprisedb.io/v1*	True		
kind _string_	*ImageCatalog*	True		
metadata _ *EmbeddedObjectMetadata* _	`metadata`のフィールドについては、Kubernetes API ドキュメントを参照してください。	True		

次のページに続く

表 53 – 前のページからの続き

Field	Description	Required	Default	Validation
spec _ *ImageCatalogSpec* _	ImageCatalog の目的の動作の仕様。 詳細情報 <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>	True		

ImageCatalogComponentRef

ImageCatalogComponentRef は、ImageCatalog または ClusterImageCatalog の componentImages リスト内の名前付けイメージを識別します。

登場場所:

- *PgBouncerSpec*

Field	Description	Required	Default	Validation
apiGroup _string_	APIGroup は、参照されるリソースのグループです。 APIGroup が指定されない場合、指定された Kind はコア API グループにある必要があります。 他のサードパーティタイプの場合、APIGroup が必要です。			
kind _string_	Kind は、参照されるリソースのタイプです	True		
name _string_	Name は参照されるリソースの名前です	True		
key _string_	Key は、カタログの componentImages リスト内のエントリを識別します。	True		MaxLength: 63 Pattern: ^ *:ref: `a-z0-9 <[-a-z0-9]*[a-z0-9]>* ?$`

ImageCatalogRef

ImageCatalogRef は、ImageCatalog 内のメジャーバージョンへの参照を定義します

登場場所:

- *ClusterSpec*

Field	Description	Required	Default	Validation
apiGroup _string_	APIGroup は、参照されるリソースのグループです。 APIGroup が指定されない場合、指定された Kind はコア API グループにある必要があります。 他のサードパーティタイプの場合、APIGroup が必要です。			
kind _string_	Kind は、参照されるリソースのタイプです	True		
name _string_	Name は参照されるリソースの名前です	True		
major _integer_	ImageCatalog から使用する PostgreSQL のメジャーバージョン	True		

ImageCatalogSpec

ImageCatalogSpec は、目的のイメージカタログを定義します

登場場所:

- *ClusterImageCatalog*
- *ClusterImageCatalog*

Field	Description	Required	Default	Validation
images _ *CatalogImage* **array**_	カタログで使用可能なカタログ画像のリスト	True		MaxItems: 8 MinItems: 1
componentImages _ *CatalogComponentImage* **array**_	ComponentImages は、PostgreSQL 以外のコンポーネント pgbouncer などの名前付け画像のリストです。キーはカタログ内で一意である必要があります。			MaxItems: 32

ImageInfo

ImageInfo には、PostgreSQL イメージに関する情報が含まれています

登場場所:

- *ClusterStatus*

Field	Description	Required	Default	Validation
image _string_	Image はイメージ名です	True		
majorVersion _integer_	MajorVersion はイメージのメジャーバージョンです	True		
extensions _ *ExtensionConfiguration* **array**_	Extensions には、現在のイメージで使用可能なコンテナイメージ拡張機能が含まれています			

Import

インポートには、外部クラスターの論理スナップショットからデータベースを初期化する構成が含まれています

登場場所:

- *BootstrapInitDB*

Field	Description	Required	Default	Validation
source _ *ImportSource* _	輸入元	True		
type _ *SnapshotType* _	インポートタイプ。 `microservice`または`monolith`です。	True		Enum: [microservice monolith]
databases _string **array**_	インポートするデータベース	True		
roles _string **array**_	インポートするロール			
postImportApplicationSQL _string **array**_	インポートされた直後にアプリケーション データベースでスーパーユーザーとして実行される SQL クエリのリスト - 細心の注意を払って使用されます デフォルトでは空。マイクロサービスタイプでのみ使用できます。			
schemaOnly _boolean_	true に設定すると、*pg_restore* `の`*pre-data* `および `*post-data* `セクションのみが呼び出され、データのインポートが回避されます。デフォルト `*false*			

次のページに続く

表 58 – 前のページからの続き

Field	Description	Required	Default	Validation
pgDumpExtraOptions _string **array**_	`pg_dump`コマンドに渡すカスタムオプションのリスト。 重要: 注意して使用します。オペレーターはこれらのオプションを検証しません そして、特定のフラグは意図された機能またはデザインを妨げる場合があります。 提供されたオプションが環境および目的の動作と互換性があることを確認するのは自分の責任です。			
pgRestoreExtraOptions _string **array**_	`pg_restore`コマンドに渡すカスタムオプションのリスト。 重要: 注意して使用します。オペレーターはこれらのオプションを検証しません そして、特定のフラグは意図された機能またはデザインを妨げる場合があります。 提供されたオプションが環境および目的の動作と互換性があることを確認するのは自分の責任です。			
pgRestorePreDataOptions _string **array**_	`pre-data` セクションで`pg_restore`コマンドに渡すカスタムオプション。この設定は、汎用の `pgRestoreExtraOptions`値をオーバーライドします。 重要: 注意して使用してください。オペレーターはこれらのオプションを検証しません そして、特定のフラグは意図された機能またはデザインを妨げる場合があります。 提供されたオプションが環境および目的の動作と互換性があることを確認するのは自分の責任です。			

次のページに続く

表 58 – 前のページからの続き

Field	Description	Required	Default	Validation
pgRestoreDataOptions _string **array**_	`data` セクションで`pg_restore`コマンドに渡すカスタムオプション。この設定は、汎用の `pgRestoreExtraOptions`値をオーバーライドします。 重要: 注意して使用してください。オペレーターはこれらのオプションを検証しません そして、特定のフラグは意図された機能またはデザインを妨げる場合があります。 提供されたオプションが環境および目的の動作と互換性があることを確認するのは自分の責任です。			
pgRestorePostdataOptions _string **array**_	`post-data` セクションで`pg_restore`コマンドに渡すカスタムオプション。この設定は、汎用の `pgRestoreExtraOptions`値をオーバーライドします。 重要: 注意して使用してください。オペレーターはこれらのオプションを検証しません そして、特定のフラグは意図された機能またはデザインを妨げる場合があります。 提供されたオプションが環境および目的の動作と互換性があることを確認するのは自分の責任です。			

ImportSource

ImportSource は、論理スナップショットのソースを説明します

登場場所:

- *OpenShift* の重要な概念

Field	Description	Required	Default	Validation
externalCluster _string_	インポートに使用される外部クラスターの名前	True		

InstanceID

InstanceID には、インスタンスを識別する情報が含まれます

登場場所:

- *BackupStatus*

Field	Description	Required	Default	Validation
podName _string_	ポッド名			
ContainerID _string_	コンテナ ID			
sessionID _string_	インスタンスマネージャーのセッション ID。これは、インスタンスマネージャーの 起動時に生成され、再起動のたびに変更されますコンテナのリブートを含む。バックアップのような長時間実行オペレーション中にインスタンスマネージャーが再起動されたかどうかを検出するために使用されます。これにより、実行中のバックアッププロセスが終了します。			

InstanceReportedState

InstanceReportedState は、調整ループ中に最後に報告されたインスタンスの状態を説明します

登場場所:

- *ClusterStatus*

Field	Description	Required	Default	Validation
isPrimary _boolean_	インスタンスがプライマリかどうかを示します	True		
timeLineID _integer_	インスタンスがどの TimelineId にあるかを示します			
ip _string_	インスタンスの IP アドレス	True		

IsolationCheckConfiguration

IsolationCheckConfiguration には、liveness プローブの分離チェック機能の構成が含まれています

登場場所:

- *LivenessProbe*

Field	Description	Required	Default	Validation
enabled _boolean_	liveness プローブのプライマリ分離チェックが有効になっているかどうか		true	
requestTimeout _integer_	プライマリ分離チェック中の要求のミリ秒単位のタイムアウト		1000	
connectionTimeout _integer_	プライマリ分離チェック中の接続のミリ秒単位のタイムアウト		1000	

LDAPBindAsAuth

LDAPBindAsAuth は、LDAP のバインド認証を使用するための必須フィールドを提供します

登場場所:

- *LDAPConfig*

Field	Description	Required	Default	Validation
prefix _string_	バインド認証オプションのプレフィックス			
suffix _string_	バインド認証オプションのサフィックス			

LDAPBindSearchAuth

LDAPBindSearchAuth は、バインド+検索 LDAP 認証プロセスを使用するための必須フィールドを提供します

登場場所:

- *LDAPConfig*

Field	Description	Required	Default	Validation
baseDN _string_	ユーザー検索を開始するルート DN			
bindDN _string_	ディレクトリにバインドするユーザーの DN			
bindPassword _ SecretKeySelector _	ディレクトリにバインドするユーザーのパスワードを使用したシークレット			
searchAttribute _string_	ユーザー名と一致する属性			

次のページに続く

表 64 – 前のページからの続き

Field	Description	Required	Default	Validation
searchFilter _string_	検索+バインド認証を実行するときに使用する検索フィルター			

LDAPConfig

LDAPConfig には、LDAP 認証に必要なパラメーターが含まれています

登場場所:

- *PostgresConfiguration*

Field	Description	Required	Default	Validation
server _string_	LDAP ホスト名または IP アドレス			
port _integer_	LDAP サーバーポート			
scheme _ *LDAPScheme* _	使用する LDAP スキーマ、使用可能なオプションは `ldap`および `ldaps`です			Enum: [ldap ldaps]
bindAsAuth _ *LDAPBindAsAuth* _	認証構成としてバインド			
bindSearchAuth _ *LDAPBindSearchAuth* _	バインド+サーチ認証構成			
tls _boolean_	LDAP over TLS を有効にするには、 'true' に設定します。 'false' がデフォルトです			

LDAPScheme

Underlying type: string

LDAPScheme は、 LDAP の使用可能なスキームを定義します

登場場所:

- *LDAPConfig*

Field	Description
ldap	NA
ldaps	NA

LivenessProbe

LivenessProbe は liveness プローブの構成です

登場場所:

- *ProbesConfiguration*

Field	Description	Required	Default	Validation
initialDelaySeconds _integer_	コンテナが起動してから liveness プローブが開始されるまでの秒数。 詳細情報 <https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes>			
timeoutSeconds _integer_	プローブがタイムアウトになるまでの秒数。 デフォルト 1 秒。最小値は 1 です。 詳細情報 <https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes>			
periodSeconds _integer_	プローブを実行する頻度秒単位。 デフォルト 10 秒。最小値は 1 です。			
successThreshold _integer_	失敗した後に成功したと見なされるプローブの最小連続成功。 デフォルト 1。 liveness および startup の場合は 1 である必要があります。最小値は 1 です。			
failureThreshold _integer_	成功した後に失敗したと考えられるプローブの最小連続失敗。 デフォルト 3。最小値は 1 です。			

次のページに続く

表 67 – 前のページからの続き

Field	Description	Required	Default	Validation
terminationGracePeriodSeconds _integer_	プローブが失敗したときにポッドが正常に終了するために必要な秒単位のオプションの期間。 猶予期間は、ポッドで実行されているプロセスが送信された後の秒単位の期間です。終了信号と、プロセスが強制的に実行される時間 Kill シグナルで停止します。 この値をプロセスの予想されるクリーンアップ時間より長く設定します。 この値が nil の場合、ポッドの終了 GracePeriodSecond が使用されます。それ以外の場合、この 値はポッド仕様が提供する値をオーバーライドします。 値は負でない整数である必要があります。値 0 は、kill シグナルを介してすぐに停止することを示しますシャットダウンの機会なし。 これはベータフィールドであり、ProbeTerminationGracePeriod 機能ゲートを有効にする必要があります。 最小値は 1 です。未設定の場合に使用されます。			
isolationCheck _ *IsolationCheckConfiguration* _	プライマリ インスタンスの liveness プローブを拡張する機能を構成します。基本的なチェックに加えて、これは プライマリが Kubernetes API サーバーおよびその レプリカから分離されているかどうかを検証し、ネットワーク パーティションまたは API が使用できない場合に安全にシャットダウンできることを保証しますが検出されます。デフォルトで有効になっています。			

ManagedConfiguration

ManageConfiguration は、インスタンスマネージャーによって管理される PostgreSQL の部分を表します

登場場所:

- *ClusterSpec*

Field	Description	Required	Default	Validation
roles _ *RoleConfiguration* **array**_	`Cluster`によって管理されるデータベースロール			
services _ *ManagedServices* _	`Cluster`によって管理されるサービスロール			

ManagedRoles

ManageRoles は、クラスターの管理対象ロールのステータスを追跡します

登場場所:

- *ClusterStatus*

Field	Description	Required	Default	Validation
byStatus _object (keys: *DatabaseRoleStatus* , values:string array)_	ByStatus は、各状態のロールのリストを提供します			
cannotReconcile _object (keys:string, values:string array)_	CannotReconcile は、調整できないロールとその原因の説明をリストします。障害は、PostgreSQL オブジェクトを所有するロールのドロップなど、または Kubernetes などに発生する可能性があります。例 参照されたパスワード Secret は取得できません。			
passwordStatus _object (keys:string, values: *PasswordState*)_	PasswordStatus は、各管理対象ロールの最後のトランザクション ID とパスワードのシークレットバージョンを提供します			

ManagedService

ManchedService は、クラスターによって管理される特定のサービスを表します。これには、サービスの種類とそれに関連するテンプレート仕様が含まれます。

登場場所:

- *ManagedServices*

Field	Description	Required	Default	Validation
selectorType _ *ServiceSelectorType* _	SelectorType は、サービスが持つセレクターのタイプを指定します。 有効な値は「rw」、「r」、および「ro」で、読み取り-書き込み、読み取り、および読み取り専用のサービスを表します。	True		Enum: [rw r ro]
updateStrategy _ *ServiceUpdateStrategy* _	UpdateStrategy は、サービスの違いを調整する方法を説明します		patch	Enum: [patch replace]
serviceTemplate _ *ServiceTemplateSpec* _	ServiceTemplate は、サービスのテンプレート仕様です。	True		

ManagedServices

ManagedServices は、クラスターによって管理されるサービスを表します。

登場場所:

- *ManagedConfiguration*

Field	Description	Required	Default	Validation
disabledDefaultServices _ *ServiceSelectorType* **array**_	DisabledDefaultServices は、デフォルトで無効になっているサービスタイプのリストです。 有効な値は、読み取りおよび読み取り専用のサービスを表す「r」および「ro」です。			Enum: [rw r ro]
additional _ *ManagedService* **array**_	Additional は、ユーザーが指定した追加の管理サービスのリストです。			

Metadata

メタデータは metav1.ObjectMeta に似た構造ですが、controller-gen によって解析され、ユーザーに適切な CRD を作成できます。PodTemplateSpec のコメントには、コアデータ型を使用しない理由の説明があります。

登場場所:

- *PodTemplateSpec*
- *ServiceAccountTemplate*

- *ServiceTemplateSpec*

Field	Description	Required	Default	Validation
name _string_	リソースの名前。特定のタイプでのみサポートされています			
labels _object (keys:string, values:string)_	オブジェクトを整理および分類するために使用できる文字列キーと値のマップ スコープと選択。レプリケーションコントローラ およびサービスのセレクターと一致する場合があります。 詳細<http://kubernetes.io/docs/user-guide/labels>			
annotations _object (keys:string, values:string)_	アノテーションは、リソースと共に保存される非構造化キー値マップであり、任意のメタデータを保存および取得するように外部ツールによって 設定される場合があります。これらはクエリー可能ではなく、オブジェクトを変更するときに保存する必要があります。 詳細<http://kubernetes.io/docs/user-guide/annotations>			

MonitoringConfiguration

MonitoringConfiguration は、特定のクラスターのすべての監視構成を含むタイプです

登場場所:

- *ClusterSpec*

Field	Description	Required	Default	Validation
disableDefaultQueries _boolean_	デフォルトのクエリーを注入するかどうか。 デフォルトのクエリーをクラスターに注入したくない場合は、`true`に設定します。 デフォルト false。		false	
customQueriesConfigMap _ConfigMapKeySelector **array**_	カスタムクエリを含む構成マップのリスト			

次のページに続く

表 73 – 前のページからの続き

Field	Description	Required	Default	Validation
customQueriesSecret _ SecretKeySelector **array**_	カスタムクエリーを含むシークレットのリスト			
enablePodMonitor _boolean_	`PodMonitor`を有効または無効にする 非推奨この機能は、今後のリリースで削除される予定です。この機能が必要な場合は、PodMonitor を手動で作成できます。		false	
tls _ *ClusterMonitoringTLSConfiguration* _	メトリックエンドポイントの TLS 通信を構成します。 tls.enabled オプションを変更すると、すべてのインスタンスが強制的にロールアウトされます。			
podMonitorMetricRelabelings _ *#10743* **array**_	`PodMonitor`のメトリックの再ラベル付けのリスト。取り込まれる前にサンプルに適用されます。 非推奨この機能は、今後のリリースで削除される予定です。この機能が必要な場合は、PodMonitor を手動で作成できます。			
podMonitorRelabelings _ *#10743* **array**_	`PodMonitor`の再ラベル付けのリスト。スクレイピングの前にサンプルに適用されます。 非推奨 この機能は、今後のリリースで削除される予定です。この機能が必要な場合は、PodMonitor を手動で作成できます。			
metricsQueriesTTL _ Duration _	クエリから計算されたメトリックが最新であると考えられる間隔。 それを超えると、新しいスクレイプがクエリの再実行をトリガーします。 設定されていない場合、デフォルトの 30 秒に これをゼロに設定すると、キャッシュメカニズムが無効になり、PostgreSQL サーバーに高負荷が発生する可能性があります。			

NodeMaintenanceWindow

NodeMaintenanceWindow には、基になるノードのアップグレード中にオペレーターが使用する情報が含まれています。

このオプションは、選択したストレージがポッドをノード間で自由に移動できない場合にのみ役立ちます。

登場場所:

- *ClusterSpec*

Field	Description	Required	Default	Validation
reusePVC _boolean_	既存の PVC を再利用するノードが再びアップするまで待機するかどうか他の場所で再作成します-*instances* >1 の場合		true	
inProgress _boolean_	進行中のノードのメンテナンスアクティビティはありますか?		false	

OnlineConfiguration

OnlineConfiguration には、オンラインボリュームスナップショットの構成パラメーターが含まれています

登場場所:

- *BackupSpec*
- *ScheduledBackupSpec*
- *VolumeSnapshotConfiguration*

Field	Description	Required	Default	Validation
waitForArchive _boolean_	false の場合、ファンクションは、バックアップの完了後すぐに戻ります。WAL がアーカイブされるのを待たずに。 この動作は、WAL アーカイブを独立して監視するバックアップソフトウェアでのみ役立ちます。それ以外の場合、バックアップの一貫性を保つために必要な WAL が欠落しており、バックアップを役に立たない可能性があります。 デフォルトまたはこのパラメータが true の場合、pg_backup_stop は、アーカイブが 有効になっているときに WAL がアーカイブされるまで待機します。<br / >スタンバイでは、これは archive_mode = Always の場合にのみ待機することを意味します。 プライマリの書き込みアクティビティが低い場合、プライマリで pg_switch_wal を実行して即時実行をトリガーします。 セグメントスイッチ。		true	
immediateCheckpoint _boolean_	PostgreSQL サーバーの `checkpoint_completion_target`設定に従って、バックアップの初期チェックポイントの I/O ワークロードを 制限するかどうかを制御します。 true に設定されている場合、即時チェックポイントが使用されます。つまり、PostgreSQL ができるだけ早くチェックポイントを完了する ことを意味します。デフォルトでは`false`。			

OptionSpec

OptionSpec は、オプションの名前、値、保証フィールドを保持します

登場場所:

- *FDWSpec*

- *ServerSpec*

Field	Description	Required	Default	Validation
name _string_	オプションの名前	True		
value _string_	オプションの値	True		
ensure _ *EnsureOption* _	データベースにオプションが存在するかどうかを指定します 。 `present`に設定されている場合、オプションは 存在しない場合は作成されます。 `absent`に設定している場合、 オプションが存在する場合は削除されます。		present	Enum: [present absent]

PasswordState

PasswordState は、管理対象ロール構成のパスワードの状態を表します

登場場所:

- *ManagedRoles*

Field	Description	Required	Default	Validation
transactionID _integer_	PostgreSQL のロール定義に影響を与える最後のトランザクション ID			
resourceVersion _string_	パスワードシークレットのリソースバージョン			

PgBouncerIntegrationStatus

PgBouncerIntegrationStatus は、クラスターを参照する pgbouncer プーラーに必要な統合をカプセル化します

登場場所:

- *PoolerIntegrations*

Field	Description	Required	Default	Validation
secrets _string **array**_	NA			

PgBouncerPoolMode

Underlying type: string

PgBouncerPoolMode は PgBouncer のモードです

検証

- 列挙型 [セッショントランザクション]

登場場所:

- *PgBouncerSpec*

PgBouncerSecrets

PgBouncerSecrets には、pgbouncer が使用するシークレットのバージョンが含まれています

登場場所:

- *PoolerSecrets*

Field	Description	Required	Default	Validation
authQuery _ *SecretVersion* _	認証クエリーのシークレットバージョン			

PgBouncerSpec

PgBouncerSpec は、PgBouncer の構成方法を定義します

登場場所:

- *PoolerSpec*

Field	Description	Required	Default	Validation
poolMode _ *PgBouncerPoolMode* _	プールモード。デフォルト`session`		session	Enum: [session transaction]
serverTLSSecret _ LocalObjectReference _	ServerTLSSecret は、TLS シークレットを指す場合、PostgreSQL に対して認証するときに使用される pgbouncer の `server_tls_key_file`および`server_tls_cert_file`を提供します。			

次のページに続く

表 80 – 前のページからの続き

Field	Description	Required	Default	Validation
serverCASecret _ LocalObjectReference _	ServerCASecret は、PgBouncer の server_tls_ca_file、PostgreSQL 証明書を検証するためのルート CA を提供します			
clientCASecret _ LocalObjectReference _	ClientCASecret は、PgBouncer の client_tls_ca_file、クライアントを検証するためのルート CA を提供します証明書			
clientTLSSecret _ LocalObjectReference _	ClientTLSSecret は、クライアントを受け入れるために使用される PgBouncer の client_tls_key_file プライベートキー と client_tls_cert_file 証明書を提供します			
authQuerySecret _ LocalObjectReference _	認証 クエリに使用する必要があるユーザーの資格情報。指定される場合、AuthQuery たとえば "SELECT usename, passwd FROM pg_catalog.pg_shadow WHERE usename=$1") も指定する必要があり、自動 CNP クラスター統合はトリガーされません。 />非推奨。			
authQuery _string_	特定のユーザーのパスワード のハッシュをダウンロードするために使用されるクエリー。デフォルト"SELECT usename, passwd FROM public.user_search($1)"。 指定される場合、AuthQuerySecret も指定する必要があり、 自動 CNP クラスター統合はトリガーされません。			
parameters _object (keys:string, values:string)_	PgBouncer に渡される追加パラメーター - 構成可能なオプションのリストについては、CNP ドキュメントをご確認ください 			
pg_hba _string **array**_	PostgreSQL ホストベースの認証ルール pg_hba.conf ファイルに追加する行 			

次のページに続く

表 80 – 前のページからの続き

Field	Description	Required	Default	Validation
paused _boolean_	`true`に設定すると、PgBouncer は PostgreSQL サーバーから切断し、最初にすべてのクエリが完了するのを待機し、この値が `false`デフォルトに設定されるまで、すべての新しい クライアント接続を一時停止します。内部的に オペレーターは PgBouncer の `PAUSE`および `RESUME`コマンドを呼び出します。		false	
image _string_	Image は、使用する pgbouncer コンテナイメージです。設定されると、 ImageCatalogRef および演算子のデフォルトよりも優先されますが、ポッドテンプレートで明示的に設定されたイメージによって オーバーライドされます。			
imageCatalogRef _ *ImageCatalogComponentRef* _	ImageCatalogRef は、ImageCatalog または ClusterImageCatalog のエントリをポイントします。 Image と相互排他的。			

PluginConfiguration

PluginConfiguration は、このクラスターをリコンサイルするためにロードする必要があるプラグインを指定します

登場場所:

- *ClusterSpec*
- *ExternalCluster*

Field	Description	Required	Default	Validation
name _string_	Name はプラグイン名です	True		
enabled _boolean_	このプラグインを使用する場合、Enabled は true		true	

次のページに続く

表 81 – 前のページからの続き

Field	Description	Required	Default	Validation
isWALArchiver _boolean_	プラグインを WAL アーカイバーとしてマークします。最大で 1 つのプラグインを WAL アーカイバーとして 指定できます。これは `.spec.backup.barmanObjectStore 成が存在する場合、有効にできません。		false	
parameters _object (keys:string, values:string)_	Parameters はプラグインの構成です			

PluginStatus

PluginStatus は、ロードされたプラグインのステータスです

登場場所:

- *ClusterStatus*

Field	Description	Required	Default	Validation
name _string_	Name はプラグインの名前です	True		
version _string_	バージョンは、 最新の調整ループによってロードされたプラグインのバージョンです。	True		
capabilities _string **array**_	機能は プラグインの機能のリストです。			
operatorCapabilities _string **array**_	OperatorCapabilities は、リコンサイラーに関する プラグインの機能のリストです。			
walCapabilities _string **array**_	WALCapabilities は、WAL 管理に関する プラグインの機能のリストです。			
backupCapabilities _string **array**_	BackupCapabilities は、バックアップ管理に関する プラグインの機能のリストです。			
restoreJobHookCapabilities _string **array**_	RestoreJobHookCapabilities は、RestoreJobHook 管理に関する プラグインの機能のリストです。			

次のページに続く

表 82 – 前のページからの続き

Field	Description	Required	Default	Validation
status _string_	ステータスには、 SetStatusInCluster インターフェイスを介してプラグインによって報告されたステータスが含まれます			

PodName

Underlying type: string

PodName はポッドの名前です

登場場所:

- *ClusterStatus*
- トポロジスプレッドコンストレイン

PodSelectorRef

PodSelectorRef は、pg_hba ルールで使用する名前付けポッドラベルセレクターを定義します。クラスターの名前空間のセレクターと一致するポッドの IP は解決され、`${podselector:NAME}` 構文を介した pg_hba アドレス拡張で利用できるようになります。

登場場所:

- *ClusterSpec*

Field	Description	Required	Default	Validation
name _string_	Name は、pg_hba ルールでこのセレクターを参照するために使用される識別子 アドレスフィールドの${podselector:NAME}構文を介して。	True		MinLength: 1 Pattern: ^ *:ref:`a-z <[a-z0-9_-]*[a-z0-9]>* ?$`
selector _ LabelSelector _	Selector は、IP を解決する必要があるポッドを識別するラベルセレクターであり ます。クラスターの名前空間内のポッドのみが考慮されます。	True		

PodSelectorRefStatus

PodSelectorRefStatus には、名前付けセレクターの解決されたポッド IP が含まれています。

登場場所:

- *ClusterStatus*

Field	Description	Required	Default	Validation
name _string_	Name は、仕様の PodSelectorRef の名前に対応します。	True		
ips _string **array**_	IPs は、セレクターに一致するポッド IP のリストです。各 IP は単一のアドレス CIDR 表記なし。			

PodStatus

Underlying type: string

PodStatus は、ポッドの考えられるステータスを表します

登場場所:

- *ClusterStatus*

PodTemplateSpec

PodTemplateSpec は、ユーザーが Pod 生成のテンプレートを設定できる構造です。

残念ながら、生成された CRD にはメタデータセクションのフィールドが含まれないため、corev1.PodTemplateSpec タイプは使用できません。

参考 https://github.com/kubernetes-sigs/controller-tools/issues/385 https://github.com/kubernetes-sigs/controller-tools/issues/448 <https://github .com/prometheus-operator/prometheus-operator/issues/3041>

登場場所:

- *PoolerSpec*

Field	Description	Required	Default	Validation
metadata _ *EmbeddedObjectMetadata* _	`metadata`のフィールドについては、Kubernetes API ドキュメントを参照してください。			
spec _ PodSpec _	ポッドの目的の動作の仕様。 詳細情報 <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>			

PodTopologyLabels

基になるタイプ *object*

PodTopologyLabels は、ポッドのトポロジを表します。map[labelName]labelValue

登場場所:

- トポロジスプレッドコンストレイン

Pooler

Pooler は、プーラー API のスキーマです

Field	Description	Required	Default	Validation
apiVersion _string_	*postgresql.k8s.enterprisedb.io/v1*	True		
kind _string_	*Pooler*	True		
metadata _ *EmbeddedObjectMetadata* _	`metadata`のフィールドについては、Kubernetes API ドキュメントを参照してください。	True		
spec _ *PoolerSpec* _	プーラーの目的の動作の仕様。 詳細情報 <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>	True		
status _ *PoolerStatus* _	プーラーの最近観察されたステータス。このデータは最新でない可能性があります 。システムによって設定されます。読み取り専用。 詳細情報 <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>			

PoolerIntegrations

PoolerIntegrations は、クラスターを参照するプーラーに必要な統合をカプセル化します

登場場所:

- *ClusterStatus*

Field	Description	Required	Default	Validation
pgBouncerIntegration _ *PgBouncerIntegrationStatus* _	NA			

PoolerMonitoringConfiguration

PoolerMonitoringConfiguration は、特定のプーラーのすべての監視構成を含むタイプです。

クラスターのモニタリング構成をミラーしますが、今のところカスタムクエリ部分はありません。

登場場所:

- *PoolerSpec*

Field	Description	Required	Default	Validation
enablePodMonitor _boolean_	`PodMonitor`を有効または無効にする 非推奨この機能は、今後のリリースで削除される予定です。この機能が必要な場合は、PodMonitor を手動で作成できます。		false	
podMonitorMetricRelabelings _ *#10743* **array**_	`PodMonitor`のメトリックの再ラベル付けのリスト。取り込まれる前にサンプルに適用されます。 非推奨この機能は、今後のリリースで削除される予定です。この機能が必要な場合は、PodMonitor を手動で作成できます。			
podMonitorRelabelings _ *#10743* **array**_	`PodMonitor`の再ラベル付けのリスト。スクレイピングの前にサンプルに適用されます。 非推奨 この機能は、今後のリリースで削除される予定です。この機能が必要な場合は、PodMonitor を手動で作成できます。			
tls _ *PoolerMonitoringTLSConfiguration* _	メトリックエンドポイントの TLS 通信を構成します。 tls.enabled オプションを変更すると、すべてのインスタンスが強制的にロールアウトされます。			

PoolerMonitoringTLSConfiguration

PoolerMonitoringTLSConfiguration は、プーラーモニタリングの TLS 構成を含むタイプです

登場場所:

- *PoolerMonitoringConfiguration*

Field	Description	Required	Default	Validation
enabled _boolean_	モニタリングエンドポイントの TLS を有効にします。 このオプションを変更すると、すべてのインスタンスが強制的にロールアウトされます。		false	

PoolerPhase

Underlying type: string

PoolerPhase は、プーラーのライフサイクルフェーズを表します。

検証

- 列挙型 [アクティブな一時停止、非アクティブな失敗]

登場場所:

- *PoolerStatus*

Field	Description
active	PoolerPhaseActive は、プーラーが通常に実行されており、トラフィックを処理していることを意味します。
paused	PoolerPhasePaused は、PgBouncer がアップして実行されているが、 spec.pgbouncer.paused が true であるため、新しいクライアント 接続をキューに保持していることを意味します。展開 調整し続けます。一時停止を解除するとトランジションがアクティブに戻ります。
inactive	PoolerPhaseInactive は、前提条件リソースクラスター、シークレット、証明書が欠落しているため、プーラーが進行できないことを意味します。 コントローラは、前提条件が表示されるまで定期的に再試行します。特定の原因については、 status.phaseReason を確認します。
failed	PoolerPhaseFailed は、構成エラーのためプーラーを調整できないことを意味します 。詳細については、status.phaseReason を確認してください。

PoolerSecrets

PoolerSecrets には、使用されるすべてのシークレットのバージョンが含まれています

登場場所:

- *PoolerStatus*

Field	Description	Required	Default	Validation
clientTLS _ *SecretVersion* _	クライアント TLS シークレットバージョン			
serverTLS _ *SecretVersion* _	サーバー TLS シークレットバージョン			
serverCA _ *SecretVersion* _	サーバー CA シークレットバージョン			
clientCA _ *SecretVersion* _	クライアント CA シークレットバージョン			
pgBouncerSecrets _ *PgBouncerSecrets* _	PgBouncer が使用するシークレットのバージョン			

PoolerSpec

PoolerSpec は、プーラーの目的の状態を定義します

登場場所:

- プーラー構成

Field	Description	Required	Default	Validation
cluster _ LocalObjectReference _	これは、プーラーが動作するクラスター参照です。 プーラー名は、同じ名前空間内のクラスター名と決して一致しないはずです。	True		
type _ *PoolerType* _	トラフィックの転送先のサービスの種類。デフォルト `rw`		rw	Enum: [rw ro r]
instances _integer_	必要なレプリカの数。デフォルト 1		1	
template _ *PodTemplateSpec* _	作成する Pod のテンプレート			
pgbouncer _ *PgBouncerSpec* _	PgBouncer 構成	True		
deploymentStrategy _ DeploymentStrategy _	pgbouncerで既存のポッドを新しいポッドに置き換えるために使用する展開戦略			

次のページに続く

表 92 – 前のページからの続き

Field	Description	Required	Default	Validation
monitoring _ *PoolerMonitoringConfiguration* _	このプーラーの監視インフラストラクチャの構成。			
serviceTemplate _ *ServiceTemplateSpec* _	作成するサービスのテンプレート			
serviceAccountName _string_	プーラーに使用する同じ名前空間内の既存のサービスアカウントの名前。 指定されると、オペレーターは新しいサービスアカウントを作成しません 提供されたものを使用します。これは、複数のプーラー間で単一の サービスアカウントを共有する場合に役立ちますクラウド IAM 構成など。 指定しない場合、プーラー名でサービスアカウントが作成されます。			MaxLength: 253 Pattern: ^ *:ref: `a-z0-9 <[-a-z0-9]*[a-z0-9]> ?$`

PoolerStatus

PoolerStatus は、プーラーの観察された状態を定義します

登場場所:

- プーラー構成

Field	Description	Required	Default	Validation
secrets _ *PoolerSecrets* _	config オブジェクトのリソースバージョン			
instances _integer_	スケジュールしようとしているポッドの数			
phase _ *PoolerPhase* _	フェーズは、プーラーの全体的なライフサイクル状態を要約します。			Enum: [active paused inactive failed]
phaseReason _string_	PhaseReason は、現在のフェーズの人間が判読可能な説明です。			

次のページに続く

表 93 – 前のページからの続き

Field	Description	Required	Default	Validation
image _string_	画像は、オペレーターが このプーラーに使用している解決された pgbouncer コンテナイメージです。spec.template からのオーバーライドを含みます。 フェーズがアクティブまたは一時停止している間、このフィールドは展開 実際に何を反映していますか実行します。フェーズが非アクティブまたは失敗している間、最後に 正常に解決された値またはプーラーがこれまでに成功したことがない場合は空になる場合があります。			
error _string_	Error は最新の入場検証エラーです			

PoolerType

Underlying type: string

PoolerType は接続プールのタイプ、つまりターゲットとしているサービスを意味します。使用できる値は `rw` および `ro` です。

検証

- 列挙型 [rw ro r]

登場場所:

- *PoolerSpec*

PostgresConfiguration

PostgresConfiguration は PostgreSQL 構成を定義します

登場場所:

- *ClusterSpec*

Field	Description	Required	Default	Validation
parameters _object (keys:string, values:string)_	PostgreSQL 構成オプション postgresql.conf			

次のページに続く

表 94 – 前のページからの続き

Field	Description	Required	Default	Validation
synchronous _ *SynchronousReplicaConfiguration* _	PostgreSQL 同期レプリケーション機能の構成			
pg_hba _string **array**_	PostgreSQL ホストベースの認証ルール pg_hba.conf ファイルに追加する行 。 ${podselector:NAME}構文を使用して、ポッドセレクターを参照します。 ルールが展開されますそのセレクターと一致する各 Pod IP ごとに。			
pg_ident _string **array**_	PostgreSQL のユーザー名マップルール pg_ident.conf ファイルに追加される行 			
epas _ *EPASConfiguration* _	EDB Postgres Advanced Server 固有の構成			
syncReplicaElectionConstraint _ *SyncReplicaElectionConstraints* _	同期レプリカが満たす要件。これは、「synchronous_standby_names」パラメーターの 設定に影響します。			
shared_preload_librari _string **array**_	デフォルトのライブラリに追加する共有プリロードライブラリのリスト			
ldap _ *LDAPConfig* _	LDAP 構成を指定するオプション			
promotionTimeout _integer_	インスタンスをプライマリに昇格するときに待機する最大秒数を指定します。 デフォルト値は 40000000 で、秒単位で 1 年より大きい 無限タイムアウトをシミュレートするのに十分な大きさ			
enableAlterSystem _boolean_	このパラメータが true の場合、ユーザーは `ALTER SYSTEM`を呼び出すことができます。CloudNativePG Cluster クラスターのこの EDB Postgres AI で これはデバッグとトラブルシューティングにのみ使用してください。 デフォルトは false です。			

次のページに続く

表 94 – 前のページからの続き

Field	Description	Required	Default	Validation
extensions _ *ExtensionConfiguration* **array**_	追加する拡張機能の構成			

PrimaryLeaseConfiguration

PrimaryLeaseConfiguration は、プライマリインスタンスが保持および更新する Kubernetes Lease のタイミングを構成し、安全なプライマリ選挙を調整します。これらの値は、基になる Kubernetes のリーダー選択パラメーターに直接マッピングされます。

登場場所:

- *ClusterSpec*

Field	Description	Required	Default	Validation
leaseDurationSeconds _integer_	プライマリリースが有効期限が切れて別のインスタンスが取得するまでに有効であると考えられる秒単位の時間。これは `renewDeadlineSeconds`より大きい必要があります。 デフォルト 15。		15	Minimum: 1
renewDeadlineSeconds _integer_	現在のプライマリがあきらめて停止するまでに、リースの更新を再試行し続ける秒単位の時間 リース。 `leaseDurationSeconds`より小さい必要があります。 デフォルト 10。		10	Minimum: 1
retryPeriodSeconds _integer_	非ホルダーインスタンスがリースの取得または 更新を再試行する頻度。 デフォルト 2。		2	Minimum: 1
releasedLeaseDurationSeconds _integer_	プライマリがクリーンシャットダウンでリースを明示的に解放したときに書き込まれる秒単位の TTL。レプリカは、完全なリース期間の有効期限が 待たずに昇格できます。デフォルト 1。		1	Minimum: 1

PrimaryUpdateMethod

Underlying type: string

PrimaryUpdateMethod は、ローリング更新の一部としてクラスターのプライマリインスタンスをアップグレードするときに使用する方法を定義します。デフォルトの方法は「restart」です

登場場所:

- *ClusterSpec*

Field	Description
switchover	PrimaryUpdateMethodSwitchover は、プライマリインスタンスをアップグレードする必要がある場合に、オペレーターが別の更新された レプリカにスイッチオーバーすることを意味します。 注この方法を使用する場合、オペレーターはイメージ名の両方を変更する更新を拒否します および PostgreSQL 構成パラメーターを同時に使用して、スイッチオーバープロセス中の 構成の不一致を回避します。
restart	PrimaryUpdateMethodRestart は、オペレーターがプライマリインスタンスをインプレースで再起動することを意味します アップグレードが必要な場合

PrimaryUpdateStrategy

Underlying type: string

PrimaryUpdateStrategy には、ローリング更新の一部としてクラスターのプライマリサーバーをアップグレードするときに従うべき戦略が含まれています

登場場所:

- *ClusterSpec*

Field	Description
supervised	PrimaryUpdateStrategySupervised は、オペレーターがプライマリ サーバー `supervised`を更新する前に ユーザーがスイッチオーバー要求を手動で発行するのを待機する必要があることを意味します
unsupervised	PrimaryUpdateStrategyUnsupervised は、オペレーターが 選択した PrimaryUpdate メソッドを別の更新されたレプリカに続行し、自動的に更新する プライマリサーバー `unsupervised`、デフォルト を意味します。

Probe

プローブは、コンテナに対して実行されるヘルスチェックを記述して、コンテナが生きているか、トラフィックを受信する準備ができているかを判断します。

登場場所:

- *LivenessProbe*
- *ProbeWithStrategy*

Field	Description	Required	Default	Validation
initialDelaySeconds _integer_	コンテナが起動してから liveness プローブが開始されるまでの秒数。 詳細情報 <https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes>			
timeoutSeconds _integer_	プローブがタイムアウトになるまでの秒数。 デフォルト 1 秒。最小値は 1 です。 詳細情報 <https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes>			
periodSeconds _integer_	プローブを実行する頻度秒単位。 デフォルト 10 秒。最小値は 1 です。			
successThreshold _integer_	失敗した後に成功したと見なされるプローブの最小連続成功。 デフォルト 1。 liveness および startup の場合は 1 である必要があります。最小値は 1 です。			
failureThreshold _integer_	成功した後に失敗したと考えられるプローブの最小連続失敗。 デフォルト 3。最小値は 1 です。			

次のページに続く

表 98 – 前のページからの続き

Field	Description	Required	Default	Validation
terminationGracePeriodSeconds _integer_	プローブが失敗したときにポッドが正常に終了するために必要な秒単位のオプションの期間。 猶予期間は、ポッドで実行されているプロセスが送信された後の秒単位の期間です。終了信号と、プロセスが強制的に実行される時間 Kill シグナルで停止します。 この値をプロセスの予想されるクリーンアップ時間より長く設定します。 この値が nil の場合、ポッドの終了 GracePeriodSecond が使用されます。それ以外の場合、この 値はポッド仕様が提供する値をオーバーライドします。 値は負でない整数である必要があります。値 0 は、kill シグナルを介してすぐに停止することを示しますシャットダウンの機会なし。 これはベータフィールドであり、ProbeTerminationGracePeriod 機能ゲートを有効にする必要があります。 最小値は 1 です。未設定の場合に使用されます。			

ProbeStrategyType

Underlying type: string

ProbeStrategyType は、PostgreSQL インスタンスの準備ができていることを宣言するために使用されるストラテジーのタイプです

登場場所:

- *ProbeWithStrategy*

Field	Description
pg_isready	ProbeStrategyPgIsReady は、 pg_isready ツールを使用して PostgreSQL が起動されているかどうかを判断することを意味します

次のページに続く

表 99 – 前のページからの続き

Field	Description
streaming	ProbeStrategyStreaming は、 pg_isready がポジティブで、レプリカが ストリーミングレプリケーションを介して現在のプライマリに接続されており、ラグが指定されている場合 制限内であることを意味します。
query	ProbeStrategyQuery は、サーバーがスーパーユーザーデータベースに接続でき、「-- ping」のような単純なクエリを実行できることを意味します

ProbeWithStrategy

ProbeWithStrategy は、スタートアッププローブの構成です

登場場所:

- *ProbesConfiguration*

Field	Description	Required	Default	Validation
initialDelaySeconds _integer_	コンテナが起動してから liveness プローブが開始されるまでの秒数。 詳細情報 <https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes>			
timeoutSeconds _integer_	プローブがタイムアウトになるまでの秒数。 デフォルト 1 秒。最小値は 1 です。 詳細情報 <https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes>			
periodSeconds _integer_	プローブを実行する頻度秒単位。 デフォルト 10 秒。最小値は 1 です。			
successThreshold _integer_	失敗した後に成功したと見なされるプローブの最小連続成功。 デフォルト 1。 liveness および startup の場合は 1 である必要があります。最小値は 1 です。			
failureThreshold _integer_	成功した後に失敗したと考えられるプローブの最小連続失敗。 デフォルト 3。最小値は 1 です。			

次のページに続く

表 100 – 前のページからの続き

Field	Description	Required	Default	Validation
terminationGracePeriodSeconds _integer_	プローブが失敗したときにポッドが正常に終了するために必要な秒単位のオプションの期間。 猶予期間は、ポッドで実行されているプロセスが送信された後の秒単位の期間です。終了信号と、プロセスが強制的に実行される時間 Kill シグナルで停止します。 この値をプロセスの予想されるクリーンアップ時間より長く設定します。 この値が nil の場合、ポッドの終了 GracePeriodSecond が使用されます。それ以外の場合、この 値はポッド仕様が提供する値をオーバーライドします。 値は負でない整数である必要があります。値 0 は、kill シグナルを介してすぐに停止することを示しますシャットダウンの機会なし。 これはベータフィールドであり、ProbeTerminationGracePeriod 機能ゲートを有効にする必要があります。 最小値は 1 です。未設定の場合に使用されます。			
type _ *ProbeStrategyType* _	プローブ戦略			Enum: [pg_isready streaming query]
maximumLag _ Quantity _	遅延制限。 `streaming`ストラテジーにのみ使用されます			

ProbesConfiguration

ProbesConfiguration は、PostgreSQL ポッドに注入されるプローブの構成を表します

登場場所:

- *ClusterSpec*

Field	Description	Required	Default	Validation
startup _ *ProbeWithStrategy* _	起動プローブ構成	True		
liveness _ *LivenessProbe* _	liveness プローブ構成	True		
readiness _ *ProbeWithStrategy* _	Readiness Probe 構成	True		

Publication

パブリケーションは、パブリケーション API のスキーマです

Field	Description	Required	Default	Validation
apiVersion _string_	*postgresql.k8s.enterprisedb.io/v1*	True		
kind _string_	*Publication*	True		
metadata _ *EmbeddedObjectMetadata* _	`metadata`のフィールドについては、Kubernetes API ドキュメントを参照してください。	True		
spec _ *PublicationSpec* _	NA	True		
status _ *PublicationStatus* _	NA	True		

PublicationReclaimPolicy

Underlying type: string

PublicationReclaimPolicy は、パブリケーションのサポートが終了したメンテナンスのポリシーを定義します。

登場場所:

- *PublicationSpec*

Field	Description
delete	PublicationReclaimDelete は、パブリケーションがそのクレームからリリース されるときに Kubernetes から削除されることを意味します。
retain	PublicationReclaimRetain は、パブリケーションが管理者による手動 再生のために現在のフェーズに残されることを意味します。デフォルトポリシーは Retain です。

PublicationSpec

PublicationSpec は、パブリケーションの望ましい状態を定義します

登場場所:

- 宣言的なパブリケーションおよびサブスクリプションオブジェクトを介した論理レプリケーション

Field	Description	Required	Default	Validation
cluster _ LocalObjectReference _	「パブリッシャー」を識別する PostgreSQL クラスターの名前	True		
name _string_	PostgreSQL 内のパブリケーションの名前	True		
dbname _string_	パブリケーションが「パブリッシャー」クラスターにインストールされるデータベースの名前 	True		
parameters _object (keys:string, values:string)_	`WITH`句のパブリケーションパラメーター一部 PostgreSQL `CREATE PUBLICATION`コマンド			
target _ *PublicationTarget* _	PostgreSQL `CREATE PUBLICATION`コマンドで予想されるパブリケーションのターゲット	True		
publicationReclaimPolicy _ *PublicationReclaimPolicy* _	この出版物のサポートが終了したメンテナンスのポリシー		retain	Enum: [delete retain]

PublicationStatus

PublicationStatus は、パブリケーションの観察された状態を定義します

登場場所:

- 宣言的なパブリケーションおよびサブスクリプションオブジェクトを介した論理レプリケーション

Field	Description	Required	Default	Validation
observedGeneration _integer_	同期された最新の 目的の状態を表すシーケンス番号			
applied _boolean_	パブリケーションが正しくリコンサイルされた場合、Apply は true です			

次のページに続く

表 105 – 前のページからの続き

Field	Description	Required	Default	Validation
message _string_	Message はリコンシリエーション出力メッセージです			

PublicationTarget

PublicationTarget は、このパブリケーションが公開するものです

登場場所:

- *PublicationSpec*

Field	Description	Required	Default	Validation
allTables _boolean_	パブリケーションを、将来作成されるテーブルを含むデータベース内のすべてのテーブル の変更をレプリケートするものとしてマークします。 PostgreSQL の `FOR ALL TABLES`に対応します。			
objects _ *PublicationTargetObject* **array**_	次のスキーマオブジェクトのみ			MaxItems: 100000

PublicationTargetObject

PublicationTargetObject は、パブリッシュするオブジェクトです

登場場所:

- *PublicationTarget*

Field	Description	Required	Default	Validation
tablesInSchema _string_	指定されたスキーマのリスト内のすべてのテーブル 将来に作成されるテーブルを含む テーブルの変更をレプリケートするものとしてパブリケーションをマークします。PostgreSQL の `FOR TABLES IN SCHEMA`に対応します。			

次のページに続く

表 107 – 前のページからの続き

Field	Description	Required	Default	Validation
table _ *PublicationTargetTable* _	パブリケーションに追加するテーブルのリストを指定します。PostgreSQL の `FOR TABLE`に 相当します。			

PublicationTargetTable

PublicationTargetTable は、パブリッシュするテーブルです

登場場所:

- *PublicationTargetObject*

Field	Description	Required	Default	Validation
only _boolean_	テーブルのみに制限するか、そのすべての子孫を含めるか			
name _string_	テーブル名	True		
schema _string_	スキーマ名			
columns _string **array**_	公開する列			

RecoveryTarget

RecoveryTarget を使用すると、復旧プロセスが停止する瞬間を構成できます。TargetTLI を除くすべてのターゲットオプションは相互排他的です。

登場場所:

- *BootstrapRecovery*

Field	Description	Required	Default	Validation
backupID _string_	リカバリプロセスを開始するバックアップの ID。 空のデフォルト場合、オペレーターは指定されている場合、targetTime または targetLSN に基づいて、バックアップを自動的に検出します。それ以外の場合は、 使用可能な最新のバックアップを時系列で使用します。			
targetTLI _string_	ターゲットタイムライン「latest」または正の整数			

次のページに続く

表 109 – 前のページからの続き

Field	Description	Required	Default	Validation
targetXID _string_	ターゲットトランザクション ID			
targetName _string_	ターゲット名以前に作成します `pg_create_restore_point`で			
targetLSN _string_	ターゲット LSN ログシーケンス番号			
targetTime _string_	RFC3339 形式または PostgreSQL タイムスタンプ形式のタイムスタンプとしてのターゲット時間。 明示的なタイムゾーンのないタイムスタンプは UTC として解釈されます。			
targetImmediate _boolean_	一貫した状態に到達すると、リカバリーを終了します			
exclusive _boolean_	ターゲットを排他的に設定します。省略した場合、デフォルト false になるため、Postgres では `recovery_target_inclusive`が true になります			

ReplicaClusterConfiguration

ReplicaClusterConfiguration は、レプリカクラスターの構成をカプセル化します

登場場所:

- *ClusterSpec*

Field	Description	Required	Default	Validation
self _string_	このクラスターの名前を自己定義します。 `primary`と比較して、これがプライマリ またはレプリカクラスターであるかどうかを判断するために使用されます			
primary _string_	Primary は、externalClusters で指定された トポロジに基づいて、分散 PostgreSQL クラスターのプライマリとして定義されるクラスターを定義します			
source _string_	レプリケーション元の外部クラスターの名前	True		MinLength: 1

次のページに続く

表 110 – 前のページからの続き

Field	Description	Required	Default	Validation
enabled _boolean_	レプリカモードが有効になっている場合、このクラスターは 既存のクラスターのレプリカになります。レプリカクラスターは、リカバリ オブジェクトストアから、または pg_basebackup を介したストリーミングを介して作成できます。 詳細については、ドキュメントの「レプリカクラスター」ページを参照してください。			
promotionToken _string_	プロモーション要件が満たされているかどうかを確認するために使用される外部クラスターによって生成された降格トークン。			
minApplyDelay _ Duration _	レプリカモードが有効になっている場合、このパラメータにより、システム時間が少なくとも構成された コミット時間を経過した時間である場合にのみトランザクションを再生できます。これは、データ損失エラーを修正する機会を提供します。このパラメーターが設定されている場合、プロモーション トークンは使用できないことに注意してください。			

ReplicationSlotsConfiguration

ReplicationSlotsConfiguration は、レプリケーションスロットの構成をカプセル化します

登場場所:

- *ClusterSpec*

Field	Description	Required	Default	Validation
highAvailability _ *ReplicationSlotsHAConfiguration* _	高可用性構成のレプリケーションスロット			

次のページに続く

表 111 – 前のページからの続き

Field	Description	Required	Default	Validation
updateInterval _integer_	スタンバイは、ローカルレプリケーションスロットのステータスを更新します `updateInterval`秒ごとにデフォルト 30。			Minimum: 1
synchronizeReplicas _ *SynchronizeReplicasConfiguration* _	ユーザー定義の物理レプリケーションスロットの同期を構成します			

ReplicationSlotsHAConfiguration

ReplicationSlotsHAConfiguration は、オペレーターが自動的に管理するレプリケーションスロットの構成をカプセル化して、高可用性 HA を目的としてスタンバイインスタンスとのストリーミングレプリケーション接続を制御します。レプリケーションスロットは、PostgreSQL の機能であり、ストリーミングクライアントこの特定の場合、HA クラスターの一部であるレプリカが切断されたときに、PostgreSQL が WAL ファイルをプライマリに自動的に保持するようにします。

登場場所:

- *ReplicationSlotsConfiguration*

Field	Description	Required	Default	Validation
enabled _boolean_	有効になっている場合デフォルトでは、オペレーターはプライマリインスタンスのレプリケーションスロット を自動的に管理し、HA クラスターの一部であるすべてのスタンバイインスタンスとのストリーミングレプリケーション 接続でそれらを使用します。無効の場合、オペレーターはレプリカとのストリーミング接続のレプリケーションスロットを利用しません。 この機能は、指定されたプライマリからそのカスケードレプリカまで、レプリカクラスターのレプリケーションスロットも制御します。			

次のページに続く

表 112 – 前のページからの続き

Field	Description	Required	Default	Validation
slotPrefix _string_	HA のオペレーターが管理するレプリケーションスロットのプレフィックス。 小文字、数字、アンダースコア文字のみを含めることができます。 これは作成時にのみ設定できます。デフォルトでは`_cnp_`に設定されます。			Pattern: *^[0-9a-z_]*$*
synchronizeLogicalDecoding _boolean_	有効にすると、オペレーターは、高可用性クラスター全体の論理 デコードレプリケーションスロットの同期を自動的に管理します。 次のいずれかの条件が必要です。 - PostgreSQL バージョン 17 以降 - PostgreSQL バージョン < 17 pg_failover_slots 拡張機能が有効になっている			

RoleConfiguration

RoleConfiguration は、追加フィールド Ensure を備えた PostgreSQL ロールの Kubernetes での表現形式であり、データベース内のロールの存在または不在を保証するかどうかを指定します

CREATE ROLE コマンドのデフォルトが適用されます参照 https://www.postgresql.org/docs/current/sql-createrole.html

登場場所:

- *DatabaseRoleSpec*
- *ManagedConfiguration*

Field	Description	Required	Default	Validation
name _string_	ロールの名前	True		
comment _string_	ロールの説明			
ensure _ *EnsureOption* _	ロールが `present`または `absent`であることを確認します-デフォルトは「present」です		present	Enum: [present absent]

次のページに続く

表 113 – 前のページからの続き

Field	Description	Required	Default	Validation
passwordSecret _ LocalObjectReference _	ロールのパスワードが含まれるシークレット存在する場合。 ヌルの場合、パスワードは DisablePassword が設定されていない限り無視されます。 設定されている場合、シークレットは `kubernetes.io/basic-auth`フォーマットに従います そして両方を含む必要があります`username`および`password`フィールド。			
connectionLimit _integer_	ロールがログインできる場合、これはロールが作成できる同時 接続の数を指定します。`-1`デフォルトは制限なしを意味します。		-1	
validUntil _ バックアップからのポイントインタイム リカバリー *PITR* _	ロールのパスワードが有効でなくなる日時。 省略すると、パスワードの有効期限はありませんデフォルト。			
inRoles _string **array**_	このロールが 新しいメンバーとしてすぐに追加される 1 つ以上の既存のロールのリスト。デフォルト空。 リストの変更は、ロールの作成時だけでなく、 `GRANT`および`REVOKE`ステートメントを介して既存のロールに適用されます。			
inherit _boolean_	ロールが、メンバーであるロールの特権を「継承」するかどうか。 デフォルトは`true`です。		true	
disablePassword _boolean_	DisablePassword は、Postgres でロールのパスワードを NULL に設定する必要があることを示します			

次のページに続く

表 113 – 前のページからの続き

Field	Description	Required	Default	Validation
superuser _boolean_	ロールが `superuser`であるかどうか データベース内のすべてのアクセス 制限をオーバーライドできる人 - スーパーユーザーステータスは危険であり、 本当に必要な場合にのみ使用する必要があります。新しいスーパーユーザーを作成するには、自分自身が スーパーユーザーである必要があります。デフォルトは `false`です。			
createdb _boolean_	`true`に設定されている場合、定義されるロールは新しいデータベースを作成できます 。`false`デフォルトを指定すると、ロールがデータベースを作成する 機能が拒否されます。			
createrole _boolean_	ロールが他のロールの作成、変更、ドロップ、コメント セキュリティラベルの変更、およびメンバーシップの付与または取り消しを許可されるかどうか ほかのロール。デフォルトは `false`です。			
login _boolean_	ロールにログインを許可するかどうか。 `login` 属性を持つロールはユーザーと考えられます。この属性のないロール はデータベース特権の管理に役立ちますが、通常の意味でのユーザーではありません 。デフォルトは `false`です。			

次のページに続く

表 113 – 前のページからの続き

Field	Description	Required	Default	Validation
replication _boolean_	ロールがレプリケーションロールであるかどうか。レプリケーションモード物理または論理レプリケーションで サーバーに接続し、次のことができるようにするには、ロールがこの 属性を持つか、スーパーユーザーである必要があります。レプリケーションスロットを作成またはドロップします。 `replication`属性を持つロールは非常に高い特権ロールであり、 レプリケーションに実際に使用されるロールでのみ使用する必要があります。デフォルト は `false`です。			
bypassrls _boolean_	ロールがすべての行レベルセキュリティ RLS ポリシーをバイパスするかどうか。 デフォルトは `false`です。			

RoleStatus

Underlying type: string

RoleStatus は、クラスター内の管理対象ロールのステータスを表します

登場場所:

- *ManagedRoles*

Field	Description
reconciled	RoleStatusReconciled は、DB のロールが仕様と一致することを示します
not-managed	RoleStatusNotManated は、ロールが仕様にないため、管理対象でないことを示します
pending-reconciliation	RoleStatusPendingReconciliation は、仕様のロールが DB の更新/作成が必要であることを示します
reserved	RoleStatusReserved は、これがオペレーターによって予約されたロールの 1 つであることを示します。例 `postgres`

SQLRefs

SQLRefs は、SQL ファイルを含む ConfigMap または Secret への参照を保持します。参照は特定の順序で処理されます。最初にすべてのシークレットが処理され、次にすべての ConfigMap が処理されます。各グループ内での処理順序は、それぞれの配列で指定されたシーケンスに従います。

登場場所:

- *BootstrapInitDB*

Field	Description	Required	Default	Validation
secretRefs _ SecretKeySelector **array**_	SecretRefs は、シークレットへの参照のリストを保持します			
configMapRefs _ ConfigMapKeySelector **array**_	ConfigMapRefs は、ConfigMaps への参照のリストを保持します			

ScheduledBackup

ScheduledBackup は、スケジュールされたバックアップ API のスキーマです

Field	Description	Required	Default	Validation
apiVersion _string_	*postgresql.k8s.enterprisedb.io/v1*	True		
kind _string_	*ScheduledBackup*	True		
metadata _ *EmbeddedObjectMetadata* _	`metadata`のフィールドについては、Kubernetes API ドキュメントを参照してください。	True		
spec _ *ScheduledBackupSpec* _	ScheduledBackup の目的の動作の仕様。 詳細情報 <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>	True		

次のページに続く

表 116 – 前のページからの続き

Field	Description	Required	Default	Validation
status _ *ScheduledBackupStatus* _	ScheduledBackup の最近観察されたステータス。このデータは最新でない可能性があります 。システムによって設定されます。読み取り専用。 詳細情報 <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>			

ScheduledBackupSpec

ScheduledBackupSpec は、 ScheduledBackup の目的の状態を定義します

登場場所:

- *ScheduledBackup*

Field	Description	Required	Default	Validation
suspend _boolean_	このバックアップが一時停止されているかどうか			
immediate _boolean_	最初のバックアップを作成後にすぐに開始する必要があるかどうか			
schedule _string_	スケジュールは、Kubernetes の Cron ジョブで使用される形式に従いません。追加の秒指定子が含まれるため、 <https://pkg.go.dev/github.com/robfig/cron#hdr-CRON_Expression_Format>を参照してください。	True		
cluster _ LocalObjectReference _	バックアップするクラスター	True		

次のページに続く

表 117 – 前のページからの続き

Field	Description	Required	Default	Validation
backupOwnerReference _string_	作成したバックアップリソース内にどの ownerReference を配置するかを示します。 - none 作成したバックアップオブジェクトの所有者リファレンスなしフィールドが導入される前と同じ動作 - self スケジュールされたバックアップオブジェクトを所有者として設定しますバックアップの -クラスタークラスターをバックアップの所有者として設定します 		none	Enum: [none self cluster]
target _ *BackupTarget* _	このバックアップを実行するインスタンスを決定するポリシー。空の場合 デフォルトの `cluster.spec.backup.target`. 使用可能なオプションは空の文字列、`primary`および `prefer-standby`です。 `primary`は常にプライマリインスタンスで実行されます。 `prefer-standby`はバックアップを優先して実行します最新の スタンバイで利用可能な場合。			Enum: [primary prefer-standby]
method _ *BackupMethod* _	使用するバックアップ方法、可能なオプションは `barmanObjectStore`、 `volumeSnapshot`または `plugin`です。デフォルト`barmanObjectStore`		barmanObjectStore	Enum: [barmanObjectStore volumeSnapshot plugin]
pluginConfiguration _ *BackupPluginConfiguration* _	このバックアップを管理するプラグインに渡される構成パラメーター			

次のページに続く

表 117 – 前のページからの続き

Field	Description	Required	Default	Validation
online _boolean_	ボリュームスナップショットを使用したバックアップのデフォルトのタイプが オンライン/ホット *true*、デフォルトかオフライン/コールド`false`かどうか クラスターフィールド '.spec.backup.volumeSnapshot で指定されたデフォルト設定をオーバーライドします。オンライン』			
onlineConfiguration _ *OnlineConfiguration* _	ボリュームスナップショットを使用してオンライン/ホットバックアップを制御する構成パラメーター クラスター '.backup.volumeSnapshot.onlineConfiguration スタンザで指定されたデフォルト設定をオーバーライドします			

ScheduledBackupStatus

ScheduledBackupStatus は、 ScheduledBackup の観察された状態を定義します

登場場所:

- *ScheduledBackup*

Field	Description	Required	Default	Validation
lastCheckTime _ バックアップからのポイント インタイム リカバリー *PITR* _	最新のスケジュール			
lastScheduleTime _ バックアップからのポイント インタイム リカバリー *PITR* _	そのバックアップが最後に正常にスケジュールされたときの情報。			
nextScheduleTime _ バックアップからのポイント インタイム リカバリー *PITR* _	次回バックアップを実行します			

次のページに続く

表 118 – 前のページからの続き

Field	Description	Required	Default	Validation
error _string_	Error は最新の入場検証エラーです			

SchemaSpec

SchemaSpec はデータベースにスキーマを構成します

登場場所:

- *DatabaseSpec*

Field	Description	Required	Default	Validation
name _string_	オブジェクトの名前拡張子、スキーマ、FDW、サーバー	True		
ensure _ *EnsureOption* _	データベースにオブジェクトスキーマなどを存在するかどうかを指定します 。`present`に設定されている場合、オブジェクトは 存在しない場合に作成されます。`absent`に設定している場合、拡張機能/スキーマは 存在する場合は削除されます。		present	Enum: [present absent]
owner _string_	PostgreSQL 内のスキーマを所有するユーザーのロール名。 これは、`CREATE SCHEMA`の`AUTHORIZATION`パラメーターと `ALTER SCHEMA`の `OWNER TO`コマンドにマッピングされます。	True		

SecretVersion

SecretVersion には、シークレット名とそのリソースバージョンが含まれます

登場場所:

- *PgBouncerSecrets*
- *PoolerSecrets*

Field	Description	Required	Default	Validation
name _string_	シークレットの名前			

次のページに続く

表 120 – 前のページからの続き

Field	Description	Required	Default	Validation
version _string_	シークレットのリソースバージョン			

SecretsResourceVersion

SecretsResourceVersion は、オペレーターが管理するシークレットのリソースバージョンです。

登場場所:

- *ClusterStatus*

Field	Description	Required	Default	Validation
superuserSecretVersion _string_	"postgres"ユーザーシークレットのリソースバージョン			
replicationSecretVersion _string_	「streaming_replica」ユーザーシークレットのリソースバージョン			
applicationSecretVersion _string_	"app"ユーザーシークレットのリソースバージョン			
managedRoleSecretVersion _object (keys:string, values:string)_	管理対象ロールシークレットのリソースバージョン			
caSecretVersion _string_	未使用。古いバージョンとの互換性のために保持されています。			
clientCaSecretVersion _string_	PostgreSQL のリソースバージョンクライアントサイドの CA シークレットバージョン			
serverCaSecretVersion _string_	PostgreSQL のリソースバージョンサーバーサイド CA シークレットバージョン			
serverSecretVersion _string_	PostgreSQL のリソースバージョンサーバーサイドシークレットバージョン			
barmanEndpointCA _string_	Barman エンドポイント CA のリソースバージョンが提供されている場合			
externalClusterSecretVersion _object (keys:string, values:string)_	外部クラスターシークレットのリソースバージョン			

次のページに続く

表 121 – 前のページからの続き

Field	Description	Required	Default	Validation
metrics _object (keys:string, values:string)_	メトリックを渡すために使用されるすべてのシークレットのバージョンを含むマップ。 マップキーはシークレット名であり、マップ値はバージョンです			

ServerSpec

ServerSpec は外部データラッパーのサーバーを構成します

登場場所:

- *DatabaseSpec*

Field	Description	Required	Default	Validation
name _string_	オブジェクトの名前拡張子、スキーマ、FDW、サーバー	True		
ensure _ *EnsureOption* _	データベースにオブジェクトスキーマなどを存在するかどうかを指定します 。`present`に設定されている場合、オブジェクトは 存在しない場合に作成されます。`absent`に設定している場合、拡張機能/スキーマは 存在する場合は削除されます。		present	Enum: [present absent]
fdw _string_	外部データラッパー FDW の名前	True		
options _ *OptionSpec* **array**_	Options は、サーバーの構成オプションを指定します キーはオプション名、value はオプション値です。			
usage _ *UsageSpec* **array**_	サーバー上の`USAGE`特権が付与または取り消されるロールのリスト。			

ServiceAccountTemplate

ServiceAccountTemplate には、サービスアカウントを生成するために必要なテンプレートが含まれています

登場場所:

- *ClusterSpec*

Field	Description	Required	Default	Validation
metadata _ EmbeddedObjectMetadata _	`metadata`のフィールドについては、Kubernetes API ドキュメントを参照してください。	True		

ServiceSelectorType

Underlying type: string

ServiceSelectorType は、サービスセレクターを生成するための有効な値を説明します。読み取り/書き込み、読み取り、読み取り専用など、セレクターが適用されるサービスの種類を示します

検証

- 列挙型 [rw r ro]

登場場所:

- *ManagedService*
- *ManagedServices*

Field	Description
rw	ServiceSelectorTypeRW は、読み取り/書き込みサービスを選択します。
r	ServiceSelectorTypeR は読み取りサービスを選択します。
ro	ServiceSelectorTypeRO は読み取り専用サービスを選択します。

ServiceTemplateSpec

ServiceTemplateSpec は、ユーザーがサービス生成のテンプレートを設定できる構造です。

登場場所:

- *ManagedService*
- *PoolerSpec*

Field	Description	Required	Default	Validation
metadata _ EmbeddedObjectMetadata _	`metadata`のフィールドについては、Kubernetes API ドキュメントを参照してください。			

次のページに続く

表 125 – 前のページからの続き

Field	Description	Required	Default	Validation
spec _ ServiceSpec _	サービスの目的の動作の仕様。 詳細情報 <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status>			

ServiceUpdateStrategy

Underlying type: string

ServiceUpdateStrategy は、管理対象サービスへの変更を処理する方法を説明します

検証

- 列挙型 [patch replace]

登場場所:

- *ManagedService*

SnapshotOwnerReference

Underlying type: string

SnapshotOwnerReference は、スナップショットの所有者の参照タイプを定義します。これは、処理されたリソースがどの所有者に関連するかを指定します。

登場場所:

- *VolumeSnapshotConfiguration*

Field	Description
none	SnapshotOwnerReferenceNone は、スナップショットに所有者参照がないことを示します。
backup	SnapshotOwnerReferenceBackup は、スナップショットがバックアップリソースによって所有されていることを示します。
cluster	SnapshotOwnerReferenceCluster は、スナップショットがクラスターリソースによって所有されていることを示します。

SnapshotType

Underlying type: string

SnapshotType は許可されるインポートのタイプです

登場場所:

- *OpenShift* の重要な概念

Field	Description
monolith	MonolithSnapshotType は、モノリスクローンタイポロジを実行することを示します
microservice	MicroserviceSnapshotType は、マイクロサービスクローンタイポロジを実行することを示します

StorageConfiguration

StorageConfiguration は、PVC の作成と調整に使用される構成であり、WAL ボリューム、PGDATA ボリューム、またはテーブルスペースに使用できます

登場場所:

- *ClusterSpec*
- *TablespaceConfiguration*

Field	Description	Required	Default	Validation
storageClass _string_	PVC に使用する StorageClass。利用可能な場合は、PVC テンプレートを評価した後に適用されます。 指定しない場合、生成された PVC は デフォルトのストレージクラスを使用します			
size _string_	ストレージのサイズ。 PVC テンプレートでまだ指定されていない場合は必須。 このフィールドの変更は、作成された PVC に自動的に再適用されます。 サイズを減らすことはできません。			
resizeInUseVolumes _boolean_	既存の PVC のサイズを変更します、デフォルトは true		true	
pvcTemplate _ PersistentVolume-ClaimSpec _	Persistent Volume Claim を生成するために使用されるテンプレート			

Subscription

サブスクリプションは、サブスクリプション API のスキーマです

Field	Description	Required	Default	Validation
apiVersion _string_	*postgresql.k8s.enterprisedb.io/v1*	True		
kind _string_	*Subscription*	True		
metadata _ *EmbeddedObjectMetadata* _	`metadata`のフィールドについては、Kubernetes API ドキュメントを参照してください。	True		
spec _ *SubscriptionSpec* _	NA	True		
status _ *SubscriptionStatus* _	NA	True		

SubscriptionReclaimPolicy

Underlying type: string

SubscriptionReclaimPolicy は、サブスクリプションのサポートが終了したメンテナンスのポリシーを説明します。

登場場所:

- *SubscriptionSpec*

Field	Description
delete	SubscriptionReclaimDelete は、サブスクリプションがそのクレームからリリース されるときに Kubernetes から削除されることを意味します。
retain	SubscriptionReclaimRetain は、サブスクリプションが現在のフェーズで残されて、管理者による手動 再利用ということを意味します。デフォルトポリシーは Retain です。

SubscriptionSpec

SubscriptionSpec は、サブスクリプションの目的の状態を定義します

登場場所:

- 宣言的なパブリケーションおよびサブスクリプションオブジェクトを介した論理レプリケーション

Field	Description	Required	Default	Validation
cluster _ LocalObjectReference _	「サブスクライバー」を識別する PostgreSQL クラスターの名前	True		
name _string_	PostgreSQL 内のサブスクリプションの名前	True		

次のページに続く

表 131 – 前のページからの続き

Field	Description	Required	Default	Validation
dbname _string_	パブリケーションが「サブスクライバー」クラスターにインストールされるデータベースの名前 	True		
parameters _object (keys:string, values:string)_	PostgreSQL の `WITH`句に含まれるサブスクリプションパラメーター `CREATE SUBSCRIPTION`コマンド。ほとんどのパラメーターはサブスクリプションの作成後 変更できません。後で変更しても無視されます。ただし、次の場所に文書化されている限定セットを除きます。 <a href="https://www.postgresql.org/docs/current/sql-altersubscription.html#SQL-ALTERSUBSCRIPTION-PA /current/sql-altersubscription.html# SQL-ALTERSUBSCRIPTION-PARAMS-SET >			
publicationName _string_	「パブリッシャー」の PostgreSQL データベース内のパブリケーションの名前 「パブリッシャー」	True		
publicationDBName _string_	外部 クラスター上のパブリケーションを含むデータベースの名前。デフォルトは外部クラスター定義のものです。			
externalClusterName _string_	パブリケーション「パブリッシャー」を含む外部クラスターの名前	True		
subscriptionReclaimPolicy _ *SubscriptionReclaimPolicy* _	このサブスクリプションのサポートが終了したメンテナンスのポリシー		retain	Enum: [delete retain]

SubscriptionStatus

SubscriptionStatus は、サブスクリプションの観察された状態を定義します

登場場所:

- 宣言的なパブリケーションおよびサブスクリプションオブジェクトを介した論理レプリケーション

Field	Description	Required	Default	Validation
observedGeneration _integer_	同期された最新の 目的の状態を表すシーケンス番号			
applied _boolean_	サブスクリプションが正しく調整された場合、Apply は true です			
message _string_	Message はリコンシリエーション出力メッセージです			

SwitchReplicaClusterStatus

SwitchReplicaClusterStatus には、クラスターからレプリカクラスターへのスイッチに関するすべてのステータスが含まれています

登場場所:

- *ClusterStatus*

Field	Description	Required	Default	Validation
inProgress _boolean_	InProgress は、クラスターをレプリカクラスターに切り替える進行中の手順があるかどうかを示します。			

SyncReplicaElectionConstraints

SyncReplicaElectionConstraints には、同期レプリカ選択のコンストレインが含まれています。

アンチアフィニティパラメーターの場合、すべてのラベル値が一致する場合、2 つのインスタンスは同じ場所にあるとみなされます。

将来的には、名前による同期レプリカ選択制限がサポートされる予定です。

登場場所:

- *PostgresConfiguration*

Field	Description	Required	Default	Validation
nodeLabelsAntiAffinity _string_ **array_**	抽出および比較して、ポッドが同じトポロジに存在するかどうかを評価するノードラベル値のリスト			
enabled _boolean_	このフラグは、同期レプリカのコンストレインを有効にします		True	

SynchronizeReplicasConfiguration

SynchronizeReplicasConfiguration には、ユーザー定義の物理レプリケーションスロットの同期のための構成が含まれています

登場場所:

- *ReplicationSlotsConfiguration*

Field	Description	Required	Default	Validation
enabled _boolean_	true に設定すると、プライマリにあるすべてのレプリケーションスロットが各スタンバイで同期されます	True		
excludePatterns _string **array**_	除外するレプリケーションスロットの名前と一致する正規表現パターンのリストデフォルトでは空			

SynchronousReplicaConfiguration

SynchronousReplicaConfiguration には、PostgreSQL 同期レプリケーション機能の構成が含まれています。重要現時点では、 `.spec.minSyncReplicas` および`.spec.maxSyncReplicas` も考慮する必要があります。

登場場所:

- *PostgresConfiguration*

Field	Description	Required	Default	Validation
method _ *SynchronousReplicaConfigurationMethod* _	リストされた サーバーから同期レプリケーションスタンバイを選択する方法。 'any' クォーラムベースの同期レプリケーションまたは 'first' プライオリティベースの同期レプリケーションを値として受け入れます。	True		Enum: [any first]
number _integer_	トランザクションが応答を待機する必要がある同期スタンバイサーバーの数を指定します。	True		
maxStandbyNamesFromCluster _integer_	PostgreSQL の `synchronous_standby_names`オプションに 自動的に含めることができるローカルクラスターポッドの最大数を指定します。			

次のページに続く

表 136 – 前のページからの続き

Field	Description	Required	Default	Validation
standbyNamesPre _string **array**_	ローカルクラスターポッドの前に `synchronous_standby_names`に追加されるアプリケーション名のユーザー定義のリスト この順序は 優先度ベースの同期レプリケーションの場合にのみ役立ちます。			
standbyNamesPost _string **array**_	ローカルクラスターポッドの後に `synchronous_standby_names`に追加されるアプリケーション名のユーザー定義のリストこの順序は 優先度ベースの同期レプリケーションの場合にのみ役立ちます。			
dataDurability _ *DataDurabilityLevel* _	「必須」に設定されている場合、データの耐久性は厳密に適用されます。同期コミット設定`on`、*remote_write*、または`remote_apply`を使用した書き込み操作 は、正常なレプリカが不十分な場合にブロックされ、データの永続性が保証されます。 「推奨」に設定されている場合、データの耐久性は維持されます正常なレプリカ は利用できますが、レプリカが利用できなくなった場合、必要なインスタンスの数は動的に調整されます 。この設定は、厳密な耐久性の適用を緩和して オペレーションの継続性を可能にします。この設定は、 `standbyNamesPre`と`standbyNamesPost`の両方が設定されていない空の場合にのみ適用されます。			Enum: [required preferred]

次のページに続く

表 136 – 前のページからの続き

Field	Description	Required	Default	Validation
failoverQuorum _boolean_	FailoverQuorum は、フェールオーバーの前にクォーラムベースのチェックを有効にし、CloudNativePG クラスター管理の PostgreSQL クラスターの EDB Postgres AI でのフェールオーバーイベント中の データの耐久性と安全性を向上させます。			

SynchronousReplicaConfigurationMethod

Underlying type: string

SynchronousReplicaConfigurationMethod は、クォーラムベースのレプリケーションまたは優先リストを使用するかどうかを構成します

登場場所:

- *SynchronousReplicaConfiguration*

TDEConfiguration

TDEConfiguration には、透過的データ暗号化構成が含まれています

登場場所:

- *EPASConfiguration*

Field	Description	Required	Default	Validation
enabled _boolean_	TDE を有効にする場合は True			
secretKeyRef _ SecretKeySelector _	暗号化キーを含むシークレットへの参照			
wrapCommand _ SecretKeySelector _	WrapCommand は、ユーザーが提供する encrypt コマンドです			
unwrapCommand _ SecretKeySelector _	UnwrapCommand は、ユーザーが提供する復号コマンドです			
passphraseCommand _ SecretKeySelector _	PassphraseCommand は、暗号化と復号化のために OpenSSL コマンドに渡されます パスフレーズを取得するために実行されるコマンド			

TablespaceConfiguration

TablespaceConfiguration はテーブルスペースの構成であり、テーブルスペースのストレージ仕様が含まれます

登場場所:

- *ClusterSpec*

Field	Description	Required	Default	Validation
name _string_	テーブルスペースの名前	True		
storage _ *StorageConfiguration* _	テーブルスペースのストレージ構成	True		
owner _ *DatabaseRoleRef* _	所有者は、テーブルスペースを所有する PostgreSQL ユーザーです			
temporary _boolean_	true に設定すると、テーブルスペースが PostgreSQL の `temp_tablespaces` エントリとして追加され、一時 データベースオブジェクト、またはその他の一時ファイルを自動的に収容するために利用できます。 *temp_tablespaces* GUC の詳細については、PostgreSQL の ドキュメントを参照してください。		false	

TablespaceState

TablespaceState は、クラスター内のテーブルスペースの状態を表します

登場場所:

- *ClusterStatus*

Field	Description	Required	Default	Validation
name _string_	Name はテーブルスペースの名前です	True		
owner _string_	所有者は、テーブルスペースを所有する PostgreSQL ユーザーです			
state _ *TablespaceStatus* _	状態は最新の調整状態です	True		
error _string_	Error はリコンシリエーションエラーである場合			

TablespaceStatus

Underlying type: string

TablespaceStatus は、クラスター内のテーブルスペースのステータスを表します

登場場所:

- *TablespaceState*

Field	Description
reconciled	TablespaceStatusReconciled は、 DB のテーブルスペースが仕様と一致することを示します
pending	TablespaceStatusPendingReconciliation は、仕様のテーブルスペースが DB での作成が必要であることを示しています

Topology

トポロジにはクラスタートポロジが含まれます

登場場所:

- *ClusterStatus*

Field	Description	Required	Default	Validation
instances _object (keys: *PodName* , values: *PodTopologyLabels*)_	インスタンスには、インスタンスのポッドトポロジが含まれます			
nodesUsed _integer_	Nodes Used は、インスタンスに対応する個別のノードの数を表します。 値が'1' は、すべてのインスタンスが単一のノードでホストされることを示し、 高可用性 HA が存在しないことを示します。理想的には、この値は Postgres HA クラスターのインスタンスの数と同じであり、コンピューティング側でシェアードナッシングアーキテクチャであることを意味します。			

次のページに続く

表 141 – 前のページからの続き

Field	Description	Required	Default	Validation
successfullyExtracted _boolean_	SuccessquietlyExtracted は、トポロジデータが抽出されたかどうかを示します。障害が発生した場合に備えて、同期レプリカ選択でフォールバック動作を制定すると役立ちます 			

UsageSpec

UsageSpec は、外部データラッパーの使用法を構成します

登場場所:

- *FDWSpec*
- *ServerSpec*

Field	Description	Required	Default	Validation
name _string_	使用の名前	True		
type _ *UsageSpecType* _	使用の種類		grant	Enum: [grant revoke]

UsageSpecType

Underlying type: string

UsageSpecType は、 `Database` オブジェクトの `usage` フィールドで指定された使用のタイプを説明します。

登場場所:

- *UsageSpec*

Field	Description
grant	GrantUsageSpecType は、使用権限の付与を示します。 デフォルトの使用権限は grant です。
revoke	RevokeUsageSpecType は、使用権限の取り消しを示します。

VolumeSnapshotConfiguration

VolumeSnapshotConfiguration は、スナップショットバックアップを実行するための構成を表します。

登場場所:

- *BackupConfiguration*

Field	Description	Required	Default	Validation
labels _object (keys:string, values:string)_	ラベルは、 .metadata.labels スナップショットリソースに追加されるキーと値のペアです。			
annotations _object (keys:string, values:string)_	.metadata.annotations スナップショットリソースに追加される注釈のキーと値のペア。			
className _string_	ClassName は、PG_DATA Persistent VolumeClaim に使用されるスナップショットクラスを指定します。 これは、特定のクラスが存在しない場合、他のタイプのデフォルトクラスです。			
walClassName _string_	WalClassName は、PG_WAL Persistent VolumeClaim に使用されるスナップショットクラスを指定します。			
tablespaceClassName _object (keys:string, values:string)_	TablespaceClassName は、テーブルスペースに使用されるスナップショットクラスを指定します。 デフォルトで PGDATA スナップショットクラスが設定されている場合			
snapshotOwnerReference _ *SnapshotOwnerReference* _	SnapshotOwnerReference は、スナップショットが持つ必要がある所有者参照のタイプを示します		none	Enum: [none cluster backup]
online _boolean_	ボリュームスナップショットを使用したバックアップのデフォルトのタイプが オンライン/ホット *true*、default またはオフライン/コールド *false*		true	
onlineConfiguration _ *OnlineConfiguration* _	ボリュームスナップショットを使用してオンライン/ホットバックアップを制御する構成パラメーター		{ immediateCheckpoint:false waitForArchive:true }	

7.5 Backup and Recovery

Appendix B - Backup on object stores および **オブジェクトストアからのリカバリー** は、2 つの別個のセクションにあります。

7.6 CNCF Projects Integrations

7.6.1 External Secrets

External Secrets は CNCF サンドボックスプロジェクトであり、TAG Security のスポンサーのもとで 2022 年に受け入れられました。

について

外部シークレットオペレーター ESO は、シークレットのストレージを Kubernetes 自分自身から分離することにより、シークレット管理を強化する Kubernetes オペレーターです。これにより、外部シークレット管理システムとネイティブ Kubernetes `Secret` リソース間のシームレスな同期が可能になります。

ESO は、以下を含む幅広いバックエンドをサポートしています。

- HashiCorp Vault
- AWS Secrets Manager
- Google Secret Manager
- Azure Key Vault
- IBM Cloud Secrets Manager

…その他にもたくさん。サポートされているプロバイダーの完全な最新のリストについては、

official External Secrets documentation を参照してください。

CloudNativePG クラスターの PostgreSQL および EDB Postgres AI との統合

PostgreSQL データベースに関しては、外部シークレットは、2 つの主要なユースケースで CloudNativePG クラスターの EDB Postgres AI とシームレスに統合します。

- **自動パスワード管理** ESO は、Kubernetes `Secret` リソースに保存されているデータベースユーザーパスワードの自動生成とローテーションを処理でき、クラスター内で実行されているアプリケーションが常に最新の資格情報にアクセスできるようにします。
- **クロスプラットフォームのシークレットアクセス** `SecretStore` リソースを介して外部キー管理サービス KMS とこれらのパスワードの透過的な同期が有効になります。これにより、Kubernetes クラスターの外部のアプリケーションと開発者 Kubernetes シークレットにアクセスできない場合があり、外部 KMS からデータベース資格情報を直接取得できます。

例 外部シークレットを使用した自動パスワード管理

quickstart guide から `cluster-example` Postgres クラスターで `app` ユーザーのパスワードを 24 時間ごとに自動的にローテーションする方法を説明します。

デフォルトでは、CloudNativePG クラスターの EDB Postgres AI は、`cluster-example-app` という名前の Kubernetes `Secret` を生成および管理します。これには、`cluster-example` クラスターの `app` ユーザーの資格情報が含まれています。これについては、 *“Connecting from an application” section* を参照してください。

外部シークレットを使用する場合、目標は次のことです。

1. パスワードの生成方法を指定する Password ジェネレーターを定義します。
2. password および pgpass フィールドのみを更新することにより、cluster-example-app シークレットの同期を維持する ExternalSecret リソースを作成します。

パスワードジェネレーターの作成

次の例では、 Password を作成します

default 名前空間の pg-password-generator という名前のリソース。ニーズに合わせて名前とプロパティをカスタマイズできます。

```
apiVersion: generators.external-secrets.io/v1alpha1
kind: Password
metadata:
  name: pg-password-generator
spec:
  length: 42
  digits: 5
  symbols: 5
  symbolCharacters: "-_$@"
  noUpper: false
  allowRepeat: true
```

この仕様は、生成されるパスワードの長さ、数字、記号、大文字の包含などの特性を定義します。

外部シークレットの作成

次の例では、 時間ごとにパスワードを更新する cluster-example-app-secret という名前の ExternalSecret リソースを作成します。 Merge ポリシーを使用して、cluster-example-app シークレットの指定されたフィールド password 、pgpass 、jdbc-uri 、および uri のみを更新します。

```
apiVersion: external-secrets.io/v1
kind: ExternalSecret
metadata:
  name: cluster-example-app-secret
spec:
  refreshInterval: "24h"
  target:
    name: cluster-example-app
    creationPolicy: Merge
    template:
      metadata:
        labels:
          k8s.enterprisedb.io/reload: "true"
      data:
        password: "{{ .password }}"
```

(次のページに続く)

(前のページからの続き)

```
      pgpass: "cluster-example-rw:5432:app:app:{{ .password }}"
      jdbc-uri: "jdbc:postgresql://cluster-example-rw.default:5432/app?password={{ .
↪password }}&user=app"
      uri: "postgresql://app:{{ .password }}@cluster-example-rw.default:5432/app"
  dataFrom:
    - sourceRef:
        generatorRef:
          apiVersion: generators.external-secrets.io/v1alpha1
          kind: Password
          name: pg-password-generator
```

ラベル `k8s.enterprisedb.io/reload: "true"` は、シークレットが変更されたときに、CloudNativePG クラスターの EDB Postgres AI がデータベースのユーザーパスワードのリロードをトリガーします。

構成の検証

`ExternalSecret` が正しく同期していることを確認するには

```
kubectl get es cluster-example-app-secret
```

リアルタイムで更新されるパスワードを観察するには、一時的に `refreshInterval` を `30s` に減らし、次のコマンドを繰り返し実行します。

```
kubectl get secret cluster-example-app \
  -o jsonpath="{.data.password}" | base64 -d
```

30 秒ごとにパスワードが変更され、ローテーションが正しく機能していることを確認します。

他にもあります

上記の例では、CloudNativePG クラスターの EDB Postgres AI によって作成されたデフォルトの `cluster-example-app` シークレットに焦点を当てていますが、同じアプローチを拡張して、パスワードを定期的にローテーションするように作成するカスタムシークレットまたは PostgreSQL ユーザーを管理できます。

例外部 KMS との統合

CNCF エコシステムで最も広く使用されているキー管理サービス KMS プロバイダーの 1 つが HashiCorp Vault です。Vault はビジネスソースライセンス BUSL に基づいてライセンスされていますが、完全な互換性がありアクティブにメンテナンスされているオープンソースの代替品 OpenBao が利用できます。 OpenBao は、HashCorp Vault と同じインターフェイスをすべてサポートしているため、真のドロップイン代替品となります。

この例では、CloudNativePG クラスター、外部シークレットオペレーター、および HashCorp Vault の EDB Postgres AI を統合して、PostgreSQL パスワードを自動的にローテーションし、Vault に安全に保存する方法を示します。

前の例から続けて、必要な SecretStore および PushSecret リソースを作成して、Vault との統合を完了します。

SecretStore の作成

この例では、 HashiCorp Vault が http://vault.vault.svc:8200 の名前空間内からアクセスでき、vault-token という名前の Kubernetes Secret が同じ名前空間に存在し、Vault での認証に使用されるトークンを含むことを前提としています。

```
apiVersion: external-secrets.io/v1
kind: SecretStore
metadata:
  name: vault-backend
spec:
  provider:
    vault:
      server: "http://vault.vault.svc:8200"
      path: "secrets"
      # Specifies the Vault KV secret engine version ("v1" or "v2").
      # Defaults to "v2" if not set.
      version: "v2"
      auth:
        # References a Kubernetes Secret that contains the Vault token.
        # See: https://www.vaultproject.io/docs/auth/token
        tokenSecretRef:
          name: "vault-token"
          key: "token"
- --
apiVersion: v1
kind: Secret
metadata:
  name: vault-token
data:
  token: aHZzLioqKioqKio= # hvs. ****  ***
```

この構成は、 vault-backend という名前の SecretStore リソースを作成します。

External Secrets Operator documentation を参照してください

サポートされている認証メカニズムの完全なリストについては、

> **注釈**
>
> HashiCorp Vault は、バージョン `v2` の `secrets` パスで KV シークレットエンジンを有効にする必要があります。 Vault インスタンスが別のパスまたはバージョンを使用している場合、それに応じて `path` および `version` フィールドを更新してください。

PushSecret の作成

PushSecret リソースは、Kubernetes Secret を HashCorp Vault にプッシュするために使用されます。この単純化された例では、サンプルクラスター cluster-example の app ユーザーの資格情報をプッシュします。

PushSecret の構成の詳細については、

External Secrets Operator documentation を参照してください。

```
apiVersion: external-secrets.io/v1alpha1
kind: PushSecret
metadata:
  name: pushsecret-example
spec:
  deletionPolicy: Delete
  refreshInterval: 24h
  secretStoreRefs:
    - name: vault-backend
      kind: SecretStore
  selector:
    secret:
      name: cluster-example-app
  data:
    - match:
        remoteRef:
          remoteKey: cluster-example-app
```

この例では、PushSecret リソースは、cluster-example-app という名前の Kubernetes Secret を HashCorp Vault にプッシュするように外部シークレットオペレーターに指示します。前の例から。remoteKey は、vault-backend という名前の SecretStore を使用して、シークレットを Vault に保存する名前を定義します。

構成の検証

PushSecret が正常に機能していることを確認するには、HashiCorp Vault UI に移動します。パス secrets の kv シークレットエンジンで、上記で定義した remoteKey に対応する cluster-example-app という名前のシークレットを見つけます。

7.6.2 Cilium

について

Cilium は、2021 年にインキュベートプロジェクトとして受け入れられ、2023 年に卒業した CNCF 卒業プロジェクトです。もともと Isovalent によって作成されました。これは、クラウドネイティブ環境のための高度なネットワーキング、セキュリティ、およびオブザーバビリティのソリューションであり、eBPF テクノロジーに基づいて構築されています。Cilium は、Linux カーネルに eBPF プログラムを動的に注入することにより、Kubernetes クラスターのネットワークトラフィックを管理し、低遅延、高性能通信を有効にし、きめの細かいセキュリティポリシーを適用します。

Cilium の主な機能

- 詳細な L3-L7 セキュリティポリシーによるきめの細かいネットワークトラフィック制御
- eBPF を介した効率的なカーネルレベルのトラフィック管理
- サービスメッシュ統合 Cilium Service Mesh
- Kubernetes NetworkPolicy と CiliumNetworkPolicy の両方のサポート
- ビルトインの可観測性とハッブルによるモニタリング

環境に Cilium をインストールするには、ドキュメントの指示に従ってください。https://docs.cilium.io/en/stable/gettingstarted/k8s-install-default/

CloudNativePG クラスターおよび Cilium の EDB Postgres AI を使用したポッド間のネットワークセキュリティ

Kubernetes のデフォルトの動作は、クラスターネットワーク内の 2 つのポッド間のトラフィックを許可することです。Cilium は、`CiliumNetworkPolicy` リソースを使用して、高度な L3 / L4 ネットワークセキュリティを提供します。これにより、Kubernetes クラスター内のポッド間のネットワークトラフィックのきめの細かい制御が可能になります。これは、アプリケーションのワークロードとバックエンドサービス間の通信を保護するために特に役立ちます。

次の例では、Cilium を使用して、入力トラフィックを承認されたポッドのみに制限することにより、CloudNativePG Cluster PostgreSQL インスタンスの EDB Postgres AI を保護する方法を示します。

Cilium のデフォルト拒否動作

デフォルトでは、Cilium は明示的に構成されていない限り、すべてのトラフィックを拒否しません**。ポリシーが名前空間に存在すると、デフォルトによる拒否モデルを使用する Kubernetes NetworkPolicy とは対照的に、Cilium はデフォルトの拒否動作をより柔軟に制御します。

Cilium でデフォルトの拒否ポスチャを適用するには、特に許可されない限り、ポッドのセットへのすべてのトラフィックを拒否するポリシーを明示的に作成する必要があります。これは一般に、**空の ``ingress`` セクション** を `endpointSelector` と組み合わせて使用するか、より広範な適用のために Cilium エージェントレベルで **``--enable-default-deny``** を有効にすることにより実現されます。

デフォルトの拒否ポリシーの最小限の例

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: default-deny
  namespace: default
spec:
  description: "Default deny all ingress traffic to all Pods in this namespace"
  endpointSelector: {}
  ingress: []
```

CloudNativePG クラスターオペレーターの EDB Postgres AI と Cilium ネットワークポリシーを機能させる

Cilium かどうかにかかわらず、ネットワークポリシーを使用する場合、最初のステップは、オペレーターがターゲット名前空間のポッドに到達できることを確認することです。オペレーターはポッドでチェックとアクションを実行できる必要があり、これらのアクションの 1 つでは、ポッドのポート 8000 にアクセスして、内部で実行されている PostgreSQL インスタンスの現在のステータスを取得する必要があるため、これは重要です。

次の CiliumNetworkPolicy により、オペレーターはターゲット default 名前空間のポッドにアクセスできます。

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: postgresql-operator-operator-access
  namespace: default
spec:
  description: "Allow EDB Postgres AI for CloudNativePG Cluster operator access to␣
↪any pod in the target namespace"
  endpointSelector: {}
  ingress:
    - fromEndpoints:
        - matchLabels:
            io.kubernetes.pod.namespace: postgresql-operator-system
      toPorts:
        - ports:
            - port: "8000"
              protocol: TCP
```

クラスターポッド間のアクセスの許可

デフォルトポリシーは「deny all」であるため、同じ名前空間内のクラスターポッド間のアクセスを明示的に許可する必要があります。必要なイングレスルールを追加することにより、以前のポリシーを改善します。

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: cnp-cluster-internal-access
  namespace: default
spec:
  description: "Allow EDB Postgres AI for CloudNativePG Cluster operator access and␣
↪connection between pods in the same namespace"
  endpointSelector: {}
  ingress:
    - fromEndpoints:
        - matchLabels:
```

(次のページに続く)

(前のページからの続き)

```
            io.kubernetes.pod.namespace: postgresql-operator-system
        - matchLabels:
            io.kubernetes.pod.namespace: default
            k8s.enterprisedb.io/cluster: cluster-example
      toPorts:
        - ports:
            - port: "8000"
              protocol: TCP
            - port: "5432"
              protocol: TCP
```

このポリシーは、postgresql-operator-system ポッドと、cluster-example にも属する default 名前空間ポッドからのアクセスを許可します。 matchLabels セレクターでは、ポッドにリストされたラベルの完全なセットが必要です。ラベルが 1 つだけない場合は、ポッドが一致しないことを意味します。

Cilium を使用して PostgreSQL へのアクセスを制限する

この例では、 default 名前空間の role=backend というラベルのポッドのみが cluster-example という名前の PostgreSQL クラスターに接続することを許可する CiliumNetworkPolicy を定義します。他のすべての入力トラフィックはデフォルトでブロックされます。

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: postgres-access-backend-label
  namespace: default
spec:
  description: "Allow PostgreSQL access on port 5432 from Pods with role=backend"
  endpointSelector:
    matchLabels:
      k8s.enterprisedb.io/cluster: cluster-example
  ingress:
    - fromEndpoints:
        - matchLabels:
            role: backend
      toPorts:
        - ports:
            - port: "5432"
              protocol: TCP
```

この CiliumNetworkPolicy は、 role=backend でラベル付けられたポッドのみが default 名前空間のポート 5432 を介して CloudNativePG クラスターの EDB Postgres AI によって管理される PostgreSQL インスタンスにアクセスできるようにします。

次のポリシーでは、名前空間のラベル role=backend を持つポッドからのみ、cluster-example という名前の PostgreSQL クラスターのポート 5432 への入力トラフィックを許可する方法を示します。

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: postgres-access-backend-any-ns
  namespace: default
spec:
  description: "Allow PostgreSQL access on port 5432 from Pods with role=backend in␣
↪any namespace"
  endpointSelector:
    matchLabels:
      k8s.enterprisedb.io/cluster: cluster-example
  ingress:
    - fromEndpoints:
        - labelSelector:
            matchLabels:
              role: backend
            matchExpressions:
              - key: io.kubernetes.pod.namespace
                operator: Exists
      toPorts:
        - ports:
            - port: "5432"
              protocol: TCP
```

次の例では、backend 名前空間のポッドから cluster-example クラスター default 名前空間にありますのポート 5432 への入力トラフィックを許可します。

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: postgres-access-backend-namespace
  namespace: default
spec:
  description: "Allow PostgreSQL access on port 5432 from any Pods in the backend␣
↪namespace"
  endpointSelector:
    matchLabels:
      k8s.enterprisedb.io/cluster: cluster-example
  ingress:
    - fromEndpoints:
        - matchLabels:
            io.kubernetes.pod.namespace: backend
      toPorts:
        - ports:
            - port: "5432"
              protocol: TCP
```

Cilium の L3 / L4 ポリシーモデルを使用して、 backend 名前空間のアプリケーションポッドからのみクラスターポッドへの入力トラフィックを明示的に許可する CiliumNetworkPolicy を定義します。追加ポリシーで明示的に許可されない限り、他のすべてのトラフィックは暗黙的に拒否されます。

次の例では、Kubernetes クラスター内の任意のソースから cluster-example クラスター default 名前空間にありますのポート 5432 への入力トラフィックを許可します。

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: postgres-access-cluster-wide
  namespace: default
spec:
  description: "Allow ingress traffic to port 5432 of the cluster-example from any␣
↪pods within the Kubernetes cluster"
  endpointSelector:
    matchLabels:
      k8s.enterprisedb.io/cluster: cluster-example
  ingress:
    - fromEntities:
        - cluster
      toPorts:
        - ports:
            - port: "5432"
              protocol: TCP
```

Cilium ネットワークポリシーの作成と検証を簡素化するビジュアルでインタラクティブツール editor.networkpolicy.io の使用を検討できます。視覚的な方法で表示することにより、構成の間違いを回避し、交通ルールをより明確に理解するのに特に役立ちます。

これらのポリシーにより、PostgreSQL のベースラインのアクセス制御が確立されました。Cilium のポリシー言語を使用して追加の出力または監査ルールを階層化したり、Envoy を使用して L7 適用に拡張したりできます。

第8章　APPENDIX

8.1 Appendix A - Backup on volume snapshots

スナップショット機能を提供します。

CloudNativePG クラスターの EDB Postgres AI は、完全に宣言的な方法で、バックアップとリカバリー操作の両方に Kubernetes ネイティブボリュームスナップショット API を直接利用するデータベースオペレーターの最初の既知のケースの 1 つです。

8.1.1 標準のボリュームスナップショットについて

ボリュームスナップショットは Kubernetes 1.12 (2018) as alpha で最初に導入され、beta in 1.17 (2019) 、および moved to GA in 1.20 (2020) に昇格しました。安定して広く利用可能な標準になり、3 つのカスタムリソース定義 `VolumeSnapshot` 、`VolumeSnapshotContent` 、`VolumeSnapshotClass` を提供します。

この Kubernetes 機能は、次の汎用インターフェイスを定義します。

- PVC から始まる新しいボリュームスナップショットの作成
- 既存のスナップショットの削除
- スナップショットからの新しいボリュームの作成

Kubernetes は、実際の実装を基になる CSI ドライバーに委任しますそれらのすべてがボリュームスナップショットをサポートしているわけではありません。通常、ボリュームスナップショットを提供するストレージクラスは、**アプリケーションの透過的な方法でのインクリメンタルおよび差分ブロックレベルのバックアップ**をサポートします。これにより、スナップショットのクラスター間の可用性を含む、スタックの複雑さと独立した管理を委任できます。

8.1.2 要件

ボリュームスナップショットが CloudNativePG クラスタークラスターの EDB Postgres AI と連携するには、PostgreSQL ボリュームつまり `storage` および `walStorage` セクションを動的にプロビジョニングするために使用される各ストレージクラスがボリュームスナップショットをサポートしていることを確認する必要があります。

手順はストレージクラスごとに異なるため、Kubernetes システムに展開した特定のストレージクラスと関連する CSI ドライバーのドキュメントを参照してください。

通常、それは VolumeSnapshotClass です

特定のストレージクラスの永続ボリュームからスナップショットを取得し、`VolumeSnapshot` および `VolumeSnapshotContent` リソースとして管理できることを保証します。

8.1.3 ボリュームスナップショットバックアップを構成する方法

CloudNativePG クラスターの EDB Postgres AI を使用すると、backup.volumeSnapshot スタンザを介したボリュームスナップショットバックアップ用に特定の Postgres クラスターを構成できます。

> **注釈**
>
> *VolumeSnapshotConfiguration* を参照してください。

オプションの完全なリストについては、API リファレンスを参照してください。

ボリュームスナップショットを使用した一般的な例 PGDATA と WAL が同じストレージクラスを共有すると仮定した場合、次のとおりです。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: snapshot-cluster
spec:
  instances: 3

  storage:
    storageClass: @STORAGE_CLASS@
    size: 10Gi
  walStorage:
    storageClass: @STORAGE_CLASS@
    size: 10Gi

  backup:
    # Volume snapshot backups
    volumeSnapshot:
       className: @VOLUME_SNAPSHOT_CLASS_NAME@

  plugins:
  - name: barman-cloud.cloudnative-pg.io
    isWALArchiver: true
    parameters:
      barmanObjectName: @OBJECTSTORE_NAME@
```

ご覧のとおり、 backup セクションには、 volumeSnapshot スタンザボリュームスナップショットの物理ベースバックアップの制御と plugins スタンザ *PostgreSQL WAL アーカイブ* の制御 の両方が含まれています。

> **注釈**
>
> *plugin* を定義すると、ボリュームスナップショットとプラグインバックアップ戦略の両方を同時に使用して物理的バックアップを取ることができます。

volumeSnapshot.className オプションを使用すると、PostgreSQL クラスターで定義したすべてのストレージボリュームに使用されるデフォルトの VolumeSnapshotClass オブジェクトを参照できます。

> **注釈**
>
> *PGDATA* および WAL ファイルに別のストレージクラスを使用している場合、 *walClassName* オプションデフォルトで `className` と同じ値を介してそのボリュームに別個の `VolumeSnapshotClass` を指定できます。

ボリュームスナップショットバックアップ用にクラスターを定義したら、このようなバックアップを定期的に要求する ScheduledBackup リソースを定義する必要があります。

8.1.4 ホットおよびコールドバックアップ

> **警告**
>
> *backup document* で述べたように、プライマリをターゲットとするように明示的に設定されたコールドスナップショットは、バックアップ中にプライマリがフェンスされ、この期間中のクラスターは読み取り専用になります。安全のため、フェンスされたインスタンスが既に含まれているクラスターでは、コールドスナップショットは拒否されます。

デフォルトでは、CloudNativePG クラスターの EDB Postgres AI は、PostgreSQL defaults of the low-level API for base backups を使用して、ボリュームスナップショットのオンライン/ホットバックアップを要求します。

- バックアップ手順の開始時に、即時チェックポイントを要求しません
- バックアップ手順を終了するときに、WAL アーカイバーがバックアップの最後のセグメントをアーカイブするのを待機します

Cluster リソースの.spec.backup.volumeSnapshot スタンザの次のオプションを使用して、デフォルトの動作を明示的に変更できます。

- online true デフォルトまたは false を値として受け入れます
- onlineConfiguration.immediateCheckpoint バックアップ手順を開始する前に、即時チェックポイントを要求するかどうか。技術的には、PostgreSQL の pg_backup_start /pg_start_backup() ファンクションに渡す fast 引数に対応し、true デフォルトまたは false を受け入れます
- onlineConfiguration.waitForArchive アーカイバーがバックアップの最後のセグメントを処理するまで待機するかどうか。技術的には、PostgreSQL の pg_backup_stop /pg_stop_backup() ファンクションに渡す wait_for_archive 引数に対応し、true デフォルトまたは false を受け入れます

デフォルトでコールドバックアップを取得するように Postgres クラスターのデフォルト動作を変更する場合、次のようにマニフェストに online: false オプションを追加するだけです。

```
#  ...

backup:
```

(次のページに続く)

(前のページからの続き)

```
  volumeSnapshot:
    online: false
    # ...
```

代わりに、デフォルトの動作として即時チェックポイントを要求する場合、このセクションを追加できます。

```
#  ...

backup:
  volumeSnapshot:
    online: true
    onlineConfiguration:
      immediateCheckpoint: true
    # ...
```

デフォルト動作のオーバーライド

Backup または ScheduledBackup オブジェクトの online および必要に応じて onlineConfiguration に別の値を設定することにより、クラスターリソースで定義されたデフォルトの動作を変更できます。

たとえば、オンデマンドのコールドバックアップを発行する場合、.spec.online: false を使用して Backup オブジェクトを作成できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Backup
metadata:
  name: snapshot-cluster-cold-backup-example
spec:
  cluster:
    name: snapshot-cluster
  method: volumeSnapshot
  online: false
```

同様に、 ScheduledBackup の場合

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: ScheduledBackup
metadata:
  name: snapshot-cluster-cold-backup-example
spec:
  schedule: "0 0 0  **  *"
  backupOwnerReference: self
  cluster:
    name: snapshot-cluster
  method: volumeSnapshot
  online: false
```

8.1.5 ボリュームスナップショットオブジェクトの永続化

デフォルトでは、CloudNativePG クラスター用に EDB Postgres AI によって作成された VolumeSnapshot オブジェクトは、それらを生成した Backup オブジェクトまたはそれらが参照する Cluster を削除した後も保持されます。このような動作は、次の値を受け入れる.spec.backup.volumeSnapshot.snapshotOwnerReference オプションによって制御されます。

- none 所有権が設定されていません。Backup および/または Cluster リソースが削除された後も VolumeSnapshot オブジェクトが持続することを意味します
- backup VolumeSnapshot オブジェクトは、それを生成した Backup リソースが所有しており、バックアップオブジェクトが削除されると、ボリュームスナップショットも削除されます
- cluster VolumeSnapshot オブジェクトは、バックアップされた Cluster リソースが所有しており、Postgres クラスターが削除されると、ボリュームスナップショットも削除されます

VolumeSnapshot が削除される場合、 VolumeSnapshotContent で指定された deletionPolicy が評価されます。

- Retain に設定されている場合、VolumeSnapshotContent オブジェクトは保持されます
- Delete に設定されている場合、VolumeSnapshotContent オブジェクトも削除されます

> **警告**
>
> *VolumeSnapshotContent* オブジェクトは、バックアップと参照するクラスターに関するすべての情報を保持していません *VolumeSnapshot* オブジェクトに含まれる注釈やラベルなど。可能ですが、この種のオブジェクトのみからの復元は簡単ではない場合があります。このため、Kubernetes レベルのデータ保護ソリューションを使用する場合でも、常に `VolumeSnapshot` 定義をバックアップすることをお勧めします。

VolumeSnapshotContent の値は、対応する VolumeSnapshotClass 定義で設定された deletionPolicy によって決定されます。これは、.spec.backup.volumeSnapshot.className オプションで参照されます。

Kubernetes documentation on Volume Snapshot Classes を参照してください。

この標準の動作の詳細については、

8.1.6 バックアップボリュームのスナップショットの期限

CloudNativePG クラスターの EDB Postgres AI は、ボリュームスナップショット方法を使用したバックアップをサポートしています。一部の環境では、ボリュームスナップショットで再試行可能な一時的な問題が発生する場合があります。

backup.k8s.enterprisedb.io/volumeSnapshotDeadline アノテーションは、CloudNativePG クラスターの EDB Postgres AI が、バックアップを失敗としてマーキングする前に回復可能なエラーの再試行を継続する時間を定義します。

backup.k8s.enterprisedb.io/volumeSnapshotDeadline アノテーションは、Backup および ScheduledBackup リソースの両方に追加できます。ScheduledBackup リソースの場合、このアノテーションは、スケジュールから作成された Backup リソースに自動的に継承されます。

指定しない場合、デフォルトの再試行期限は **10 分** です。

エラー処理

ボリュームスナップショット操作中に再試行可能なエラーが発生した場合

1. CloudNativePG クラスターの EDB Postgres AI は、最初のエラーの時間を記録します。
2. システムは **10 秒** ごとに操作を再試行します。
3. エラーが指定された期限またはデフォルトの 10 分を超えて続く場合、バックアップは**失敗** としてマークされます。

再試行可能なエラー

CloudNativePG クラスターの EDB Postgres AI は、次の種類のエラーを再試行可能として扱います。

- **サーバータイムアウトエラー** HTTP 408、429、500、502、503、504
- **競合** オプティミスティックロックエラー
- **内部エラー**
- **コンテキスト期限超過エラー**
- **CSI スナップショットコントローラからのタイムアウトエラー**

例

次のようにして、アノテーションを ScheduledBackup リソースに追加できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: ScheduledBackup
metadata:
  name: daily-backup-schedule
  annotations:
    backup.k8s.enterprisedb.io/volumeSnapshotDeadline: "20"
spec:
  schedule: "0 0  **  *"
  backupOwnerReference: self
  method: volumeSnapshot
  # other configuration...
```

アノテーションを使用して ScheduledBackup を定義すると、このスケジュールから作成される Backup リソースは、指定されたタイムアウト値を自動的に継承します。

次の例では、スケジュールから作成されたすべてのバックアップには、回復可能なスナップショットエラーを再試行するために 30 分のタイムアウトがあります。

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: ScheduledBackup
metadata:
  name: weekly-backup
  annotations:
    backup.k8s.enterprisedb.io/volumeSnapshotDeadline: "30"
spec:
  schedule: "0 0  **  0" # Weekly backup on Sunday
  method: volumeSnapshot
  cluster:
    name: my-postgresql-cluster
```

または、アノテーションを Backup リソースに直接追加できます。

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Backup
metadata:
  name: my-backup
  annotations:
    backup.k8s.enterprisedb.io/volumeSnapshotDeadline: "15"
spec:
  method: volumeSnapshot
  # other backup configuration...
```

8.1.7 ボリュームスナップショットバックアップの例

次の例は、ebs-sc ストレージクラスと csi-aws-vsc ボリュームスナップショットクラスを使用して、AWS の EKS クラスターでボリュームスナップショットベースバックアップを構成する方法を示しています。

ストレージクラスとスナップショットクラスのインストールプロセスの詳細な手順については、を参照してください。

次のマニフェストは、ボリュームスナップショットに使用する準備ができ、サービスアカウント IRSA の IAM ロールを介して S3 バケットに WAL アーカイブを保存するを作成します *AWS S3* を参照してください。

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: hendrix
spec:
  instances: 3

  storage:
    storageClass: ebs-sc
    size: 10Gi
  walStorage:
```

(次のページに続く)

(前のページからの続き)

```
    storageClass: ebs-sc
    size: 10Gi

  backup:
    volumeSnapshot:
       className: csi-aws-vsc

  plugins:
  - name: barman-cloud.cloudnative-pg.io
    isWALArchiver: true
    parameters:
      barmanObjectName: @OBJECTSTORE_NAME@

  serviceAccountTemplate:
    metadata:
      annotations:
        eks.amazonaws.com/role-arn: "@ARN@"
- --
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: ScheduledBackup
metadata:
  name: hendrix-vs-backup
spec:
  cluster:
    name: hendrix
  method: volumeSnapshot
  schedule: 0 0 0  **  *
  backupOwnerReference: cluster
  immediate: true
```

最後のリソースは、深夜に毎日のボリュームスナップショットバックアップを定義し、クラスターが作成された直後に 1 つを要求します。

8.2 Appendix B - Backup on object stores

> **警告**
>
> CloudNativePG クラスター 1.26 の EDB Postgres AI の時点で、** Barman Cloud プラグイン**を優先して、**ネイティブ Barman Cloud サポートは非推奨です**。このページは、参考のために付録に移動しました。現時点ではネイティブ統合は機能し続けていますが、適切なテストの後、プラグインベースのインターフェイスへの段階的な移行を開始することを強くお勧めします。ガイダンスについては、Migrating from Built-in EDB Postgres AI for CloudNativePG Cluster Backup を参照してください。

CloudNativePG クラスターの EDB Postgres AI は、オブジェクトストアでの継続的な物理バックアップと WAL アーカイブを介して、PostgreSQL クラスターの**オンライン/ホットバックアップ**をネイティブにサポートしています。これは、データベースが常に稼働しており、ダウンタイムが必要なく、ポイントインタイムリカバリーが利用できることを意味します。

オペレーターは、Barman Cloud ツールに基づいた継続的なバックアップインフラストラクチャをオーケストレーションできます。多くの PostgreSQL インスタンスをバックアップする Barman サーバーで古典的なアーキテクチャを使用する代わりに、オペレーターは `barman-cloud-wal-archive` 、`barman-cloud-check-wal-archive` 、`barman-cloud-backup` 、`barman-cloud-backup-list` 、および `barman-cloud-backup-delete` ツールに依存します。その結果、ベースバックアップは_tarballs_になります。ベースバックアップと WAL ファイルは両方とも圧縮および暗号化できます。

このためには、 `barman-cli-cloud` が含まれるイメージを使用する必要があります。コミュニティ PostgreSQL イメージと最新の `barman-cli-cloud` パッケージで構成されているため、このスコープにイメージ `docker.enterprisedb.com/k8s/postgresql` を使用できます。

バックアップは、`Cluster` のプライマリまたは指定されたプライマリインスタンスから実行されます**レプリカクラスター** を参照してください

指定されたプライマリインスタンスの詳細については、 **スタンバイからのバックアップ** 。

8.2.1 共通オブジェクトストア

AWS S3 、 *Microsoft Azure Blob Storage* 、 *Google Cloud Storage* 、互換性のあるプロバイダーなどの特定のオブジェクトストア、または互換性のあるプロバイダーを探している場合は、*Appendix C - Common object stores for backups* を参照してください。

8.2.2 WAL アーカイブ

WAL アーカイブは、 *PostgreSQL WAL アーカイブ* をフィードするプロセスです

CloudNativePG クラスターの EDB Postgres AI で。

WAL アーカイブは、`Cluster` リソースの`.spec.backup.barmanObjectStore` スタンザで定義されています。

> **注釈**
>
> BarmanObjectStoreConfiguration を参照してください。

オプションの完全なリストについては、barman-cloud API で 。

必要に応じて、WAL ファイルがアップロードされたらすぐに圧縮および/または暗号化することを選択できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
[...]
spec:
```

(次のページに続く)

(前のページからの続き)

```
  backup:
    barmanObjectStore:
      [...]
      wal:
        compression: gzip
        encryption: AES256
```

バケットに暗号化を直接構成できます。クラスター構成でオーバーライドしない限り、オペレーターはそれを使用します。

PostgreSQL は、シークエンシャルアーカイブスキームを実装しており、アーカイブされるすべての WAL セグメントに対して archive_command がシークエンシャルに実行されます。

PostgreSQL インスタンスとオブジェクトストア間の帯域幅で複数の WAL ファイルを並列してアーカイブできる場合、次の例のようにインスタンスマネージャーの並列 WAL アーカイブ機能を使用できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
[...]
spec:
  backup:
    barmanObjectStore:
      [...]
      wal:
        compression: gzip
        maxParallel: 8
        encryption: AES256
```

前の例では、インスタンスマネージャーは、PostgreSQL によって要求されたものを含む最大 8 つの準備ができている WAL を並列アーカイブすることにより、WAL アーカイブプロセスを最適化します。

PostgreSQL が、最適化としてインスタンスマネージャーによって既にアーカイブされた WAL のアーカイブを要求する場合、そのアーカイブ要求はポジティブなステータスで却下されるだけです。

8.2.3 リテンション ポリシー

CloudNativePG クラスターの EDB Postgres AI は、リカバリウィンドウに基づいて**リテンションポリシー**を使用して、バックアップオブジェクトストアからのバックアップファイルの自動削除を管理できます。

内部的には、保持ポリシー機能は barman-cloud-backup-delete と--retention-policy “RECOVERY WINDOW OF {{ retention policy value }} {{ retention policy unit }}”を使用します。

たとえば、次のように 30 日の保持ポリシーを使用してバックアップを定義できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
[...]
spec:
```

(次のページに続く)

(前のページからの続き)

```
backup:
  barmanObjectStore:
    destinationPath: "<destination path here>"
    s3Credentials:
      accessKeyId:
        name: aws-creds
        key: ACCESS_KEY_ID
      secretAccessKey:
        name: aws-creds
        key: ACCESS_SECRET_KEY
  retentionPolicy: "30d"
```

8.2.4 圧縮アルゴリズム

CloudNativePG クラスターの EDB Postgres AI は、デフォルトでバックアップと WAL ファイルを非圧縮方法でアーカイブします。ただし、`barman-cloud-backup` バックアップ用および `barman-cloud-wal-archive` WAL ファイル用を介して次の圧縮アルゴリズムもサポートします。

- bzip2
- gzip
- lz4
- snappy
- xx
- zstd

バックアップと WAL の圧縮設定は独立しています。 barman-cloud API リファレンスの DataBackupConfiguration および WALBackupConfiguration セクションを参照してください。

アーカイブ時間、復元時間、およびサイズはアルゴリズムによって異なるため、ユースケースに応じて圧縮アルゴリズムを選択する必要があることに注意してください。

Barman チームは、 Barman Cloud でサポートされているアルゴリズムのパフォーマンスの評価を実行しました。次の表は、ローカルの MinIO 展開でバックアップが作成されるシナリオをまとめています。Barman GitHub プロジェクトには、 deeper analysis が含まれています。

Compression	Backup Time (ms)	Restore Time (ms)	Uncompressed size (MB)	Compressed size (MB)	Approx ratio
None	10927	7553	395	395	1:1
bzip2	25404	13886	395	67	5.9:1
gzip	116281	3077	395	91	4.3:1
snappy	8134	8341	395	166	2.4:1

8.2.5 バックアップオブジェクトのタグ付け

Barman 2.18 では、 barman-cloud-backup および barman-cloud-wal-archive を介してオブジェクトストアに保存する際のバックアップリソースのタグ付けのサポートを導入しました。その結果、PostgreSQL コンテナイメージにバージョン 2.18 以降の Barman が含まれている場合、CloudNativePG クラスターの EDB Postgres AI を使用して、バックアップオブジェクト、つまりベースバックアップ、WAL ファイル、履歴ファイルのキーと値のペアとしてタグを指定できます。

.spec.backup.barmanObjectStore 定義では 2 つのプロパティを使用できます。

- tags バックアップオブジェクトとバックアップオブジェクトストアのアーカイブされた WAL ファイルに追加するキーと値のペアタグ
- historyTags バックアップオブジェクトストアのアーカイブされた履歴ファイルに追加されるキーと値のペアタグ

以下の YAML マニフェストの抜粋は、この機能の使用例を提供します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
[...]
spec:
  backup:
    barmanObjectStore:
      [...]
      tags:
        backupRetentionPolicy: "expire"
      historyTags:
        backupRetentionPolicy: "keep"
```

8.2.6 backup および WAL コマンドの追加オプション

.spec.backup.barmanObjectStore.data および.spec.backup.barmanObjectStore.wal セクションの additionalCommandArgs プロパティを使用して、barman-cloud-backup および barman-cloud-wal-archive コマンドに追加オプションを追加できます。これらのプロパティは、barman-cloud-backup および barman-cloud-wal-archive コマンドに追加される文字列のリストです。

たとえば、 --read-timeout=60 を使用して、接続の読み取りタイムアウトをカスタマイズできます。

barman-cloud-backup および barman-cloud-wal-archive コマンドでサポートされる追加オプションについては、公式バーマンドキュメント here を参照してください。

additionalCommandArgs で提供されるオプションが、そのセクション.spec.backup.barmanObjectStore.data または.spec.backup.barmanObjectStore.wal の宣言されたオプションに既に存在する場合、追加のオプションは無視されます。

次に、このプロパティの使用方法の例を示します。

バックアップの場合

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
[...]
spec:
  backup:
    barmanObjectStore:
      [...]
      data:
        additionalCommandArgs:
        - "--min-chunk-size=5MB"
        - "--read-timeout=60"
```

WAL ファイルの場合

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
[...]
spec:
  backup:
    barmanObjectStore:
      [...]
      wal:
        additionalCommandArgs:
        - "--max-concurrency=1"
        - "--read-timeout=60"
```

8.2.7 オブジェクトストアからのリカバリー

Barman Cloud によって作成され、サポートされているオブジェクトストアに保存されているバックアップから復元できます。`barmanObjectStore` セクションで必要な構成をすべて含む外部クラスターを定義した後、`.spec.recovery.source` オプションで参照する必要があります。

この例では、Azure の BLOB コンテナにリカバリオブジェクトストアを定義します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-restore
spec:
  [...]

  superuserSecret:
    name: superuser-secret

  bootstrap:
    recovery:
```

(次のページに続く)

(前のページからの続き)

```
    source: clusterBackup

  externalClusters:
    - name: clusterBackup
      barmanObjectStore:
        destinationPath: https://STORAGEACCOUNTNAME.blob.core.windows.net/
↪CONTAINERNAME/
        azureCredentials:
          storageAccount:
            name: recovery-object-store-secret
            key: storage_account_name
          storageKey:
            name: recovery-object-store-secret
            key: storage_account_key
        wal:
          maxParallel: 8
```

前の例では、アプリケーションデータベースとその所有ユーザーの名前がデフォルトで app であることを前提としています。リストアする PostgreSQL クラスターが別の名前を使用している場合、**アプリケーションデータベースの構成** に記載されているように、リカバリフェーズを終了する前にこれらの名前を指定する必要があります。

注釈

この例では、並列 WAL リストア機能を利用し、最大 8 つのジョブを専用に作成して、アーカイブから必要な WAL ファイルを同時に取得します。この機能により、復旧時間を大幅に短縮できます。このシナリオを事前に計画し、ご使用の環境に合わせてこのパラメーターの値を正しく調整してください。必要なときに違いを生みますし、そうします。

8.3 Appendix C - Common object stores for backups

警告

CloudNativePG クラスター 1.26 の EDB Postgres AI の時点で、** Barman Cloud プラグイン**を優先して、**ネイティブ Barman Cloud サポートは非推奨です**。現時点ではネイティブ統合は機能し続けていますが、適切なテストの後、プラグインベースのインターフェイスへの段階的な移行を開始することを強くお勧めします。 Barman Cloud Plugin ドキュメントでは、

how to use common object stores について説明しています。

Appendix B - Backup on object stores ファイルは、 Barman Cloud インフラストラクチャでサポートされている任意のサービスに保存できます。つまり

- *AWS S3*

- *Azure Blob Storage*
- *Google Cloud Storage*

サポートされているサービスの互換性のある実装を使用することもできます。

必要なセットアップは、選択したストレージプロバイダーによって異なり、以降のセクションで説明します。

8.3.1 AWS S3

AWS Simple Storage Service (S3) は、Amazon が提供する非常に人気のオブジェクトストレージサービスです。

CloudNativePG クラスターバックアップの EDB Postgres AI に関する限り、2 つの方法で S3 バケットにバックアップを保存する権限を定義できます。

- CloudNativePG クラスターの EDB Postgres AI が EKS で実行されている場合。IRSA authentication method を使用することができます
- または、 `ACCESS_KEY_ID` および `ACCESS_SECRET_KEY` 資格情報を使用できます

AWS アクセスキー

環境に関する次の情報が必要です。

- `ACCESS_KEY_ID` S3 にファイルをアップロードするために使用されるアクセスキーの ID
- `ACCESS_SECRET_KEY` 上記のアクセスキーのシークレット部分
- `ACCESS_SESSION_TOKEN` オプションのセッショントークン、必要な場合

使用されるアクセスキーには、バケットにファイルをアップロードする権限が必要です。そのため、資格情報を使用して Kubernetes シークレットを作成する必要があります。これは、次のコマンドで行うことができます。

```
kubectl create secret generic aws-creds \
  --from-literal=ACCESS_KEY_ID=<access key here> \
  --from-literal=ACCESS_SECRET_KEY=<secret key here>

#  --from-literal=ACCESS_SESSION_TOKEN=<session token here> # if required
```

資格情報は Kubernetes 内に保存され、インストールで保存時の暗号化が構成されている場合、暗号化されます。

そのシークレットが作成されたら、次の例のようにクラスターを構成できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
[...]
spec:
  backup:
    barmanObjectStore:
```

(次のページに続く)

(前のページからの続き)

```
    destinationPath: "<destination path here>"
    s3Credentials:
      accessKeyId:
        name: aws-creds
        key: ACCESS_KEY_ID
      secretAccessKey:
        name: aws-creds
        key: ACCESS_SECRET_KEY
```

宛先パスには、インスタンスが WAL ファイルをアップロードできるフォルダーを指す任意の URL を指定できます。`s3://BUCKET_NAME/path/to/folder` .

サービスアカウント IRSA の IAM ロール

IRSA を使用するには、Postgres クラスターの `ServiceAccount` で `annotation` を設定する必要があります。

`serviceAccountTemplate` スタンザを使用してそれらを注入するように CloudNativePG クラスターの EDB Postgres AI を構成できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
[...]
spec:
  serviceAccountTemplate:
    metadata:
      annotations:
        eks.amazonaws.com/role-arn: arn:[...]
        [...]
```

S3 ライフサイクルポリシー

Barman Cloud は S3 にオブジェクトを書き込み、 Barman Cloud リテンションポリシーによって削除されるまで更新しません。S3 ライフサイクルポリシーで推奨されるアプローチは、オブジェクトの現在のバージョンを Barman 保持ポリシーより数日長く有効期限があり、オブジェクトのバージョニングを有効にし、数日後に非最新バージョンの有効期限を有効にすることです。このようなポリシーは、誤った削除から保護し、CloudNativePG クラスターワークロードの EDB Postgres AI への権限を制限できるため、オブジェクトを完全に削除する権限を付与せずに S3 からオブジェクトを削除できます。

その他の S3 互換オブジェクトストレージプロバイダー

MinIO や **Linode Object Storage** のような S3 互換オブジェクトストレージを使用している場合、デフォルトの S3 のものを使用する代わりにエンドポイントを指定できます。

この例では、リージョン `us-east1` の **Linode** の `bucket` を使用します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
[...]
spec:
  backup:
    barmanObjectStore:
      destinationPath: "s3://bucket/"
      endpointURL: "https://us-east1.linodeobjects.com"
      s3Credentials:
        [...]
```

Digital Ocean Spaces を使用している場合、Path スタイルの構文を使用する必要があります。この例では、リージョン SFO3 の **Digital Ocean Spaces** の bucket を使用します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
[...]
spec:
  backup:
    barmanObjectStore:
      destinationPath: "s3://[your-bucket-name]/[your-backup-folder]/"
      endpointURL: "https://sfo3.digitaloceanspaces.com"
      s3Credentials:
        [...]
```

注釈

boto3 implementation の最近の変更

Amazon S3 Data Integrity Protections の

x-amz-content-sha256 エラーが発生する可能性があります。この問題が発生した場合は、spec.env を介してクラスターレベルで特定の環境変数を設定することにより、次の回避策を適用できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
[...]
spec:
  env:
    - name: AWS_REQUEST_CHECKSUM_CALCULATION
      value: when_required
    - name: AWS_RESPONSE_CHECKSUM_VALIDATION
      value: when_required
```

例、HTTPS 経由で OpenShift または MinIO を使用する場合、プライベート CA で署名された証明書を使用するオブジェクトストレージプロバイダーを構成するとします。その場合、CA バンドルを含むシーク

レットを参照する barmanObjectStore 内にオプション endpointCA を設定して、Barman が証明書を正しく検証できるようにする必要があります。*TLS* 証明書の構成 ドキュメントの証明書ファイルを使用してシークレットを作成する手順を見つけることができます。シークレットを作成したら、次の例のように endpointCA を設定できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
[...]
spec:
  [...]
  backup:
    barmanObjectStore:
      endpointURL: <myEndpointURL>
      endpointCA:
        name: my-ca-secret
        key: ca.crt
```

注釈

ConfigMaps と Secrets をインスタンスによって**自動的に**リロードしたい場合は、キー `k8s.enterprisedb.io/reload` を持つラベルを Secrets/ConfigMaps に追加できます。それ以外の場合は、*kubectl cnp reload* サブコマンドを使用してインスタンスをリロードする必要があります。

8.3.2 Azure Blob Storage

Azure Blob Storage は、Microsoft が提供するオブジェクトストレージサービスです。

CloudNativePG クラスターの EDB Postgres AI は、Azure Blob Storage の次の認証方法をサポートしています。

-

Connection String

- ストレージアカウント名+

Storage Account Access Key

- ストレージアカウント名+

Storage Account SAS Token

- Azure AD Managed Identity
- Default Azure Credentials

Azure AD マネージド ID を使用すると、資格情報を Kubernetes シークレットに保存せず、次のように inheritFromAzureAD を追加するクラスター構成にすることができます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
[...]
spec:
  backup:
    barmanObjectStore:
      destinationPath: "<destination path here>"
      azureCredentials:
        inheritFromAzureAD: true
```

または、 **デフォルトの Azure 資格情報**認証メカニズムを使用できます。これは、環境変数、管理対象 ID、Azure CLI 資格情報を含む複数の認証方法をサポートすることにより、シームレスな認証エクスペリエンスを提供します。次のように useDefaultAzureCredentials フラグを追加します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
[...]
spec:
  backup:
    barmanObjectStore:
      destinationPath: "<destination path here>"
      azureCredentials:
        useDefaultAzureCredentials: true
```

一方、 **ストレージアカウントアクセスキー** または**ストレージアカウント SAS トークン**の両方を使用する場合、資格情報を Kubernetes Secret 内に保存し、必要な場合にのみデータエントリを追加する必要があります。次のコマンドはそれを実行します。

```
kubectl create secret generic azure-creds \
  --from-literal=AZURE_STORAGE_ACCOUNT=<storage account name> \
  --from-literal=AZURE_STORAGE_KEY=<storage account key> \
  --from-literal=AZURE_STORAGE_SAS_TOKEN=<SAS token> \
  --from-literal=AZURE_STORAGE_CONNECTION_STRING=<connection string>
```

使用される Kubernetes クラスターでこの機能が有効になっている場合、資格情報は保存時に暗号化されます。

前述のシークレットを踏まえて、提供された資格情報をクラスター構成内に注入できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
[...]
spec:
  backup:
    barmanObjectStore:
      destinationPath: "<destination path here>"
      azureCredentials:
```

(次のページに続く)

(前のページからの続き)

```
      connectionString:
        name: azure-creds
        key: AZURE_CONNECTION_STRING
      storageAccount:
        name: azure-creds
        key: AZURE_STORAGE_ACCOUNT
      storageKey:
        name: azure-creds
        key: AZURE_STORAGE_KEY
      storageSasToken:
        name: azure-creds
        key: AZURE_STORAGE_SAS_TOKEN
```

Azure Blob Storage を使用する場合、destinationPath は次の構造を満たします。

```
<http|https>://<account-name>.<service-name>.core.windows.net/<resource-path>
```

<resource-path> は<container>/<blob> です。**ストレージアカウント名** とも呼ばれる **アカウント名**は、使用されるホスト名に含まれます。

その他の Azure Blob Storage 互換プロバイダー

Azure Blob Storage API の別の実装を使用している場合、destinationPath は次の構造になります。

```
<http|https>://<local-machine-address>:<port>/<account-name>/<resource-path>
```

その場合、<account-name> はパスの最初のコンポーネントです。

これは、 Azure ストレージエミュレータまたは Azurite を介して Azure サポートをテストする場合に必要です。

8.3.3 Google Cloud Storage

現在、CloudNativePG クラスターオペレーターの EDB Postgres AI は、Google Cloud Storage の 2 つの認証方法をサポートしています。

- 最初のものは、ポッドが Google Kubernetes Engine クラスター内で実行されていることを前提としています
- 2 つ目は、環境変数 GOOGLE_APPLICATION_CREDENTIALS を活用します

Google Kubernetes Engine 内での実行

Google Kubernetes Engine 内で実行されている場合、資格情報を設定せずに Workload Identity に単に依存するようにバックアップを構成できます。特に、次のことを行う必要があります。

- .spec.backup.barmanObjectStore.googleCredentials.gkeEnvironment を true に設定します

- serviceAccountTemplate スタンザに iam.gke.io/gcp-service-account アノテーションを設定する

次の例を参考として使用してください。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
[...]
spec:
  [...]
  backup:
    barmanObjectStore:
      destinationPath: "gs://<destination path here>"
      googleCredentials:
        gkeEnvironment: true

  serviceAccountTemplate:
    metadata:
      annotations:
        iam.gke.io/gcp-service-account:  [...].iam.gserviceaccount.com
        [...]
```

認証を使用する

instruction from Google をフォローする

認証に必要な情報をすべて含む JSON ファイルを取得します。

JSON ファイルのコンテンツは、次のコマンドで作成できる Secret を使用して提供する必要があります。

```
kubectl create secret generic backup-creds --from-file=gcsCredentials=gcs_credentials_
↪file.json
```

これにより、次のように yaml ファイルで使用される backup-creds という名前で Secret が作成されます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
[...]
spec:
  backup:
    barmanObjectStore:
      destinationPath: "gs://<destination path here>"
      googleCredentials:
        applicationCredentials:
          name: backup-creds
          key: gcsCredentials
```

次に、オペレーターは資格情報を使用して Google Cloud Storage に対して認証します。

8.4 CNPG-I

CloudNativePG インターフェイス CNPG-I は、コアコードベースを変更せずに CloudNativePG クラスターの EDB Postgres AI を拡張およびカスタマイズする標準の方法です。

8.4.1 なぜ CNPG-I なのか？

CloudNativePG クラスターの EDB Postgres AI は、幅広いユースケースをサポートしていますが、そのビルトイン機能では十分でない場合、または特定の機能をメインプロジェクトに直接追加するのが現実的でない場合があります。

CNPG-I が登場するまで、ユーザーには 2 つの主なオプションがありました。

- プロジェクトをフォークしてカスタム動作を追加する、または
- 上流のコードベースにカスタムコンポーネントを記述することにより、上流のコードベースを拡張します。

どちらのアプローチも、メンテナンスのオーバーヘッドが発生し、アップグレードが遅く、重要な機能のデリバリーが遅延します。

CNPG-I は、コアプロジェクトを中断せずに、クラスターのライフサイクルのキーポイントでバックアップ、リカバリ、サブリソースのリコンシリエーションなど、CloudNativePG クラスターの EDB Postgres AI を拡張するための安定した gRPC ベースの統合ポイントを提供することにより、これらの問題を解決します。

CNPG-I は以下を拡張できます。

- 演算子、および/または
- PostgreSQL ポッド内で実行されているインスタンスマネージャー。

8.4.2 プラグインの登録

CNPG-I は、Kubernetes

Container Storage Interface (CSI) からインスピレーションを得ています。オペレーターは、CNPG-I protocol に従って、 **gRPC** を使用して登録されたプラグインと通信します。

2 つの方法のいずれかでプラグインを登録できます。

- サイドカーコンテナ – オペレーターの展開内でプラグインを実行します
- スタンドアロン展開 – 同じ名前空間内の別のワークロードとしてプラグインを実行します

どちらの場合も、プラグインはコンテナイメージとしてパッケージ化する必要があります。

サイドカーコンテナ

サイドカープラグインは、オペレーターのスタートアップに 1 回検出されます。

プラグインは、 **Unix ドメインソケット**を介して gRPC サーバーを公開する必要があります。このソケットは、オペレーターコンテナと共有するディレクトリに配置し、`PLUGIN_SOCKET_DIR` デフォルト`/plugin` で設定したパスにマウントする必要があります。

例

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: controller-manager
spec:
  template:
    spec:
      containers:
      - image: cloudnative-pg:latest
        [...]
        name: manager
        volumeMounts:
        - mountPath: /plugins
          name: cnpg-i-plugins

      - image: cnpg-i-plugin-example:latest
        name: cnpg-i-plugin-example
        volumeMounts:
        - mountPath: /plugins
          name: cnpg-i-plugins
      volumes:
      - name: cnpg-i-plugins
        emptyDir: {}
```

スタンドアロン展開推奨

プラグインを独自の展開として実行すると、そのライフサイクルがオペレーターのライフサイクルから分離され、独立したスケーリングが可能になります。この設定では、プラグインは、安全な通信のための **mTLS** を使用して、サービスの背後にある TCP gRPC エンドポイントを公開します。スタンドアロンプラグインは、必要なラベルとアノテーションを持つサービスを監視することにより、動的に検出されます。オペレーターの再起動は必要ありません。

スタンドアロンプラグインのポッドがロールされるとたとえば、イメージをアップグレードした後、オペレーターはプラグインサービスをバッキングするエンドポイントスライスを介して変更を検出し、そのプラグインを使用してすべてのクラスターをリコンサイルとするため、待機するのではなく、すぐに新しいポッドとの対話を開始します。定期的な再同期。

展開例

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cnpg-i-plugin-example
spec:
  template:
```

(次のページに続く)

(前のページからの続き)

```
    [...]
    spec:
      containers:
      - name: cnpg-i-plugin-example
        image: cnpg-i-plugin-example:latest
        ports:
        - containerPort: 9090
          protocol: TCP
```

プラグインの関連サービスには以下を含める必要があります。

- ラベル `k8s.enterprisedb.io/plugin: <plugin-name>` — CloudNativePG クラスターの EDB Postgres AI がプラグインを検出するために必要
- アノテーション `k8s.enterprisedb.io/pluginPort: <port>` - プラグインの gRPC サーバーが公開されるポートを指定します

サービス例

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    k8s.enterprisedb.io/pluginPort: "9090"
  labels:
    k8s.enterprisedb.io/pluginName: cnpg-i-plugin-example.my-org.io
  name: cnpg-i-plugin-example
spec:
  ports:
    - port: 9090
      protocol: TCP
      targetPort: 9090
  selector:
    app: cnpg-i-plugin-example
```

TLS 証明書の構成

プラグインが `Deployment` として実行される場合、CloudNativePG クラスターの EDB Postgres AI との通信はネットワーク経由で発生します。それを保護するために、**mTLS が適用され** 、両方の側に TLS 証明書が必要です。

証明書は Kubernetes TLS Secrets として保存する必要があります

およびプラグインのサービスアノテーション `k8s.enterprisedb.io/pluginClientSecret` および `k8s.enterprisedb.io/pluginServerSecret` で参照されます。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    k8s.enterprisedb.io/pluginClientSecret: cnpg-i-plugin-example-client-tls
    k8s.enterprisedb.io/pluginServerSecret: cnpg-i-plugin-example-server-tls
    k8s.enterprisedb.io/pluginPort: "9090"
  name: barman-cloud
  namespace: postgresql-operator-system
spec: [...]
```

> **注釈**
>
> 独自の証明書バンドルを提供できますが、 Cert-manager を使用することをお勧めします。

証明書の DNS 名のカスタマイズ

デフォルトでは、CloudNativePG クラスターの EDB Postgres AI は、プラグインに接続するときの TLS 検証のサーバー名としてサービス名を使用します。ご使用の環境により、証明書に別の DNS 名 `barman-cloud.svc` などが必要な場合、 `k8s.enterprisedb.io/pluginServerName` アノテーションを使用してカスタマイズできます。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    k8s.enterprisedb.io/pluginClientSecret: cnpg-i-plugin-example-client-tls
    k8s.enterprisedb.io/pluginServerSecret: cnpg-i-plugin-example-server-tls
    k8s.enterprisedb.io/pluginPort: "9090"
    k8s.enterprisedb.io/pluginServerName: barman-cloud.svc
  name: barman-cloud
  namespace: postgresql-operator-system
spec: [...]
```

これにより、オペレーターは、デフォルトのサービス名ではなく指定された DNS 名と比較してプラグインの証明書を検証できます。サーバー証明書には、この DNS 名がサブジェクト代替名 SAN に含まれている必要があります。

8.4.3 プラグインを使用する

プラグインを有効にするには、 `Cluster` リソースの `.spec.plugins` セクションを構成します。完全な PluginConfiguration の詳細については、 EDB Postgres AI for CloudNativePG クラスター API リファレンスを参照してください。

仕様。

例

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-with-plugins
spec:
  instances: 1
  storage:
    size: 1Gi
  plugins:
    - name: cnpg-i-plugin-example.my-org.io
      enabled: true
      parameters:
        key1: value1
        key2: value2
```

各プラグインは独自のパラメーターを持つ場合があります。詳細については、プラグインのドキュメントを確認してください。`spec.plugins` の `name` フィールドは、プラグインが展開される方法によって異なります。

- サイドカーコンテナ Unix ソケットファイル名を使用します
- Deployment サービスの `k8s.enterprisedb.io/pluginName` label の値を使用します

8.4.4 コミュニティプラグイン

CNPG-I プロトコルは、コアプロジェクトを保守可能に保ちながら、CloudNativePG クラスターの EDB Postgres AI を拡張するための実証された信頼性の高いパターンになりました。時間の経過とともに、コミュニティは実世界のニーズに対応し、開発者の例として機能するプラグインを構築および共有してきました。

CNPG-I でビルドされたプラグインの完全な最新のリストについては、CNPG-I GitHub page を参照してください。