EDB Postgres® AI for CloudNativePG™ Cluster

リリース *1.27.0*

EnterpriseDB Corporation

2025 年 09 月 08 日

目 次

第 1 章 Introduction and Release Notes **1**

1.1 Key features, long time support and licensing 1
1.1.1 CloudNativePG™と共通の主な機能 1
1.1.2 Kubernetes の EDB Postgres に固有の機能 2
1.1.3 ライセンス 2
1.1.4 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 3
1.1.5 サポートされている Postgres バージョン 3
1.1.6 このガイドについて 3
1.2 EDB CloudNativePG Cluster release notes 4
1.2.1 EDB CloudNativePG Cluster 1.27.0 release notes 8
1.2.2 EDB CloudNativePG Cluster 1.26.1 release notes 10
1.2.3 EDB CloudNativePG Cluster 1.26.0 release notes 10
1.2.4 EDB CloudNativePG Cluster 1.25.3 release notes 10
1.2.5 EDB CloudNativePG Cluster 1.25.2 release notes 11
1.2.6 EDB Postgres for Kubernetes 1.25.1 release notes 11
1.2.7 EDB Postgres for Kubernetes 1.25.0 release notes 11
1.2.8 EDB CloudNativePG Cluster 1.24.4 release notes 11
1.2.9 EDB Postgres for Kubernetes 1.24.3 release notes 12
1.2.10 EDB Postgres for Kubernetes 1.24.2 release notes 12
1.2.11 EDB Postgres for Kubernetes 1.24.1 release notes 12
1.2.12 EDB Postgres for Kubernetes 1.24.0 release notes 13
1.2.13 EDB Postgres for Kubernetes 1.23.6 release notes 13
1.2.14 EDB Postgres for Kubernetes 1.23.5 release notes 13
1.2.15 EDB Postgres for Kubernetes 1.23.4 release notes 13
1.2.16 EDB Postgres for Kubernetes 1.23.3 release notes 14
1.2.17 EDB Postgres for Kubernetes 1.23.2 release notes 14
1.2.18 EDB Postgres for Kubernetes 1.23.1 release notes 14
1.2.19 EDB Postgres for Kubernetes 1.23.0 release notes 14
1.2.20 EDB CloudNativePG Cluster 1.22.11 release notes 15
1.2.21 EDB CloudNativePG Cluster 1.22.10 release notes 16
1.2.22 EDB Postgres for Kubernetes 1.22.9 release notes 17
1.2.23 EDB Postgres for Kubernetes 1.22.8 release notes 19
1.2.24 EDB Postgres for Kubernetes 1.22.7 release notes 20
1.2.25 EDB Postgres for Kubernetes 1.22.6 release notes 21
1.2.26 EDB Postgres for Kubernetes 1.22.5 release notes 22
1.2.27 EDB Postgres for Kubernetes 1.22.4 release notes 22
1.2.28 EDB Postgres for Kubernetes 1.22.3 release notes 22
1.2.29 EDB Postgres for Kubernetes 1.22.2 release notes 22
1.2.30 EDB Postgres for Kubernetes 1.22.1 release notes 23

1.2.31 EDB Postgres for Kubernetes 1.22.0 release notes . . . 23
1.2.32 EDB Postgres for Kubernetes 1.21.6 release notes . . . 23
1.2.33 EDB Postgres for Kubernetes 1.21.5 release notes . . . 23
1.2.34 EDB Postgres for Kubernetes 1.21.4 release notes . . . 24
1.2.35 EDB Postgres for Kubernetes 1.21.3 release notes . . . 24
1.2.36 EDB Postgres for Kubernetes 1.21.2 release notes . . . 24
1.2.37 EDB Postgres for Kubernetes 1.21.1 release notes . . . 24
1.2.38 EDB Postgres for Kubernetes 1.21.0 release notes . . . 25
1.2.39 EDB Postgres for Kubernetes 1.20.6 release notes . . . 25
1.2.40 EDB Postgres for Kubernetes 1.20.5 release notes . . . 25
1.2.41 EDB Postgres for Kubernetes 1.20.4 release notes . . . 25
1.2.42 EDB Postgres for Kubernetes 1.20.3 release notes . . . 26
1.2.43 EDB Postgres for Kubernetes 1.20.2 release notes . . . 26
1.2.44 EDB Postgres for Kubernetes 1.20.1 release notes . . . 26
1.2.45 EDB Postgres for Kubernetes 1.20.0 release notes . . . 26
1.2.46 EDB Postgres for Kubernetes 1.19.6 release notes . . . 27
1.2.47 EDB Postgres for Kubernetes 1.19.5 release notes . . . 27
1.2.48 EDB Postgres for Kubernetes 1.19.4 release notes . . . 27
1.2.49 EDB Postgres for Kubernetes 1.19.3 release notes . . . 27
1.2.50 EDB Postgres for Kubernetes 1.19.2 release notes . . . 28
1.2.51 EDB Postgres for Kubernetes 1.19.1 release notes . . . 28
1.2.52 EDB Postgres for Kubernetes 1.19.0 release notes . . . 28
1.2.53 EDB Postgres for Kubernetes 1.18.13 release notes . . . 28
1.2.54 EDB Postgres for Kubernetes 1.18.12 release notes . . . 29
1.2.55 EDB Postgres for Kubernetes 1.18.11 release notes . . . 30
1.2.56 EDB Postgres for Kubernetes 1.18.10 release notes . . . 31
1.2.57 EDB Postgres for Kubernetes 1.18.9 release notes . . . 32
1.2.58 EDB Postgres for Kubernetes 1.18.8 release notes . . . 33
1.2.59 EDB Postgres for Kubernetes 1.18.7 release notes . . . 34
1.2.60 EDB Postgres for Kubernetes 1.18.6 release notes . . . 36
1.2.61 EDB Postgres for Kubernetes 1.18.5 release notes . . . 36
1.2.62 EDB Postgres for Kubernetes 1.18.4 release notes . . . 36
1.2.63 EDB Postgres for Kubernetes 1.18.3 release notes . . . 37
1.2.64 EDB Postgres for Kubernetes 1.18.2 release notes . . . 37
1.2.65 EDB Postgres for Kubernetes 1.18.1 release notes . . . 37
1.2.66 EDB Postgres for Kubernetes 1.18.0 release notes . . . 37
1.2.67 EDB Postgres for Kubernetes 1.17.5 release notes . . . 38
1.2.68 EDB Postgres for Kubernetes 1.17.4 release notes . . . 38
1.2.69 EDB Postgres for Kubernetes 1.17.3 release notes . . . 38
1.2.70 EDB Postgres for Kubernetes 1.17.2 release notes . . . 38
1.2.71 EDB Postgres for Kubernetes 1.17.1 release notes . . . 39
1.2.72 EDB Postgres for Kubernetes 1.17.0 release notes . . . 39
1.2.73 EDB Postgres for Kubernetes 1.16.5 release notes . . . 40
1.2.74 EDB Postgres for Kubernetes 1.16.4 release notes . . . 40
1.2.75 EDB Postgres for Kubernetes 1.16.3 release notes . . . 40
1.2.76 EDB Postgres for Kubernetes 1.16.2 release notes . . . 41
1.2.77 EDB Postgres for Kubernetes 1.16.1 release notes . . . 41

1.2.78 EDB Postgres for Kubernetes 1.16.0 release notes . . . 42
1.2.79 EDB Postgres for Kubernetes 1.15.5 release notes . . . 44
1.2.80 EDB Postgres for Kubernetes 1.15.4 release notes . . . 44
1.2.81 EDB Postgres for Kubernetes 1.15.3 release notes . . . 45
1.2.82 EDB Postgres for Kubernetes 1.15.2 release notes . . . 45
1.2.83 EDB Postgres for Kubernetes 1.15.1 release notes . . . 46
1.2.84 EDB Postgres for Kubernetes 1.15.0 release notes . . . 46
1.2.85 EDB Postgres for Kubernetes 1.14.0 release notes . . . 46
1.2.86 EDB Postgres for Kubernetes 1.13.0 release notes . . . 47
1.2.87 EDB Postgres for Kubernetes 1.12.0 release notes . . . 49
1.2.88 EDB Postgres for Kubernetes 1.11.0 release notes . . . 50
1.2.89 EDB Postgres for Kubernetes 1.10.0 release notes . . . 52
1.2.90 EDB Postgres for Kubernetes 1.9.2 release notes . . . 53
1.2.91 EDB Postgres for Kubernetes 1.9.1 release notes . . . 54
1.2.92 EDB Postgres for Kubernetes 1.9.0 release notes . . . 54
1.2.93 EDB Postgres for Kubernetes 1.8.0 release notes . . . 54
1.2.94 EDB Postgres for Kubernetes 0.8 release notes . . . 56
1.2.95 EDB Postgres for Kubernetes 1.7.1 release notes . . . 56
1.2.96 EDB Postgres for Kubernetes 1.7.0 release notes . . . 56
1.2.97 EDB Postgres for Kubernetes 1.6.0 release notes . . . 57
1.2.98 EDB Postgres for Kubernetes 1.5.1 release notes . . . 58
1.2.99 EDB Postgres for Kubernetes 1.5.0 release notes . . . 58
1.2.100 EDB Postgres for Kubernetes 1.4.0 release notes . . . 59
1.2.101 EDB Postgres for Kubernetes 1.3.0 release notes . . . 60
1.2.102 EDB Postgres for Kubernetes 1.2.1 release notes . . . 61
1.2.103 EDB Postgres for Kubernetes 1.2.0 release notes . . . 61
1.2.104 EDB Postgres for Kubernetes 1.1 release notes . . . 61
1.2.105 EDB Postgres for Kubernetes 1.0 release notes . . . 62
1.2.106 EDB Postgres for Kubernetes 0.7 release notes . . . 63
1.2.107 EDB Postgres for Kubernetes 0.6 release notes . . . 63
1.2.108 EDB Postgres for Kubernetes 0.5 release notes . . . 64
1.2.109 EDB Postgres for Kubernetes 0.4 release notes . . . 64
1.2.110 EDB Postgres for Kubernetes 0.3 release notes . . . 65
1.2.111 EDB Postgres for Kubernetes 0.2 release notes . . . 65
1.2.112 EDB Postgres for Kubernetes 0.1 release notes . . . 66
1.2.113 EDB Postgres for Kubernetes 0.0.1 release notes . . . 66

第 2 章 Getting Started **69**
2.1 Architecture . . . 69
2.1.1 状態の同期 . . . 69
2.1.2 Kubernetes アーキテクチャ . . . 70
2.1.3 PostgreSQL アーキテクチャ . . . 74
2.1.4 Kubernetes クラスター全体のデプロイメント . . . 77
2.2 Before You Start . . . 79
2.2.1 Kubernetes の用語 . . . 79
2.2.2 PostgreSQL の用語 . . . 80
2.2.3 クラウド用語 . . . 81

2.2.4 次に何をするか . . . 81
2.3 Use cases . . . 82
2.3.1 ケース 1Kubernetes 内のアプリケーション . . . 82
2.3.2 ケース 2Kubernetes 外部のアプリケーション . . . 82
2.4 Installation and upgrades . . . 83
2.4.1 Kubernetes へのインストール . . . 84
2.4.2 デプロイメントの詳細 . . . 86
2.4.3 アップグレード . . . 87
2.5 Quickstart . . . 92
2.5.1 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 92
2.5.2 パート 2 CloudNativePG™クラスターに EDB Postgres ® AI をインストールする . . 94
2.5.3 パート 3 PostgreSQL クラスターを展開する . . . 94
2.5.4 パート 4 Prometheus と Grafana を使用してクラスターを監視する . . . 96

第 3 章 Configuration **101**
3.1 PostgreSQL Configuration . . . 101
3.1.1 `postgresql` セクション . . . 101
3.1.2 `pg_hba` セクション . . . 108
3.1.3 `pg_ident` セクション . . . 110
3.1.4 構成の変更 . . . 111
3.1.5 `ALTER SYSTEM` の有効化 . . . 111
3.1.6 動的共有メモリ設定 . . . 111
3.1.7 固定パラメーター . . . 113
3.2 Operator configuration . . . 115
3.2.1 使用可能なオプション . . . 116
3.2.2 オペレーター構成マップの定義 . . . 118
3.2.3 オペレーターシークレットの定義 . . . 118
3.2.4 オペレーターを再起動して構成をリロードする . . . 118
3.2.5 pprof HTTP Server . . . 119
3.3 Instance pod configuration . . . 120
3.3.1 予測される量 . . . 120
3.3.2 エフェメラルボリューム . . . 121
3.3.3 環境変数 . . . 122
3.4 Examples . . . 123
3.4.1 基本 . . . 123
3.4.2 バックアップ . . . 124
3.4.3 レプリカクラスター . . . 125
3.4.4 PostGIS . . . 125
3.4.5 管理対象ロール . . . 125
3.4.6 管理対象サービス . . . 125
3.4.7 宣言的テーブルスペース . . . 125
3.4.8 プーラー構成 . . . 126
3.4.9 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 . . . 126
3.4.10 Postgres データベースの宣言的管理 . . . 127

第 4 章 Using **129**

4.1 Bootstrap . . . 129
4.1.1 bootstrap セクション . . . 130
4.1.2 externalClusters セクション . . . 131
4.1.3 空のクラスターをブートストラップする initdb . . . 132
4.1.4 別のクラスターからのブートストラップ . . . 138
4.2 Importing Postgres databases . . . 145
4.2.1 その仕組み . . . 145
4.2.2 microservice タイプ . . . 146
4.2.3 monolith タイプ . . . 148
4.2.4 実際の例 . . . 150
4.2.5 インポートの最適化 . . . 152
4.2.6 pg_dump および pg_restore の動作のカスタマイズ . . . 153
4.2.7 オンラインインポートとアップグレード . . . 154
4.3 Security . . . 154
4.3.1 コード . . . 154
4.3.2 コンテナ . . . 155
4.3.3 クラスター . . . 156
4.4 Postgres instance manager . . . 163
4.4.1 スタートアッププローブ . . . 163
4.4.2 Liveness Probe . . . 166
4.4.3 Readiness Probe . . . 168
4.4.4 シャットダウン制御 . . . 170
4.4.5 フェイルオーバー . . . 171
4.4.6 ディスクフル障害 . . . 171
4.5 Scheduling . . . 172
4.5.1 ポッドアフィニティとアンチアフィニティ . . . 172
4.5.2 nodeSelector によるノード選択 . . . 174
4.5.3 許容範囲 . . . 175
4.5.4 PostgreSQL ワークロードの分離 . . . 175
4.6 Resource management . . . 176
4.7 Failure Modes . . . 178
4.7.1 自己修復 . . . 179
4.7.2 手動介入 . . . 179
4.8 Rolling Updates . . . 180
4.8.1 自動更新 unsupervised . . . 180
4.8.2 手動更新 supervised . . . 181
4.9 Replication . . . 181
4.9.1 アプリケーションレベルのレプリケーション . . . 182
4.9.2 非常に成熟したテクノロジー . . . 182
4.9.3 ストリーミングレプリケーションのサポート . . . 183
4.9.4 同期レプリケーション . . . 184
4.9.5 同期レプリケーション非推奨 . . . 190
4.9.6 レプリケーションスロット . . . 192
4.10 Backup . . . 197
4.10.1 メインコンセプト . . . 198
4.10.2 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 . . . 200

4.10.3 スケジュールされたバックアップ . . . 201
4.10.4 オンデマンド バックアップ . . . 203
4.10.5 バックアップ方法 . . . 205
4.10.6 スタンバイからのバックアップ . . . 205
4.10.7 リテンションポリシー . . . 207
4.11 Recovery . . . 207
4.11.1 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 . . . 208
4.11.2 `VolumeSnapshot` オブジェクトからのリカバリー . . . 210
4.11.3 `Backup` オブジェクトからのリカバリー . . . 211
4.11.4 追加の考慮事項 . . . 212
4.11.5 ポイント インタイム リカバリー PITR . . . 212
4.11.6 アプリケーションデータベースの構成 . . . 217
4.11.7 リカバリーの内部的な仕組み . . . 218
4.11.8 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 . . . 219
4.12 Appendix A - Backup on volume snapshots . . . 219
4.12.1 標準のボリュームスナップショットについて . . . 220
4.12.2 要件 . . . 220
4.12.3 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 . . . 220
4.12.4 ホットおよびコールドバックアップ . . . 222
4.12.5 ボリュームスナップショットオブジェクトの永続化 . . . 224
4.12.6 バックアップボリュームのスナップショットの期限 . . . 224
4.12.7 ボリュームスナップショットバックアップの例 . . . 226
4.13 Appendix B - Backup on object stores . . . 227
4.13.1 共通オブジェクトストア . . . 228
4.13.2 WAL アーカイブ . . . 228
4.13.3 リテンション ポリシー . . . 230
4.13.4 圧縮アルゴリズム . . . 230
4.13.5 バックアップオブジェクトのタグ付け . . . 231
4.13.6 backup および WAL コマンドの追加オプション . . . 232
4.13.7 オブジェクトストアからのリカバリー . . . 233
4.14 WAL archiving . . . 234
4.14.1 プラグインベースのアーキテクチャ . . . 234
4.14.2 サポートされているプラグイン . . . 234
4.14.3 非推奨の通知 ネイティブ Barman Cloud . . . 234
4.14.4 アーカイブタイムアウトについて . . . 235
4.15 PostgreSQL Role Management . . . 235
4.15.1 パスワード管理 . . . 236
4.15.2 実現不可能なロール構成 . . . 238
4.15.3 管理対象ロールのステータス . . . 238
4.16 Storage . . . 240
4.16.1 バックアップ リカバリー . . . 241
4.16.2 CloudNativePG™クラスターの EDB Postgres ® AI のベンチマーク . . . 241
4.16.3 保存時の暗号化 . . . 241
4.16.4 Persistent Volume Claim Claim PVC . . . 242
4.16.5 ストレージクラスを介した構成 . . . 242
4.16.6 PVC テンプレートを介した構成 . . . 243
4.16.7 WAL のボリューム . . . 243

4.16.8 テーブルスペースのボリューム . . . 244
4.16.9 ボリュームの拡張 . . . 244
4.16.10 永続ボリュームの静的プロビジョニング . . . 246
4.16.11 ブロックストレージの考慮事項 Ceph/Longhorn . . . 247
4.17 Labels and annotations . . . 248
4.17.1 事前定義されたラベル . . . 248
4.17.2 事前定義されたアノテーション . . . 249
4.17.3 前提条件 . . . 252
4.17.4 クラスターのメタデータの定義 . . . 252
4.17.5 現在の制限 . . . 252
4.18 Monitoring . . . 253
4.18.1 モニタリングインスタンス . . . 253
4.18.2 CloudNativePG™クラスターオペレーターの EDB Postgres ® AI のモニタリング . . . 271
4.18.3 エクスポートされたメトリックを検査する方法 . . . 272
4.18.4 補助リソース . . . 274
4.18.5 OpenShift でのモニタリング . . . 274
4.19 Logging . . . 275
4.19.1 クラスターログ . . . 276
4.19.2 オペレーターログ . . . 276
4.19.3 PostgreSQL ログ . . . 276
4.19.4 PG 監査ログ . . . 277
4.19.5 EDB Audit log . . . 279
4.19.6 その他のログ . . . 281
4.20 Certificates . . . 282
4.20.1 オペレーター管理モード . . . 283
4.20.2 ユーザー提供の証明書モード . . . 284
4.21 Client TLS/SSL connections . . . 289
4.21.1 新しい証明書の発行 . . . 290
4.21.2 TLS 証明書を介した接続のテスト . . . 290
4.21.3 TLS プロトコルバージョンについて . . . 293
4.22 Connecting from an application . . . 293
4.22.1 DNS 解決 . . . 293
4.22.2 環境変数 . . . 294
4.22.3 シークレット . . . 294
4.23 Connection pooling . . . 295
4.23.1 アーキテクチャ . . . 295
4.23.2 クイックスタート . . . 296
4.23.3 リソースのライフサイクルのプール化 . . . 297
4.23.4 セキュリティ . . . 297
4.23.5 認証 . . . 298
4.23.6 ポッドテンプレート . . . 299
4.23.7 サービステンプレート . . . 301
4.23.8 高可用性 HA . . . 301
4.23.9 PgBouncer 構成オプション . . . 302
4.23.10 モニタリング . . . 304
4.23.11 ロギング . . . 313
4.23.12 接続の一時停止 . . . 313

4.23.13 制限事項 . . . 314
4.24 Replica clusters . . . 314
4.24.1 基本概念 . . . 315
4.24.2 レプリカクラスターのセットアップ . . . 317
4.24.3 分散トポロジ . . . 318
4.24.4 スタンドアロンレプリカクラスター . . . 322
4.24.5 遅延レプリカ . . . 325
4.25 Kubernetes Upgrade and Maintenance . . . 326
4.25.1 定期的なアップデートの重要性 . . . 327
4.25.2 クラスター内のメンテナンス操作 . . . 327
4.25.3 一時的な PostgreSQL クラスターの低下 . . . 327
4.25.4 ポッドの中断予算 . . . 327
4.25.5 開発またはテストに使用される PostgreSQL クラスター . . . 328
4.25.6 ノードのメンテナンスウィンドウ . . . 328
4.26 EDB Postgres ® AI for CloudNativePG™ Cluster Plugin . . . 330
4.26.1 インストール . . . 330
4.26.2 使用する . . . 333
4.26.3 K9s との統合 . . . 357
4.26.4 プラグインに必要な権限 . . . 357
4.27 Automated failover . . . 360
4.27.1 RTO と RPO のインパクト . . . 361
4.27.2 遅延フェイルオーバー . . . 361
4.27.3 Failover Quorum クォーラムベースのフェールオーバー . . . 362
4.28 Fencing . . . 367
4.28.1 インスタンスをフェンスする方法 . . . 367
4.28.2 フェンシングのリフティング方法 . . . 368
4.28.3 フェンシングの仕組み . . . 369
4.29 Declarative hibernation . . . 369
4.29.1 ハイバネーション . . . 369
4.29.2 リハイドレート . . . 370
4.30 PostGIS . . . 371
4.30.1 PostGIS クラスターに関する基本的な概念 . . . 371
4.30.2 PostGIS で新しい PostgreSQL クラスターを作成する . . . 372
4.31 Container Image Requirements . . . 374
4.31.1 イメージタグの要件 . . . 375
4.32 Image Volume Extensions . . . 376
4.32.1 メリット . . . 376
4.32.2 要件 . . . 377
4.32.3 その仕組み . . . 377
4.32.4 新しい拡張機能を追加する方法 . . . 378
4.32.5 高度なトピック . . . 380
4.32.6 画像の仕様 . . . 382
4.32.7 注意事項 . . . 383
4.33 CNP-I . . . 383
4.33.1 なぜ CNP-I なのでしょうか？ . . . 383
4.33.2 プラグインの登録 . . . 384
4.33.3 プラグインを使用する . . . 387

4.33.4 コミュニティプラグイン . . . 387
4.34 Custom Pod Controller . . . 387
4.34.1 PVC のサイズ変更 . . . 388
4.34.2 プライマリインスタンスとレプリカ . . . 388
4.34.3 PVC のコヒーレンス . . . 389
4.34.4 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 . . . 389
4.35 Networking . . . 390
4.35.1 オペレーターのクロス名前空間ネットワークポリシー . . . 390
4.35.2 クロスクラスターネットワーキング . . . 390
4.36 Benchmarking . . . 391
4.36.1 pgbench . . . 391
4.36.2 fio . . . 393

第 5 章 EDB Enhancements 395
5.1 Free evaluation . . . 395
5.1.1 PostgreSQL を使用した評価 . . . 395
5.2 License and License keys . . . 395
5.2.1 会社レベルのライセンスキー . . . 395
5.2.2 クラスターレベルのライセンスキー . . . 396
5.2.3 クラスターレベルでのライセンスキーシークレット . . . 397
5.2.4 ライセンスの有効期限が切れるとどうなるか . . . 397
5.3 Red Hat OpenShift . . . 397
5.3.1 アーキテクチャ . . . 398
5.3.2 OpenShift の重要な概念 . . . 399
5.3.3 チャンネル . . . 401
5.3.4 Web コンソールを介したインストール . . . 402
5.3.5 oc CLI を介したインストール . . . 407
5.3.6 事前定義された RBAC オブジェクト . . . 410
5.3.7 ユーザーと権限 . . . 417
5.3.8 ポッドセキュリティ標準 . . . 418
5.3.9 プーラーイメージのカスタマイズ . . . 419
5.3.10 Velero の OADP . . . 420
5.3.11 モニタリングとメトリック . . . 420
5.4 Transparent Data Encryption (TDE) . . . 423
5.4.1 パスフレーズシークレット . . . 423
5.4.2 カスタムパスフレーズコマンド . . . 425
5.4.3 カスタム wrap/unwrap コマンド . . . 425
5.4.4 HashiCorp Vault を使用した例 . . . 426
5.5 Add-ons . . . 428
5.5.1 外部バックアップアダプター . . . 429
5.5.2 カステン . . . 437
5.5.3 ヴェレロ . . . 438

第 6 章 Reference 441
6.1 Operator capability levels . . . 441
6.1.1 レベル 1 基本インストール . . . 441
6.1.2 レベル 2 シームレスなアップグレード . . . 446

6.1.3 レベル 3 完全なライフサイクル . . . 448
6.1.4 レベル 4 深い洞察 . . . 453
6.1.5 レベル 5 自動操縦 . . . 454
6.2 Frequently Asked Questions (FAQ) . . . 454
6.2.1 Kubernetes での PostgreSQL の実行 . . . 454
6.2.2 高可用性 . . . 457
6.2.3 データベース管理 . . . 458
6.3 Troubleshooting . . . 461
6.3.1 始める前に . . . 461
6.3.2 最初の一歩 . . . 462
6.3.3 バックアップはありますか？ . . . 462
6.3.4 緊急バックアップ . . . 462
6.3.5 ログ . . . 463
6.3.6 オペレーター情報 . . . 464
6.3.7 クラスター情報 . . . 466
6.3.8 ポッド情報 . . . 466
6.3.9 バックアップ情報 . . . 469
6.3.10 ストレージ情報 . . . 469
6.3.11 ノード情報 . . . 469
6.3.12 条件 . . . 470
6.3.13 ネットワーキング . . . 471
6.3.14 PostgreSQL コアダンプ . . . 471
6.3.15 いくつかの既知の問題 . . . 472
6.4 API Reference . . . 476
6.4.1 Resource Types . . . 476
6.4.2 Backup . . . 476
6.4.3 Cluster . . . 476
6.4.4 ClusterImageCatalog . . . 477
6.4.5 Database . . . 477
6.4.6 FailoverQuorum . . . 478
6.4.7 ImageCatalog . . . 478
6.4.8 Pooler . . . 478
6.4.9 Publication . . . 479
6.4.10 ScheduledBackup . . . 479
6.4.11 Subscription . . . 479
6.4.12 AffinityConfiguration . . . 480
6.4.13 AvailableArchitecture . . . 481
6.4.14 BackupConfiguration . . . 481
6.4.15 BackupMethod . . . 482
6.4.16 BackupPhase . . . 482
6.4.17 BackupPluginConfiguration . . . 482
6.4.18 BackupSnapshotElementStatus . . . 482
6.4.19 BackupSnapshotStatus . . . 483
6.4.20 BackupSource . . . 483
6.4.21 BackupSpec . . . 484
6.4.22 BackupStatus . . . 484
6.4.23 BackupTarget . . . 485

6.4.24 BootstrapConfiguration . . . 486
6.4.25 BootstrapInitDB . . . 486
6.4.26 BootstrapPgBaseBackup . . . 488
6.4.27 BootstrapRecovery . . . 488
6.4.28 CatalogImage . . . 489
6.4.29 CertificatesConfiguration . . . 490
6.4.30 CertificatesStatus . . . 491
6.4.31 ClusterMonitoringTLSConfiguration . . . 491
6.4.32 ClusterSpec . . . 491
6.4.33 ClusterStatus . . . 495
6.4.34 ConfigMapResourceVersion . . . 498
6.4.35 DataDurabilityLevel . . . 498
6.4.36 DataSource . . . 498
6.4.37 DatabaseObjectSpec . . . 499
6.4.38 DatabaseObjectStatus . . . 499
6.4.39 DatabaseReclaimPolicy . . . 499
6.4.40 DatabaseRoleRef . . . 499
6.4.41 DatabaseSpec . . . 500
6.4.42 DatabaseStatus . . . 502
6.4.43 EPASConfiguration . . . 502
6.4.44 EmbeddedObjectMetadata . . . 502
6.4.45 EnsureOption . . . 503
6.4.46 EphemeralVolumesSizeLimitConfiguration . . . 503
6.4.47 ExtensionConfiguration . . . 503
6.4.48 ExtensionSpec . . . 503
6.4.49 ExternalCluster . . . 504
6.4.50 FailoverQuorumStatus . . . 505
6.4.51 ImageCatalogRef . . . 505
6.4.52 ImageCatalogSpec . . . 505
6.4.53 ImageInfo . . . 506
6.4.54 Import . . . 506
6.4.55 ImportSource . . . 507
6.4.56 InstanceID . . . 507
6.4.57 InstanceReportedState . . . 507
6.4.58 IsolationCheckConfiguration . . . 507
6.4.59 LDAPBindAsAuth . . . 508
6.4.60 LDAPBindSearchAuth . . . 508
6.4.61 LDAPConfig . . . 508
6.4.62 LDAPScheme . . . 509
6.4.63 LivenessProbe . . . 509
6.4.64 ManagedConfiguration . . . 509
6.4.65 ManagedRoles . . . 510
6.4.66 ManagedService . . . 510
6.4.67 ManagedServices . . . 510
6.4.68 Metadata . . . 511
6.4.69 MonitoringConfiguration . . . 511
6.4.70 NodeMaintenanceWindow . . . 512

6.4.71 OnlineConfiguration . . . 513
6.4.72 PasswordState . . . 513
6.4.73 PgBouncerIntegrationStatus . . . 514
6.4.74 PgBouncerPoolMode . . . 514
6.4.75 PgBouncerSecrets . . . 514
6.4.76 PgBouncerSpec . . . 514
6.4.77 PluginConfiguration . . . 515
6.4.78 PluginStatus . . . 515
6.4.79 PodTemplateSpec . . . 516
6.4.80 PodTopologyLabels . . . 516
6.4.81 PoolerIntegrations . . . 517
6.4.82 PoolerMonitoringConfiguration . . . 517
6.4.83 PoolerSecrets . . . 517
6.4.84 PoolerSpec . . . 518
6.4.85 PoolerStatus . . . 518
6.4.86 PoolerType . . . 519
6.4.87 PostgresConfiguration . . . 519
6.4.88 PrimaryUpdateMethod . . . 520
6.4.89 PrimaryUpdateStrategy . . . 520
6.4.90 Probe . . . 520
6.4.91 ProbeStrategyType . . . 521
6.4.92 ProbeWithStrategy . . . 521
6.4.93 ProbesConfiguration . . . 522
6.4.94 PublicationReclaimPolicy . . . 522
6.4.95 PublicationSpec . . . 522
6.4.96 PublicationStatus . . . 523
6.4.97 PublicationTarget . . . 523
6.4.98 PublicationTargetObject . . . 523
6.4.99 PublicationTargetTable . . . 524
6.4.100 RecoveryTarget . . . 524
6.4.101 ReplicaClusterConfiguration . . . 525
6.4.102 ReplicationSlotsConfiguration . . . 525
6.4.103 ReplicationSlotsHAConfiguration . . . 526
6.4.104 RoleConfiguration . . . 526
6.4.105 SQLRefs . . . 528
6.4.106 ScheduledBackupSpec . . . 528
6.4.107 ScheduledBackupStatus . . . 529
6.4.108 SchemaSpec . . . 530
6.4.109 SecretVersion . . . 530
6.4.110 SecretsResourceVersion . . . 530
6.4.111 ServiceAccountTemplate . . . 531
6.4.112 ServiceSelectorType . . . 531
6.4.113 ServiceTemplateSpec . . . 532
6.4.114 ServiceUpdateStrategy . . . 532
6.4.115 SnapshotOwnerReference . . . 532
6.4.116 SnapshotType . . . 532
6.4.117 StorageConfiguration . . . 533

6.4.118 SubscriptionReclaimPolicy . . . 533
6.4.119 SubscriptionSpec . . . 533
6.4.120 SubscriptionStatus . . . 534
6.4.121 SwitchReplicaClusterStatus . . . 534
6.4.122 SyncReplicaElectionConstraints . . . 535
6.4.123 SynchronizeReplicasConfiguration . . . 535
6.4.124 SynchronousReplicaConfiguration . . . 535
6.4.125 SynchronousReplicaConfigurationMethod . . . 536
6.4.126 TDEConfiguration . . . 536
6.4.127 TablespaceConfiguration . . . 537
6.4.128 TablespaceState . . . 537
6.4.129 TablespaceStatus . . . 538
6.4.130 Topology . . . 538
6.4.131 VolumeSnapshotConfiguration . . . 538
6.5 CNCF Projects Integrations . . . 539
6.5.1 External Secrets . . . 539
6.5.2 Cilium . . . 544

第 7 章 Appendix **551**
7.1 Appendix C - Common object stores for backups . . . 551
7.1.1 AWS S3 . . . 551
7.1.2 Azure Blob Storage . . . 554
7.1.3 Google Cloud Storage . . . 556
7.2 PostgreSQL Database Management . . . 558
7.2.1 主要なコンセプト . . . 558
7.2.2 予約されたデータベース名 . . . 559
7.2.3 調整とステータス . . . 559
7.2.4 データベースの削除 . . . 560
7.2.5 データベース内の拡張機能の管理 . . . 561
7.2.6 データベースでのスキーマの管理 . . . 562
7.2.7 制限と注意事項 . . . 563
7.3 Image Catalog . . . 564
7.3.1 CloudNativePG™クラスターカタログ用の EDB Postgres ® AI . . . 566
7.4 Iron Bank . . . 566
7.4.1 Iron Bank から EDB {{name.abbr}}とオペランドイメージをプルする . . . 567
7.4.2 Iron Bank イメージを使用した{{name.abbr}}オペレーターのインストール . . . 567
7.4.3 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 . . . 568
7.5 Logical Replication . . . 569
7.5.1 出版物 . . . 570
7.5.2 サブスクリプション . . . 572
7.5.3 制限事項 . . . 575
7.5.4 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 575
7.6 PostgreSQL Upgrades . . . 578
7.6.1 マイナーバージョンのアップグレード . . . 578
7.6.2 メジャーバージョンのアップグレード . . . 579
7.6.3 オフラインインプレースメジャーアップグレード . . . 579
7.7 Preview Versions . . . 583

7.7.1 リリース候補の目的 . . . 583
7.7.2 コミュニティへの参加 . . . 583
7.7.3 使用アドバイザリー . . . 583
7.7.4 現在のプレビューバージョン . . . 584
7.8 EDB private container registries . . . 584
7.8.1 どのリポジトリを選択しますか？ . . . 584
7.8.2 トークンの取得方法 . . . 585
7.8.3 オペランドイメージ . . . 585
7.8.4 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 . . . 586
7.8.5 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 . . . 586
7.9 Service Management . . . 587
7.9.1 デフォルトサービスの無効化 . . . 588
7.9.2 独自のサービスの追加 . . . 588
7.10 Tablespaces . . . 590
7.10.1 宣言的テーブルスペース . . . 590
7.10.2 宣言的テーブルスペースの使用 . . . 591
7.10.3 ストレージクラスとテーブルスペース . . . 592
7.10.4 テーブルスペースの所有権 . . . 593
7.10.5 バックアップ リカバリー . . . 594
7.10.6 レプリカクラスター . . . 595
7.10.7 一時テーブルスペース . . . 595
7.10.8 kubectl プラグインのサポート . . . 596
7.10.9 制限事項 . . . 596

第1章　Introduction and Release Notes

1.1 Key features, long time support and licensing

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。

1.1.1 CloudNativePG™と共通の主な機能

- 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

1.1.2 Kubernetes の EDB Postgres に固有の機能

- 長期サポート
- 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™クラスターで 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。

長期サポート

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。1.25 が現在の LTS バージョンです。1.18 が最初でした。各 LTS バージョンはメンテナンスリリースを受け取り、同じバージョンの CloudNativePG の最後のコミュニティリリースからさらに 12 か月間サポートされます。

たとえば、CloudNativePG の 1.22 リリースは、オープンソースコミュニティにおいて、2024 年 7 月 24 日にサポート終了になりました。CloudNativePG™クラスターの EDB Postgres ® AI の LTS バージョンとして宣言されたため、2025 年 7 月 24 日までさらに 12 か月間サポートされます。

さらに、顧客は LTS バージョン間を移動するために常に少なくとも 6 か月があります。たとえば、1.25 LTS バージョンは 2024 年 12 月 23 日にリリースされ、ユーザーが 2025 年 7 月のサポート終了に先立って 1.22LTS から移行するための十分な時間を与えました。

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。

1.1.3 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 Limited Use License 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™クラスターで 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。

1.1.4 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。

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。

1.1.5 サポートされている Postgres バージョン

Postgres の次のバージョンは、現在バージョン 1.25 でサポートされています。

- PostgreSQL 13〜17
- EDB Postgres Advanced 13 - 17
- EDB Postgres 拡張 13〜17

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。

1.1.6 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。

- Postgres, PostgreSQL and the Slonik Logo

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。*

1.2 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
1.27.0 または 1.26.1 へのアップグレード	19 Aug 2025	Upstream *1.27.0 または 1.26.1 へのアップグレード*
1.27.0 または 1.26.1 へのアップグレード	25 Jul 2025	Upstream *1.27.0 または 1.26.1 へのアップグレード*
EDB CloudNativePG Cluster 1.26.0 release notes	23 May 2025	Upstream *EDB CloudNativePG Cluster 1.26.0 release notes*
EDB CloudNativePG Cluster 1.25.3 release notes	25 Jul 2025	Upstream *EDB CloudNativePG Cluster 1.25.3 release notes*
EDB CloudNativePG Cluster 1.25.2 release notes	23 May 2025	Upstream *EDB CloudNativePG Cluster 1.25.2 release notes*
EDB Postgres for Kubernetes 1.25.1 release notes	04 Mar 2025	Upstream *EDB Postgres for Kubernetes 1.25.1 release notes*
EDB Postgres for Kubernetes 1.25.0 release notes	23 Dec 2025	Upstream *EDB Postgres for Kubernetes 1.25.0 release notes*
EDB CloudNativePG Cluster 1.24.4 release notes	23 May 2025	Upstream *EDB CloudNativePG Cluster 1.24.4 release notes*
EDB Postgres for Kubernetes 1.24.3 release notes	04 Mar 2025	Upstream *EDB Postgres for Kubernetes 1.24.3 release notes*
EDB Postgres for Kubernetes 1.24.2 release notes	23 Dec 2024	Upstream *EDB Postgres for Kubernetes 1.24.2 release notes*
EDB Postgres for Kubernetes 1.24.1 release notes	18 Oct 2024	Upstream *EDB Postgres for Kubernetes 1.24.1 release notes*
EDB Postgres for Kubernetes 1.24.0 release notes	26 Aug 2024	Upstream *EDB Postgres for Kubernetes 1.24.0 release notes*
EDB Postgres for Kubernetes 1.23.6 release notes	23 Dec 2024	Upstream *EDB Postgres for Kubernetes 1.23.6 release notes*
EDB Postgres for Kubernetes 1.23.5 release notes	18 Oct 2024	Upstream *EDB Postgres for Kubernetes 1.23.5 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 Postgres for Kubernetes 1.23.4 release notes	26 Aug 2024	Upstream *EDB Postgres for Kubernetes 1.23.4 release notes*
EDB Postgres for Kubernetes 1.23.3 release notes	01 Aug 2024	Upstream *EDB Postgres for Kubernetes 1.23.3 release notes*
EDB Postgres for Kubernetes 1.23.2 release notes	13 Jun 2024	Upstream *EDB Postgres for Kubernetes 1.23.2 release notes*
EDB Postgres for Kubernetes 1.23.1 release notes	29 Apr 2024	Upstream *EDB Postgres for Kubernetes 1.23.1 release notes*
EDB Postgres for Kubernetes 1.23.0 release notes	24 Apr 2024	Upstream *EDB Postgres for Kubernetes 1.23.0 release notes*
EDB CloudNativePG Cluster 1.22.11 release notes	25 Jul 2025	None
EDB CloudNativePG Cluster 1.22.10 release notes	23 May 2025	None
archive_timeout	04 Mar 2025	None
archive_timeout	23 Dec 2024	None
archive_timeout	18 Oct 2024	None
EDB Postgres for Kubernetes 1.22.6 release notes	26 Aug 2024	Upstream *EDB Postgres for Kubernetes 1.22.6 release notes*
EDB Postgres for Kubernetes 1.22.5 release notes	01 Aug 2024	Upstream *EDB Postgres for Kubernetes 1.22.5 release notes*
EDB Postgres for Kubernetes 1.22.4 release notes	13 Jun 2024	Upstream *EDB Postgres for Kubernetes 1.22.4 release notes*
EDB Postgres for Kubernetes 1.22.3 release notes	24 Apr 2024	Upstream *EDB Postgres for Kubernetes 1.22.3 release notes*
EDB Postgres for Kubernetes 1.22.2 release notes	22 Mar 2024	Upstream *EDB Postgres for Kubernetes 1.22.2 release notes*
EDB CloudNativePG Cluster 1.22.10 release notes	02 Feb 2024	Upstream *EDB CloudNativePG Cluster 1.22.10 release notes*
EDB Postgres for Kubernetes 1.22.0 release notes	22 Dec 2023	Upstream *EDB Postgres for Kubernetes 1.22.0 release notes*
EDB Postgres for Kubernetes 1.21.6 release notes	13 Jun 2024	Upstream *EDB Postgres for Kubernetes 1.21.6 release notes*
EDB Postgres for Kubernetes 1.21.5 release notes	24 Apr 2024	Upstream *EDB Postgres for Kubernetes 1.21.5 release notes*
EDB Postgres for Kubernetes 1.21.4 release notes	22 Mar 2024	Upstream *EDB Postgres for Kubernetes 1.21.4 release notes*
EDB Postgres for Kubernetes 1.21.3 release notes	02 Feb 2024	Upstream *EDB Postgres for Kubernetes 1.21.3 release notes*
EDB Postgres for Kubernetes 1.21.2 release notes	22 Dec 2023	Upstream *EDB Postgres for Kubernetes 1.21.2 release notes*
EDB Postgres for Kubernetes 1.21.1 release notes	08 Nov 2023	Upstream *EDB Postgres for Kubernetes 1.21.1 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 Postgres for Kubernetes 1.21.0 release notes	18 Oct 2023	Upstream *EDB Postgres for Kubernetes 1.21.0 release notes*
EDB Postgres for Kubernetes 1.20.6 release notes	02 Feb 2024	Upstream *EDB Postgres for Kubernetes 1.20.6 release notes*
EDB Postgres for Kubernetes 1.20.5 release notes	22 Dec 2023	Upstream *EDB Postgres for Kubernetes 1.20.5 release notes*
EDB Postgres for Kubernetes 1.20.4 release notes	08 Nov 2023	Upstream *EDB Postgres for Kubernetes 1.20.4 release notes*
EDB Postgres for Kubernetes 1.20.3 release notes	18 Oct 2023	Upstream *EDB Postgres for Kubernetes 1.20.3 release notes*
EDB Postgres for Kubernetes 1.20.2 release notes	27 Jul 2023	Upstream *EDB Postgres for Kubernetes 1.20.2 release notes*
EDB Postgres for Kubernetes 1.20.1 release notes	13 Jun 2023	Upstream *EDB Postgres for Kubernetes 1.20.1 release notes*
EDB Postgres for Kubernetes 1.20.0 release notes	27 Apr 2023	Upstream *EDB Postgres for Kubernetes 1.20.0 release notes*
EDB Postgres for Kubernetes 1.19.6 release notes	08 Nov 2023	Upstream *EDB Postgres for Kubernetes 1.19.6 release notes*
EDB Postgres for Kubernetes 1.19.5 release notes	18 Oct 2023	Upstream *EDB Postgres for Kubernetes 1.19.5 release notes*
EDB Postgres for Kubernetes 1.19.4 release notes	27 Jul 2023	Upstream *EDB Postgres for Kubernetes 1.19.4 release notes*
EDB Postgres for Kubernetes 1.19.3 release notes	13 Jun 2023	Upstream *EDB Postgres for Kubernetes 1.19.3 release notes*
EDB Postgres for Kubernetes 1.19.2 release notes	27 Apr 2023	Upstream *EDB Postgres for Kubernetes 1.19.2 release notes*
EDB Postgres for Kubernetes 1.19.1 release notes	20 Mar 2023	Upstream *EDB Postgres for Kubernetes 1.19.1 release notes*
EDB Postgres for Kubernetes 1.19.0 release notes	14 Feb 2023	Upstream *EDB Postgres for Kubernetes 1.19.0 release notes*
archive_timeout	13 Jun 2024	None
archive_timeout	24 Apr 2024	None
archive_timeout	22 Mar 2024	None
archive_timeout	02 Feb 2024	None
archive_timeout	22 Dec 2023	None
archive_timeout	08 Nov 2023	None
archive_timeout	18 Oct 2023	None
archive_timeout	27 Jul 2023	None
EDB Postgres for Kubernetes 1.18.5 release notes	13 Jun 2023	Upstream *EDB Postgres for Kubernetes 1.18.5 release notes*
EDB Postgres for Kubernetes 1.18.4 release notes	27 Apr 2023	Upstream *EDB Postgres for Kubernetes 1.18.4 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 Postgres for Kubernetes 1.18.3 release notes	20 Mar 2023	Upstream *EDB Postgres for Kubernetes 1.18.3 release notes*
EDB Postgres for Kubernetes 1.18.2 release notes	14 Feb 2023	Upstream *EDB Postgres for Kubernetes 1.18.2 release notes*
EDB Postgres for Kubernetes 1.18.10 release notes	21 Dec 2022	Upstream *EDB Postgres for Kubernetes 1.18.10 release notes*
EDB Postgres for Kubernetes 1.18.0 release notes	14 Nov 2022	Upstream *EDB Postgres for Kubernetes 1.18.0 release notes*
EDB Postgres for Kubernetes 1.17.5 release notes	20 Mar 2023	Upstream *EDB Postgres for Kubernetes 1.17.5 release notes*
EDB Postgres for Kubernetes 1.17.4 release notes	14 Feb 2023	Upstream *EDB Postgres for Kubernetes 1.17.4 release notes*
EDB Postgres for Kubernetes 1.17.3 release notes	21 Dec 2022	Upstream *EDB Postgres for Kubernetes 1.17.3 release notes*
EDB Postgres for Kubernetes 1.17.2 release notes	14 Nov 2022	Upstream *EDB Postgres for Kubernetes 1.17.2 release notes*
EDB Postgres for Kubernetes 1.17.1 release notes	07 Oct 2022	Upstream *EDB Postgres for Kubernetes 1.17.1 release notes*
EDB Postgres for Kubernetes 1.17.0 release notes	06 Sep 2022	Upstream *EDB Postgres for Kubernetes 1.17.0 release notes*
EDB Postgres for Kubernetes 1.16.5 release notes	21 Dec 2022	Upstream *EDB Postgres for Kubernetes 1.16.4 release notes*
EDB Postgres for Kubernetes 1.16.4 release notes	14 Nov 2022	Upstream *EDB Postgres for Kubernetes 1.16.4 release notes*
EDB Postgres for Kubernetes 1.16.3 release notes	07 Oct 2022	Upstream *EDB Postgres for Kubernetes 1.16.3 release notes*
EDB Postgres for Kubernetes 1.16.2 release notes	06 Sep 2022	Upstream *EDB Postgres for Kubernetes 1.16.2 release notes*
EDB Postgres for Kubernetes 1.16.1 release notes	12 Aug 2022	Upstream *EDB Postgres for Kubernetes 1.16.1 release notes*
EDB Postgres for Kubernetes 1.16.0 release notes	07 Jul 2022	Upstream *EDB Postgres for Kubernetes 1.16.0 release notes*
EDB Postgres for Kubernetes 1.15.5 release notes	07 Oct 2022	Upstream *EDB Postgres for Kubernetes 1.15.5 release notes*
EDB Postgres for Kubernetes 1.15.4 release notes	06 Sep 2022	Upstream *EDB Postgres for Kubernetes 1.15.4 release notes*
EDB Postgres for Kubernetes 1.15.3 release notes	12 Aug 2022	Upstream *EDB Postgres for Kubernetes 1.15.3 release notes*
EDB Postgres for Kubernetes 1.15.2 release notes	07 Jul 2022	Upstream *EDB Postgres for Kubernetes 1.15.2 release notes*
EDB Postgres for Kubernetes 1.15.1 release notes	27 May 2022	Upstream *EDB Postgres for Kubernetes 1.15.1 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 Postgres for Kubernetes 1.15.0 release notes	21 Apr 2022	Upstream *EDB Postgres for Kubernetes 1.15.0 release notes*
archive_timeout	25 Mar 2022	NA
archive_timeout	17 Feb 2022	NA
archive_timeout	11 Jan 2022	NA
archive_timeout	15 Dec 2021	NA
archive_timeout	11 Nov 2021	NA
archive_timeout	15 Oct 2021	NA
archive_timeout	30 Sep 2021	NA
archive_timeout	28 Sep 2021	NA
archive_timeout	13 Sep 2021	NA
archive_timeout	11 Aug 2021	NA
archive_timeout	28 Jul 2021	NA
archive_timeout	12 Jul 2021	NA
archive_timeout	11 Jun 2021	NA
archive_timeout	17 Jun 2021	NA
archive_timeout	18 May 2021	NA
archive_timeout	23 Apr 2021	NA
archive_timeout	06 Apr 2021	NA
archive_timeout	31 Mar 2021	NA
archive_timeout	03 Mar 2021	NA
archive_timeout	04 Feb 2021	NA
archive_timeout	29 Jan 2021	NA
archive_timeout	31 Dec 2020	NA
archive_timeout	04 Dec 2020	NA
archive_timeout	20 Nov 2020	NA
archive_timeout	05 Nov 2020	NA
archive_timeout	25 Sep 2020	NA
archive_timeout	11 Aug 2020	NA
archive_timeout	03 Apr 2020	NA
archive_timeout	05 Mar 2020	NA

1.2.1 EDB CloudNativePG Cluster 1.27.0 release notes

リリース日 2025 年 8 月 19 日

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
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

1.2.2 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。

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 *Introduction and Release Notes* を参照してください。	

1.2.3 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。

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 *Introduction and Release Notes* を参照してください。	

1.2.4 EDB CloudNativePG Cluster 1.25.3 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。

の変更

Description	Addresses
コミュニティ CloudNativePG 1.25.3 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 *Introduction and Release Notes* を参照してください。	

1.2.5 EDB CloudNativePG Cluster 1.25.2 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。

の変更

Description	Addresses
コミュニティ CloudNativePG 1.25.2 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 *Introduction and Release Notes* を参照してください。	

1.2.6 EDB Postgres for Kubernetes 1.25.1 release notes

リリース日 2025 年 3 月 4 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.25.1 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.7 EDB Postgres for Kubernetes 1.25.0 release notes

リリース日 2024 年 12 月 23 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.25.0 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.8 EDB CloudNativePG Cluster 1.24.4 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。

の変更

Description	Addresses
コミュニティ CloudNativePG 1.24.4 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 *Introduction and Release Notes* を参照してください。	

1.2.9 EDB Postgres for Kubernetes 1.24.3 release notes

リリース日 2025 年 3 月 4 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.24.3 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.10 EDB Postgres for Kubernetes 1.24.2 release notes

リリース日 2024 年 12 月 23 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.24.2 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.11 EDB Postgres for Kubernetes 1.24.1 release notes

リリース日 2024 年 10 月 18 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.24.1 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.12 EDB Postgres for Kubernetes 1.24.0 release notes

リリース日 2024 年 8 月 26 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.24.0 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.13 EDB Postgres for Kubernetes 1.23.6 release notes

リリース日 2024 年 12 月 23 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.23.6 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.14 EDB Postgres for Kubernetes 1.23.5 release notes

リリース日 2024 年 10 月 18 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.23.5 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.15 EDB Postgres for Kubernetes 1.23.4 release notes

リリース日 2024 年 8 月 26 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.23.4 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.16 EDB Postgres for Kubernetes 1.23.3 release notes

リリース日 2024 年 8 月 1 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.23.3 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.17 EDB Postgres for Kubernetes 1.23.2 release notes

リリース日 2024 年 6 月 13 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.23.2 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.18 EDB Postgres for Kubernetes 1.23.1 release notes

リリース日 2024 年 4 月 29 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.23.1 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.19 EDB Postgres for Kubernetes 1.23.0 release notes

リリース日 2024 年 4 月 24 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.23.0 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.20 EDB CloudNativePG Cluster 1.22.11 release notes

リリース日 2025 年 7 月 25 日

EDB Postgres for Kubernetes バージョン 1.22.11 は、 EDB Postgres for Kubernetes の LTS リリースです。CloudNativePG の対応するアップストリームリリースはありません。

- DJ Walker-Morgan を偲んで .*

> **警告**
>
> これは、1.22.x シリーズの最後のリリースになる予定です。 1.22 はサポートされなくなったため、新しいマイナー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。

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。

機能強化

Description	Addresses
`HugePages`設定を正しく考慮することにより`shared_buffers`の検証が向上し、正確なメモリ構成チェックを保証します。	#7864
PostgreSQL ワーカープロセスの`oom_score_adj`を設定して、メモリ不足の場合の優先順位付けを向上させます。	#7891
PostgreSQL システム識別子を追跡するために、`Cluster`ステータスに`systemID`フィールドと関連条件を追加しました。	#7717

の変更

Description	Addresses
`cnp`プラグインビルドマトリックスから`386`および ARM v5/v6/v7 アーキテクチャを削除し、公開されたバイナリの数を削減しました。	#7648

バグ修正

Description	Addresses
接続プーラーにミューテックスを追加して、接続マップへの同時アクセスを保護し、高同時実行環境での安定性を向上させます。	#7804

1.2.21 EDB CloudNativePG Cluster 1.22.10 release notes

リリース日 2025 年 5 月 23 日

EDB Postgres for Kubernetes バージョン 1.22.10 は、 EDB Postgres for Kubernetes の LTS リリースです。CloudNativePG の対応するアップストリームリリースはありません。

> **警告**
>
> 新しい LTS リリースバージョン 1.25.2 が利用可能になりました。 1.22 のサポートは 2025 年 7 月に正式に終了することにノートてください。1.25 LTS リリースにスムーズに移行するために、今すぐアップグレード計画を開始することをお勧めします。

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。

機能強化

Description	Addresses
`k8s.enterprisedb.io/validation`アノテーションを実装し、ユーザーが Kubernetes 管理リソースの EDB Postgres で検証 Webhook を無効にできるようにします。無制限の変更が許可されるため、使用には注意してください。	#7196
PostgreSQL 18 での`pg_stat_wal`メトリックの収集のサポートを追加しました	#7005
Barman Cloud **非推奨** を介して WAL ファイルをアーカイブする場合、LZ4、XZ、および Z 標準圧縮方法のサポートを追加しました。	#7151

セキュリティ修正

Description	Addresses
オペレーター展開の `imagePullPolicy`から `Always`を設定して、イメージが常にレジストリから取得されるようにし、古いまたは安全でない可能性のあるローカルイメージを使用するリスクを削減します。	#7250

の変更

Description	Addresses
新しいクラスター定義に合わせて、デフォルトの PostgreSQL バージョンを 17.5 に更新しました。	#7556
新しい `Pooler` 展開のデフォルトの PgBouncer バージョンを **1.24.1** に更新しました。	#7399

バグ修正

Description	Addresses
レプリカ構成で `dbname` パラメーターを `primary_conninfo` に追加することにより、PostgreSQL 17 のネイティブレプリケーションスロット同期と論理レプリケーションフェイルオーバーを修正しました。	#7298
WAL アーカイブにフェイルファストメカニズムを導入することにより、バックアップ効率が向上し、予期しないプライマリ降格を迅速に検出し、不必要な再試行を回避できます。	#7483
要求された数を超えて 1 つの追加のワーカープロセスが生成される可能性があるパラレル WAL アーカイブの off-by-one エラーを修正しました。	#7389
PostgreSQL 構成を更新するときにオペレーターが 2 つのスイッチオーバーを実行する原因となった競合状態を解決しました。	#6991
ターゲットとされたプーラーおよびクラスターポッドの `matchLabels` スコープを調整することにより、`PodMonitor` 構成を修正しました。以前は、`matchLabels` の範囲が広すぎ、誤ってクラスターからラベルを継承し、意図しないターゲットからデータ収集が発生していました。	#7063
`shared_buffers` 構成で欠落している単位 MB、GB などのクラスターの Webhook 警告を追加しました。これは、将来のリリースではエラーになります。ユーザーは構成を更新して、明示的なユニットたとえば、`512` の代わりに `512MB` などを含める必要があります。	#7160
`cnp` プラグイン `primaryUpdateMethod` が `restart` に設定されている場合、強制的再起動中にプライマリポッドが再作成され、その定義をレプリカと調整するようにしました。	#7122

1.2.22 EDB Postgres for Kubernetes 1.22.9 release notes

リリース日 2025 年 3 月 4 日

EDB Postgres for Kubernetes バージョン 1.22.9 は、 EDB Postgres for Kubernetes の LTS リリースです。CloudNativePG の対応するアップストリームリリースはありません。

> **警告**
>
> 新しい LTS リリースバージョン 1.25.1 が利用可能になりました。 1.22 のサポートは 2025 年 7 月に正式に終了することにノートてください。1.25 LTS リリースにスムーズに移行するために、今すぐアップグレード計画を開始することをお勧めします。

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

機能強化

- オペレーターのスタートアッププローブを導入して、信頼性を向上させ、初期化中の早期の活性プローブの障害を防止します。#7008
- プーラーでの-r サービスを使用するためのサポートを追加しました。#6868
- pgbench プラグインのオプションの--ttl フラグを導入し、ユーザー定義の期間の後に完了したジョブの自動削除を有効にします。#6701
- 新しいクラスター定義に向けて、デフォルトの PostgreSQL バージョンを 17.4 に更新しました。#6960

セキュリティ

- オペレーターイメージビルドプロセスが強化され、セキュリティと透明性が強化されました。イメージは cosign で署名され、ソフトウェア部品表 SBOM と来歴データを組み込んだ OCI 証明書が生成されます。さらに、OCI アノテーションが追加されて、トレーサビリティを向上させ、イメージの完全性を保証します。

バグ修正

- .spec.probes が定義されている場合のデフォルトのプローブノブ値の一貫性のない動作を修正し、ユーザーが failureThreshold を含むすべての設定をオーバーライドできるようにします。起動プローブで指定されない場合、failureThreshold は.spec.startupDelay / periodSeconds デフォルト 10 、オーバーライド可能になりました。同じロジックが.spec.livenessProbeTimeout を介して liveness プローブに適用されます。 #6656
- インスタンスの再起動後、構成変更が適用されるまで、WAL メトリックを利用できない問題を修正しました。#6816
- モノリシックデータベースのインポートで、ロールが指定されていない場合にロールのインポートがスキップされる問題を修正しました。#6646
- PgBouncer 1.24 で導入された新しいメトリックのサポートを追加しました。#6630
- プライマリサーバー再起動後のタイムリーな調整を保証することにより、レプリケーション依存のパラメーター削減の処理が向上しました。#6440
- レプリカクラスターのブートストラップ中に override.conf が一貫して postgresql.conf に含まれるように保証、構成設定の欠落によるレプリケーション障害を防ぎます。#6808
- プライマリロールの変更中の障害を防ぐために、pg_rewind を呼び出す前に override.conf が正しく初期化されていることを確認しました。 #6670
- Webhook 応答を強化して、該当する場合は警告とエラーの両方を結果、診断の精度を向上させます。#6579
- オペレーターバージョンが正しくリコンサイルされていることを確認しました。#6496
- データディレクトリのより正確なチェックを使用することにより、PostgreSQL バージョンの検出が向上しました。#6659

- ボリュームスナップショットバックアップ
  - 未使用のバックアップ接続が適切にクリーンアップされない問題を修正しました。#6882
  - インスタンスマネージャーが、失敗したボリュームスナップショットバックアップによって残された古い PostgreSQL 接続を閉じるようにしました。#6879
  - 別のボリュームが既に進行中であるときに、オペレーターが新しいボリュームスナップショットバックアップを開始できないようにしました。#6890
- cnp プラグイン:
  - `promote` プラグインコマンドの機能を復元しました。 #6476
  - サイドカーを含むすべてのコンテナからログを収集するように `kubectl cnp report --logs <cluster>` を強化しました。 #6636
  - `Cluster` が `ImageCatalog` を使用するときに `pgbench` ジョブを実行できるようにしました。#6868

技術的な機能強化

- Kubernetes `client-gen` のサポートを追加し、Kubernetes CRD のすべての EDB Postgres の Go クライアントの自動生成を有効にします。#6695

1.2.23 EDB Postgres for Kubernetes 1.22.8 release notes

リリース日 2024 年 12 月 23 日

EDB Postgres for Kubernetes バージョン 1.22.8 は、 EDB Postgres for Kubernetes の LTS リリースです。CloudNativePG の対応するアップストリームリリースはありません。

> **警告**
>
> 新しい LTS リリースバージョン 1.25.0 が利用可能になりました。 1.22 のサポートは 2025 年 7 月に正式に終了することにノートてください。1.25 LTS リリースにスムーズに移行するために、今すぐアップグレード計画を開始することをお勧めします。

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

機能強化

- `.spec.probes` スタンザを介してスタートアップ、ライブネス、および準備状況プローブのカスタマイズを有効にします。#6266
- 事前定義されたユーザー、具体的には `superuser` および `app` 用に生成されたシークレットに `cnpg.io/userType` ラベルを追加します。 #4392
- ScheduledBackups の `spec.schedule` フィールドの検証が向上し、潜在的な構成ミスの警告が生成されます。#5396
- cnpg プラグイン - API サーバーとの HTTP 要求の `User-Agent` ヘッダーを尊重します。 #6153

バグ修正

- レプリカとして再同期する前に、元のプライマリが WAL ファイルキューをアーカイブにフラッシュすることにより、復旧時間を削減し、フェールオーバー中のデータの一貫性を強化します。#6141
- ブートストラップ中に PGDATA ボリュームとともに WAL ボリュームをクリーンアップします。 #6265
- 以前に生成された PersistentVolumeClaims がすべて欠落している場合、クラスターフェーズを Unrecoverable に設定するようにオペレーターを更新します。 #6170
- .spec.postgresql.synchronous.method が first に設定されている場合の synchronous_standby_names GUC の解析を修正しました。 #5955
- CRD ステータスの特定の条件にパッチを適用する際の潜在的な競合状態を解決し、同時更新の信頼性を向上させます。#6328
- ロールの変更を修正して、データベースコンテキストではなくトランザクションレベルで適用します。#6064
- プライマリ上の override.conf ファイルから primary_slot_name 定義を削除して、常に空になるようにします。 #6219
- PgBouncer ポッドで PGHOST を含む libpq 環境変数を構成して、コンテナ内から psql を使用して pgbouncer 仮想データベースへのシームレスなアクセスを有効にします。#6247
- 適切なバックアップ構成なしで外部サーバーからのリカバリー中のパニックを防止します。#6300
- 2 つの異なる値のログに名前フィールドが一貫して使用されない、構造化ログのキー衝突の問題を解決しました。#6324
- SQL ステートメントの inRoles フィールドの適切な引用符を確認して、ロール管理中に生成された SQL の構文エラーを防ぎます。#6346
- cnpg プラグイン - kubectl コンテキストが psql コマンドで適切に渡されることを確認します。 #6257 - status コマンドで空のときに物理バックアップブロックを表示しないようにします。#5998

1.2.24 EDB Postgres for Kubernetes 1.22.7 release notes

リリース日 2024 年 10 月 18 日

EDB Postgres for Kubernetes バージョン 1.22.7 は、 EDB Postgres for Kubernetes の LTS リリースです。CloudNativePG の対応するアップストリームリリースはありません。

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

機能強化

- PGDATA ディレクトリ全体をスキャンしてデータベースサイズを計算することにより、リソース使用率が高くなる原因となったため、ステータスプローブから pg_database_size の使用を削除します。kubectl status プラグインは、du に依存して、詳細なサイズ情報の取得#5689 を提供するようになりました。 - PostgreSQL で full_page_writes パラメーターを構成する機能を追加します。この設定は、PostgreSQL の推奨事項#5516 に従って、デフォルトの on です。 - プラグイン - cnp プラグインに logs pretty コマンドを追加して、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

形式で出力します。ログエントリをフィルタリングするオプションを使用します#5770 - 複数の-v オプションを許可することにより、status コマンドを強化します。より詳細な出力のために冗長性を高めます#5765 - pgadmin4 プラグインコマンドの--image フラグを使用してカスタム Docker イメージを指定するサポートを追加し、ユーザーが pgAdmin4 展開に使用される Docker イメージを制御できるようにします#5515

修正

- 2 つのインスタンスにスケールダウンするときにレプリカ PodDisruptionBudgetPDB が削除されるようにし、レプリカ#5487 をホストするノードのメンテナンスが簡単になります。- インプレースリスタートよりも完全なロールアウトを優先します#5407。 - 接続が失われた場合に二重フェイルオーバーを引き起こす可能性のある問題を修正します#5788。- ポッドとジョブの TMPDIR および PSQL_HISTORY 環境変数を正しく設定し、一時ファイルと履歴の管理を向上させます#5503。- プラグイン - logs cluster コマンド#5775 の競合状態を解決します。- potential の同期状態を status プラグイン #5533 に表示します。- pgadmin4 コマンドで展開されたポッドに書き込み可能なホームディレクトリ#5800 がなかった問題を修正します。

サポートされているバージョン

- PostgreSQL 17 PostgreSQL 17.0 がデフォルトイメージです。

1.2.25 EDB Postgres for Kubernetes 1.22.6 release notes

リリース日 2024 年 8 月 26 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

機能

- **ポッド中断バジェット PDB の構成** .spec.enablePDB フィールドを導入して、プライマリインスタンスで PDB を無効にし、メンテナンス操作中にポッドの適切なエビクションを可能にします。これは、単一インスタンスの展開に特に役立ちます。この機能は、ノードのメンテナンスウィンドウ機能を置き換えることを目的としています。

機能強化

- **cnp プラグインの更新**–control-plane オプションを追加することにより installgenerate コマンドを強化し、ノードのアフィニティと許容範囲を設定することによりコントロールプレーンノードにオペレーターを展開できます#5271。* destroy コマンドを強化して、ターゲットインスタンス#5298 に関連するジョブも削除します。

修正

- 同期レプリケーション自己修復チェックは、終了したポッドを除外し、アクティブで機能的なポッドにのみ焦点を当てます#5210。
- インスタンスマネージャーは、レプリカクラスター#5209 でのロールの変更に続いて、既存のすべてのオペレーター関連のレプリケーション接続を終了するようになりました。

- smartShutdownTimeout をゼロに設定できるようにし、即時高速シャットダウンを有効にし、必要に応じてスマートシャットダウンプロセスをバイパスします#5347。

1.2.26 EDB Postgres for Kubernetes 1.22.5 release notes

リリース日 2024 年 8 月 1 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.22.5 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.27 EDB Postgres for Kubernetes 1.22.4 release notes

リリース日 2024 年 6 月 13 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.22.4 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.28 EDB Postgres for Kubernetes 1.22.3 release notes

リリース日 2024 年 4 月 24 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.22.3 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.29 EDB Postgres for Kubernetes 1.22.2 release notes

リリース日 2024 年 3 月 22 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.22.2 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.30 EDB Postgres for Kubernetes 1.22.1 release notes

リリース日 2024 年 2 月 2 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.22.1 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.31 EDB Postgres for Kubernetes 1.22.0 release notes

リリース日 2023 年 12 月 22 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.22.0 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.32 EDB Postgres for Kubernetes 1.21.6 release notes

リリース日 2024 年 6 月 13 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.21.6 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.33 EDB Postgres for Kubernetes 1.21.5 release notes

リリース日 2024 年 4 月 23 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.21.5 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.34 EDB Postgres for Kubernetes 1.21.4 release notes

リリース日 2024 年 3 月 22 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.21.4 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.35 EDB Postgres for Kubernetes 1.21.3 release notes

リリース日 2024 年 2 月 2 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.21.3 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.36 EDB Postgres for Kubernetes 1.21.2 release notes

リリース日 2023 年 12 月 22 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.21.2 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.37 EDB Postgres for Kubernetes 1.21.1 release notes

リリース日 2023 年 11 月 8 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.21.1 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.38 EDB Postgres for Kubernetes 1.21.0 release notes

リリース日 2023 年 10 月 18 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.21.0 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.39 EDB Postgres for Kubernetes 1.20.6 release notes

リリース日 2024 年 2 月 2 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.20.6 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.40 EDB Postgres for Kubernetes 1.20.5 release notes

リリース日 2023 年 12 月 22 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.20.5 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.41 EDB Postgres for Kubernetes 1.20.4 release notes

リリース日 2023 年 11 月 8 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.20.4 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.42 EDB Postgres for Kubernetes 1.20.3 release notes

リリース日 2023 年 10 月 18 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.20.3 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.43 EDB Postgres for Kubernetes 1.20.2 release notes

リリース日 2023 年 7 月 27 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.20.2 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.44 EDB Postgres for Kubernetes 1.20.1 release notes

リリース日 2023 年 6 月 13 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.20.1 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.45 EDB Postgres for Kubernetes 1.20.0 release notes

リリース日 2023 年 4 月 27 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.20.0 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.46 EDB Postgres for Kubernetes 1.19.6 release notes

リリース日 2023 年 11 月 8 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.19.6 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.47 EDB Postgres for Kubernetes 1.19.5 release notes

リリース日 2023 年 10 月 18 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.19.5 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.48 EDB Postgres for Kubernetes 1.19.4 release notes

リリース日 2023 年 7 月 27 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.19.4 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.49 EDB Postgres for Kubernetes 1.19.3 release notes

リリース日 2023 年 6 月 13 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.19.3 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.50 EDB Postgres for Kubernetes 1.19.2 release notes

リリース日 2023 年 4 月 27 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.19.2 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.51 EDB Postgres for Kubernetes 1.19.1 release notes

リリース日 2023 年 3 月 20 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.19.1 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.52 EDB Postgres for Kubernetes 1.19.0 release notes

リリース日 2023 年 2 月 14 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.19.0 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。
Feature	EDB Postgres Advanced Server 15 での *Transparent Data Encryption (TDE)* のサポート TDE は、データベースシステムに保存されているユーザーデータをユーザーに対して透過的に暗号化します。
Feature	新しい外部バックアップアダプタは、バックアップ用のサードパーティツールに Kubernetes の EDB Postgres を統合する汎用方法を提供します。詳細については、 **外部バックアップアダプター** を参照してください。

1.2.53 EDB Postgres for Kubernetes 1.18.13 release notes

リリース日 2024 年 6 月 13 日

EDB Postgres for Kubernetes バージョン 1.18.13 は、 EDB Postgres for Kubernetes の LTS リリースです。CloudNativePG の対応するアップストリームリリースはありません。

> **警告**
>
> これは、1.18.X シリーズの最後のリリースになる予定です。ユーザーは、すぐに新しいマイナーバージョンに更新することをお勧めします。

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Enhancement	.spec.livenessProbeTimeout`オプションを介した liveness プローブタイムアウトの構成を有効にしました`#4719
Change	デフォルトのオペランドイメージを PostgreSQL 16.3 #4584 に設定しました
Change	名前空間オブジェクト #4753 のすべての RBAC 要件を削除しました。
Bug fix	フェンスされたインスタンスが不要なループに入り、使用可能なすべての CPU を消費するのを防ぎました #4625
Bug fix	プライマリのインスタンスマネージャーが、停止操作に続いて障害が発生した後、インスタンスが起動するのを無期限に待機する問題を解決しました #4434
Bug fix	ポッドコンテナのステータスが欠落しているときに発生したバックアップコントローラのパニックを修正しました #4765
Bug fix	インスタンスマネージャー #4670 の不必要なシャットダウンの防止
Bug fix	PostgreSQL 構成が変更されていない場合の不要なリロードを防止 #4531
Bug fix	コンテンツを書き込む一貫した一意の方法を保証することにより、ident マップの不必要なリロードを防止しました #4648
Bug fix	リソースのステータスを更新する代わりにパッチを適用することにより、フェーズ登録中の競合を回避しました #4637
Bug fix	PostgreSQL を再起動し、フェンシングをリフティングするときにタイムアウトを実装しました #4504
Bug fix	アーカイブモードを適切に設定するための昇格後にレプリカクラスターが再起動されるようにした #4399
Bug fix	pgaudit.log_rows`オプションが有効になっているときに返される追加行フィールドの正しい解析を保証し、監査ログが通常のログストリーム`#4394 に誤ってルーティングされるのを防ぎます

1.2.54 EDB Postgres for Kubernetes 1.18.12 release notes

リリース日 2024 年 4 月 24 日

EDB Postgres for Kubernetes バージョン 1.18.12 は、 EDB Postgres for Kubernetes の LTS リリースです。CloudNativePG の対応するアップストリームリリースはありません。

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Enhancement	1.18.x LTS から 1.22.x LTS へのアップグレードプロセスを追加しました

次のページに続く

表 58 – 前のページからの続き

Type	Description
Enhancement	Kubernetes 1.29.x 以降のドキュメント #3729
Enhancement	所有権のないポッドがクラスターのリストア中に削除されるように確認 #4141
Bug fix	レプリカで空の場合、LSN ソートを適切に処理する #4283
Bug fix	利用可能なインスタンスステータスがない場合に調整ループの停止を回避します #4132
Bug fix	スイッチオーバーの前に、選択されたレプリカがストリーミングモードになるのを待機します #4288
Bug fix	Velero バックアップの使用中にバックアップフックを呼び出すことを許可します
Bug fix	Restic init コンテナが完了するまで待機します
Security	すべての Go の依存関係を更新して、最新のセキュリティ問題を修正しました

1.2.55 EDB Postgres for Kubernetes 1.18.11 release notes

リリース日 2024 年 3 月 22 日

EDB Postgres for Kubernetes バージョン 1.18.11 は、 EDB Postgres for Kubernetes の LTS リリースです。CloudNativePG の対応するアップストリームリリースはありません。

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Enhancement	PostgreSQL #4020 で wal_level GUC のカスタマイズを許可します。
Enhancement	*enabled* #4055 に設定されている場合に WAL アーカイブを無効にするための `cnpg.io/skipWalArchiving`アノテーションを追加しました。
Enhancement	publication`および `subscription`コマンドグループを使用して `kubectl`の `cnpg`プラグインを強化して、PostgreSQL ネイティブ論理レプリケーション `#4052 を強制的にセットアップします。
Enhancement	オペレーター #3686 によって処理される TLS 証明書の自動管理のための`CERTIFICATE_DURATION`および`EXPIRING_CHECK_THRESHOLD`のカスタマイズを許可します。
Enhancement	インプレース更新中に、実行中のオペレーターイメージの対応するカタログから正しいアーキテクチャのバイナリを取得するようになりました。オペレーターは、サポートされているアーキテクチャ #3840 を持つポッドに正しいバイナリを注入できます。
Security	OLM のオペレーター権限に`ClusterRole`の代わりに`Role`を使用するようになりました、名前空間ごとにインストールするときに必要な特権が減少しました #3855 、 #3990 。
Security	PgBouncer プーラー #4080 の SQL クエリに完全修飾オブジェクト名を適用するようになりました。
Bug fix	PVC グループラベルをポッド上のラベルと適切に同期するようになりました。これは、すべてのポッドが削除され、オペレーターが最初にどのポッドを再作成するかを決定する必要がある場合に重要な側面 #3930 。

次のページに続く

表 59 – 前のページからの続き

Type	Description
Bug fix	レプリカのクローンを作成するときに`wal_sender_timeout`を無効にして、低速接続 #4080 によるタイムアウトエラーを防ぎます。
Bug fix	リカバリブートストラップ手順を開始する前にバックアップの準備ができていることを確認し、不完全なバックアップによるリカバリーがエラーループ #3663 に入る可能性のあるエラー状態を防ぎます。
Bug fix	インスタンスポッドは削除されたが、クラスターステータスの休止状態条件が false #3970 に設定されたままである、ハイバーネーションハイバーネーションのコーナーケースを解決します。
Bug fix	Barman Cloud #3931 の Google Cloud 機能を正しく検出します。
Update	デフォルトのオペランドイメージを PostgreSQL 16.2 #3823 に設定します。

1.2.56 EDB Postgres for Kubernetes 1.18.10 release notes

リリース日 2024 年 2 月 2 日

EDB Postgres for Kubernetes バージョン 1.18.10 は、 EDB Postgres for Kubernetes の LTS リリースです。CloudNativePG の対応するアップストリームリリースはありません。

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Enhancement	ephemeralVolumeSource`オプション`#3678 からクレームテンプレートを使用して、Postgres クラスターの一時ボリュームストレージを調整します。
Enhancement	kubectl`の`cnp`プラグインに`pgadmin4`コマンドを導入し、特定のデータベースクラスターへの接続を示し、種類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 - 評価目的のみ`#3701 。
Enhancement	.spec.postgresql.pg_ident`スタンザ、ユーザー名マップのリスト`#3534 を介して PostgreSQL の ident マップファイルのカスタマイズを許可します。
Bug fix	*ALTER SYSTEM* SQL コマンドが無効になっているクラスターで `postgresql.auto.conf`が読み取り専用であるため、 pg_rewind`が失敗するという回復不能な問題が発生します - デフォルト`#3728 。
Bug fix	インポートプロセス #3743 の期間中に PostgreSQL インスタンスの耐久性設定を無効にすることにより、`initdb`ブートストラップ方法のインポート機能を使用する際のディスク領域の不足のリスクを削減します。
Bug fix	誤ったリソース量の比較によるポッドの再起動を回避します。 "1 != 1000m" #3706
Bug fix	pgpass`接続フィールド`#3713 の予約文字を適切にエスケープします。
Bug fix	*.spec.projectedVolumeTemplate.sources* #3647 でゼロと nil の異なる値を考慮することにより、ポッドの体系的なロールアウトを防止します。
Bug fix	*override.conf* #3773 に組み込まれているオプションを`postgresql.auto.conf`からプルーニングすることにより、構成の一貫性を確保します。

1.2.57 EDB Postgres for Kubernetes 1.18.9 release notes

リリース日 2023 年 12 月 22 日

EDB Postgres for Kubernetes バージョン 1.18.9 は、 EDB Postgres for Kubernetes の LTS リリースです。CloudNativePG の対応するアップストリームリリースはありません。

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Subsystem	Description
Security		デフォルトでは、TLSv1.3 はすべての PostgreSQL 12 以降のインストールに適用されます。さらに、ユーザーは `ssl_ciphers`、*ssl_min_protocol_version*、および `ssl_max_protocol_version` GUC #3408 を構成できます。
Security		Dockle を使用した Docker イメージスキャンを統合して、セキュリティ対策を強化します。
Defaults		デフォルトのオペランドイメージは PostgreSQL 16.1 #3270 になりました。
Enhancement		外部クラスター #3533 のリコンシリエーションの向上。
Enhancement		ALTER SYSTEM`コマンド `#3535 を有効/無効にする機能が導入されました。
Enhancement		Cluster`および `Pooler`リソース `#3075 の `.spec.monitoring`スタンザの `podMonitorMetricRelabelings`および `podMonitorRelabelings`オプションを介して Prometheus の動的な再ラベル付けのサポートを追加しました。
Enhancement		PostgreSQL #3522 へのネットワーク接続を確立する際に、`PGPASSFILE`環境変数の使用を削除しました。
Enhancement		クラスターの PVC #3357 を収集することにより、`cnp report`プラグインコマンドを改善しました。
Enhancement	Connection pooler	Pooler`リソースのインスタンスを 0 にスケールダウンすることが可能になりました `#3517 。
Enhancement	Connection pooler	値'pooler' を持つ `k8s.enterprisedb.io/podRole`ラベルをすべてのプーラー展開に追加し、インスタンスポッド #3396 と区別します。
Bug fix		PodDisruptionBudget`リソース `#3312 および #3434 のメタデータ、アノテーション、およびラベルを調整しました。
Bug fix		管理対象資格情報シークレット #3316 のメタデータを調整しました。

次のページに続く

表 61 – 前のページからの続き

Type	Subsystem	Description
Bug fix		pg_basebackup #3586 を介して参加する場合、wal_sender_timeout を無効にしました。
Bug fix		cnpg.io/reload=true`というラベルが付けられ、外部クラスターによって使用されるシークレットが、変更されたときにリロードされるようになりました`#3565 。
Bug fix	Connection pooler	コントローラーが`Pooler`リソース #3428 が所有するすべてのシークレットを監視するようにしました。
Bug fix	Connection pooler	Pooler`リソース`#3391 の`RoleBinding`を調整しました。
Bug fix	Connection pooler	Pooler`リソース`#3389 の`imagePullSecret`を調整しました。
Bug fix	Connection pooler	Pooler`のサービスと必須ラベル`#3349 の追加を調整しました。
Bug fix	Connection pooler	Pooler`ラベルをポッド`#3350 だけでなく、展開にも拡張しました。

1.2.58 EDB Postgres for Kubernetes 1.18.8 release notes

リリース日 2023 年 11 月 8 日

EDB Postgres for Kubernetes バージョン 1.8.8 は、EDB Postgres for Kubernetes の LTS リリースです。CloudNativePG の対応するアップストリームリリースはありません。

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Enhancement	アクティブなストリーミングベースバックアップに関する進行状況情報を使用して、`kubectl`の`cnp`プラグインの`status`コマンドを強化しました。
Enhancement	pgBouncer `Pooler`リソースを使用した`max_prepared_statements`の構成が許可されました。
Technical Enhancement	インスタンスマネージャーで PostgreSQL の拡張クエリプロトコルを使用します。
Bug fix	スイッチオーバー中に WAL アーカイブを一時停止し、完了すると再開します。
Bug fix	PostgreSQL クラスターを管理するときに、インスタンスマネージャーが常に`synchronous_commit = local`を使用するようにしました。
Bug fix	`.spec.certificates.replicationTLSSecret`を介したストリーミングレプリケーションユーザーのカスタム証明書が機能するようになりました。
Bug fix	`k8s.enterprisedb.io/cluster`ラベルを`Pooler`ポッドに設定します。

次のページに続く

表 62 – 前のページからの続き

Type	Description
Change	Postgres レプリケーション設定を制御するために PGDATA 内の `postgresql.auto.conf`ファイルを使用するのを停止し、`override.conf`という名前のファイルに置き換えます。

1.2.59 EDB Postgres for Kubernetes 1.18.7 release notes

リリース日 2023 年 10 月 18 日

EDB Postgres for Kubernetes バージョン 1.8.7 は、EDB Postgres for Kubernetes の LTS リリースです。CloudNativePG の対応するアップストリームリリースはありません。

Kubernetes 1.8.7 の EDB Postgres のハイライト

- `stopDelay` のデフォルト値を 30 秒ではなく 1800 秒に変更しました
- 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ウィンドウを制御するための `smartShutdownTimeout` と呼ばれる新しいパラメーターを導入しました。Postgres を停止するまでの全体的なタイムアウトを計算する一般的な式は `max(stopDelay - smartShutdownTimeout, 30)` です
- `startDelay` のデフォルト値を 30 秒ではなく 3600 に変更しました
- livenessProbe の初期遅延を、より適切な Kubernetes 起動プローブに置き換えて、Postgres サーバーの起動に対応しました
- `switchoverDelay` のデフォルト値を 40000000 秒ではなく 3600 秒に変更しました

さらに、Kubernetes の EDB Postgres のこのリリースには以下が含まれます。

Type	Description
Security fix	デフォルトの`seccompProfile`をオペレーター展開に追加しました。
Enhancement	PostgreSQL のクラッシュが発生した場合に生成されるコアダンプの内容を制御するために `k8s.enterprisedb.io/coredumpFilter`アノテーションを導入しました。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設定されています。
Enhancement	共有メモリおよび一時データのエフェメラルボリュームの一時ストレージ制限の構成を許可します。
Enhancement	Webhook を介したリソース制限と要求の検証。
Enhancement	PostgreSQL の`shared_buffers`がポッドの割り当てられたメモリリソースとコヒーレントであることを確認します。
Enhancement	資格情報シークレットに`uri`および`jdbc-uri`フィールドを追加して、開発者がアプリケーションをデータベースに接続する際に容易にします。
Enhancement	レプリカの準備が整うのを待機するクラスターの状態をより詳細に制御するための新しいフェーズ`Waiting for the instances to become active`を追加しました。

次のページに続く

表 63 – 前のページからの続き

Type	Description
Enhancement	`podSpec`アノテーションを介したポッドのロールアウト条件の検出が向上しました。
Enhancement	プライマリタイムスタンプとアップタイムを kubectl プラグインの `status`コマンドに追加しました。
Technical enhancement	API リファレンスドキュメントを自動的にビルドするように、 `k8s-api-docgen`を `gen-crd-api-reference-docs`に置き換えました。
Bug fix	プライマリロールで準備ができている PVC を優先することにより、プライマリインスタンスが常に最初に再作成されるようにします。
Bug fix	リカバリ中に`k8s.enterprisedb.io/skipEmptyWalArchiveCheck`アノテーションを尊重して、空の WAL アーカイブのチェックをバイパスします。
Bug fix	PostgreSQL サーバーはダウンしているが、プライマリでポッドがアップしているときに、クラスターがスタックするのを防ぎます。
Bug fix	レプリケーションスロットが有効になっている場合、レプリカクラスター内の指定されたプライマリを通常の HA レプリカとして扱わないようにします。
Bug fix	セレクターが変更されるたび、またはラベル/注釈を変更する必要がある場合に、サービスを調整します。
Bug fix	リカバリブートストラップ中に、所有者とデータベースの両方でデフォルトの `app`になります。
Bug fix	キャッシュされたクラスターでの書き込み読み取りの同時実行を回避します。
Bug fix	空の項目を削除し、一意に作成し、生成されたロールの `ResourceName`セクションでソートします。
Bug fix	エラーが発生した場合に、 `ContinuousArchiving`条件が適切に'failed' に設定されていることを確認します。
Bug fix	PodMonitor `labels`および `annotations`を調整します。
Bug fix	`Role`オブジェクトに RBAC `resourceNames`がないことによるバックアップの失敗を修正しました。
Observability	TCP ポートラベルをデフォルトの `pg_stat_replication`メトリックに追加しました。
Observability	Prometheus の `pg_wal_stat`デフォルトメトリックを修正しました。
Observability	Prometheus の `pg_replication`デフォルトメトリックを改善しました
Observability	提供された Grafana ダッシュボードで `alertName`の代わりに `alertInstanceLabelFilter`を使用しました
Observability	メトリック収集で `standard_conforming_strings`を適用します。
Change	デフォルトのオペランドイメージを PostgreSQL 16.0 に設定します。
Change	フェンシングは、スマートシャットダウンの代わりに PostgreSQL の高速シャットダウンを使用してインスタンスを停止するようになりました。
Change	Webhook の名前を kb.io から k8s.enterprisedb.io グループに変更します。
Change	`k8s.enterprisedb.io/instanceRole`ラベルを追加し、既存の `role`ラベルを非推奨にしました。

1.2.60 EDB Postgres for Kubernetes 1.18.6 release notes

リリース日 2023 年 7 月 27 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Enhancement	Kubernetes クラスターの EDB Postgres によるノードの使用状況を監視するために、メトリックとステータスフィールドを追加しました。
Enhancement	huge ページに関連するトラブルシューティングの手順をドキュメントに追加しました。
Enhancement	ドキュメントの FAQ ページを拡張しました。
Enhancement	復元プロセスの開始にチェックを追加して、続行できることを確認します。それができない場合は、改善されたエラー診断を提供します。
Bug fix	リカバリターゲットを設定するロジックが Postgres のロジックと一致することを確認しました。
Bug fix	サービスアカウントの作成中にのみ ownerMetadata を設定することにより、クラスターが所有していないサービスアカウントの引き継ぎを防止しました。
Bug fix	構成のリロード中にインスタンスマネージャーがクラッシュする可能性を防止しました。
Bug fix	失敗したバックアップの後に新しいバックアップが成功した場合に、 LastFailedArchiveTime アラートがトリガーされないようにしました。
Security fix	すべてのプロジェクトの依存関係を最新バージョンに更新しました

1.2.61 EDB Postgres for Kubernetes 1.18.5 release notes

リリース日 2023 年 6 月 13 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.18.5 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.62 EDB Postgres for Kubernetes 1.18.4 release notes

リリース日 2023 年 4 月 27 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.18.4 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.63 EDB Postgres for Kubernetes 1.18.3 release notes

リリース日 2023 年 3 月 20 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.18.3 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.64 EDB Postgres for Kubernetes 1.18.2 release notes

リリース日 2023 年 2 月 14 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.18.2 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。
Feature	EDB Postgtres Advanced Server 15 での *Transparent Data Encryption (TDE)* のサポート TDE は、データベースシステムに保存されているユーザーデータをユーザーに対して透過的に暗号化します。
Feature	新しい外部バックアップアダプタは、バックアップ用のサードパーティツールに Kubernetes の EDB Postgres を統合する汎用方法を提供します。詳細については、 **外部バックアップアダプター** を参照してください。

1.2.65 EDB Postgres for Kubernetes 1.18.1 release notes

リリース日 2022 年 12 月 21 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.18.1 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.66 EDB Postgres for Kubernetes 1.18.0 release notes

リリース日 2022 年 11 月 14 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.18 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.67 EDB Postgres for Kubernetes 1.17.5 release notes

リリース日 2023 年 3 月 20 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.17.5 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.68 EDB Postgres for Kubernetes 1.17.4 release notes

リリース日 2023 年 2 月 14 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.17.4 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.69 EDB Postgres for Kubernetes 1.17.3 release notes

リリース日 2022 年 12 月 21 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.17.2 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.70 EDB Postgres for Kubernetes 1.17.2 release notes

リリース日 2022 年 11 月 14 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.17.2 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.71 EDB Postgres for Kubernetes 1.17.1 release notes

リリース日 2022 年 10 月 07 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.17.1 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。
Enhancement	コントローラマネージャーに `leaseDuration`および `renewDeadline`パラメーターを導入して、オペレーター展開でのリーダー選択の構成を強化します。
Enhancement	初めて WAL ファイルをアーカイブする前に、バックアップオブジェクトストアが空であることを確認するメカニズムを改善します。 `.check-empty-wal-archive`という新しいファイルは、クラスターがブートストラップされた直後に `PGDATA`に配置されます。最初の WAL ファイルが正常にアーカイブされた後、削除されます。
Security	`nonroot:nonroot`ユーザーを使用して、`distroless/static:nonroot`コンテナイメージにコピーされるインスタンスマネージャーバイナリの権限を明示的に設定します。
Bug fix	プライマリに昇格した後、スタンバイでアクティブな接続をドロップします。
Bug fix	ネイティブ Prometheus エクスポータでの `MAPPEDMETRIC`および `DURATION`メトリックタイプ変換を優先します。
Bug fix	PostgreSQL 形式を使用してマイクロ秒の精度で指定されたタイムスタンプが正しく解析されることを保証します。

1.2.72 EDB Postgres for Kubernetes 1.17.0 release notes

リリース日:2022 年 9 月 06 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.17 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.73 EDB Postgres for Kubernetes 1.16.5 release notes

リリース日 2021 年 12 月 21 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.16.5 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.74 EDB Postgres for Kubernetes 1.16.4 release notes

リリース日 2022 年 11 月 14 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.16.4 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.75 EDB Postgres for Kubernetes 1.16.3 release notes

リリース日 2022 年 10 月 07 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.16.3 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。
Enhancement	コントローラマネージャーに`leaseDuration`および`renewDeadline`パラメーターを導入して、オペレーター展開でのリーダー選択の構成を強化します。
Enhancement	初めて WAL ファイルをアーカイブする前に、バックアップオブジェクトストアが空であることを確認するメカニズムを改善します。`.check-empty-wal-archive`という新しいファイルは、クラスターがブートストラップされた直後に`PGDATA`に配置されます。最初の WAL ファイルが正常にアーカイブされた後、削除されます。
Security	`nonroot:nonroot`ユーザーを使用して、`distroless/static:nonroot`コンテナイメージにコピーされるインスタンスマネージャーバイナリの権限を明示的に設定します。
Bug fix	プライマリに昇格した後、スタンバイでアクティブな接続をドロップします。

次のページに続く

表 79 – 前のページからの続き

Type	Description
Bug fix	ネイティブ Prometheus エクスポータでの `MAPPEDMETRIC`および `DURATION`メトリックタイプ変換を優先します。

1.2.76 EDB Postgres for Kubernetes 1.16.2 release notes

リリース日:2022 年 9 月 06 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.16.2 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.77 EDB Postgres for Kubernetes 1.16.1 release notes

リリース日 2022 年 8 月 12 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.16 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

アップグレードポリシーの重要な変更

CloudNativePG コミュニティに合わせて、製品の最後の 2 つのマイナーバージョンをサポートする新しいポリシーを採用します。

その結果、OpenShift の新しい OLM チャネルに次のヘッドバージョンを導入して、Kubernetes の EDB Postgres の更新ストリームを管理します。

- `fast` 最新の利用可能なマイナーリリースの最新の利用可能なパッチリリース
- `stable-v1.16` 1.16 マイナーリリースの最新の利用可能なパッチリリース
- `stable-v1.15` 1.15 マイナーリリースの最新の利用可能なパッチリリース

このリリースより前は、サポートしていた唯一のチャネルは `stable` チャネルでした。このチャンネルは現在廃止されています。ただし、下位互換性するために、現在は `stable-v1.15` チャネルのエイリアスとして設定されています。バージョン 1.15 がサポート終了に達すると、削除されます。

Openshift での 1.16.x オペレーターバージョンへのアップグレードに関する重要な情報

バージョン 1.16.0 以降で、オペレーターのステータスで条件が表される方法を変更しました。この変更により、Operator Lifecycle Manager が既存の CR に対して新しい CRD をチェックする方法が原因で、アップグレードプロセス中に古い条件のいずれかが設定されている場合、Openshift でオペレーターのアップグレードがハングアップする可能性があります。

Openshift に 1.16.x をインストールする前に、オペレーターの 1.15.x またはそれ以前のバージョンからアップグレードする場合、既存のバージョンのオペレーターをアンインストールし、すべてのステータスから古い条件をすべて削除することをお勧めします。Kubernetes クラスターの既存の EDB Postgres 。これは、Kubernetes クラスターの既存の EDB Postgres の操作性には影響しません。

既存の条件を削除するには、次を実行します。

```
while IFS=  read  NS CLUSTER; do
  kubectl -n ${NS} patch --type=json ${CLUSTER} --subresource=status -p=[{"op":
↪"remove", "path": "/status/conditions"}];
done < <(kubectl get cluster -A --no-headers=true -o jsonpath={range .items[*]}{.
↪metadata.namespace}{" cluster/"}{.metadata.name}{"\n"}{end})
```

重要

kubectl コマンドはバージョン 1.24 以降である必要があります。出力 `The request is invalid` を取得した場合、それはターゲットクラスターに条件がなかったことを意味します。

このコマンドは、Openshift クラスター内の Kubernetes クラスターのすべての EDB Postgres からすべての条件を削除します。コマンドが完了すると、バージョン 1.16.x を安全にインストールできます。

すでに 1.15.x またはそれ以前から 1.16.x にアップグレードしようとしており、1.16.x のインストールは「保留中」と表示され、以前のバージョンは「更新できません」と表示されている場合は、オペレーターの両方のバージョンをアンインストールして実行しますステータスを削除するコマンド。

1.2.78 EDB Postgres for Kubernetes 1.16.0 release notes

リリース日 2022 年 7 月 7 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.16 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。
Enhancement	OpenShift で利用可能な新しい「fast」チャネル、Kubernetes 用 EDB Postgres の最新の利用可能なマイナーリリースでの最新の利用可能なパッチリリースへの更新パスを提供します。詳細については、次のセクションを参照してください。

次のページに続く

表 82 – 前のページからの続き

Type	Description
Enhancement	OpenShift で利用可能な新しい「stable-v1.16」チャネル、Kubernetes 用 EDB Postgres の 1.16 リリースブランチで利用可能な最新のパッチリリースへの更新パスを提供します。詳細については、次のセクションを参照してください。

アップグレードポリシーの重要な変更

CloudNativePG コミュニティに合わせて、製品の最後の 2 つのマイナーバージョンをサポートする新しいポリシーを採用します。

その結果、OpenShift の新しい OLM チャネルに次のヘッドバージョンを導入して、Kubernetes の EDB Postgres の更新ストリームを管理します。

- `fast` 最新の利用可能なマイナーリリースの最新の利用可能なパッチリリース
- `stable-v1.16` 1.16 マイナーリリースの最新の利用可能なパッチリリース
- `stable-v1.15` 1.15 マイナーリリースの最新の利用可能なパッチリリース

このリリースより前は、サポートしていた唯一のチャネルは `stable` チャネルでした。このチャンネルは現在廃止されています。ただし、下位互換性するために、現在は `stable-v1.15` チャネルのエイリアスとして設定されています。バージョン 1.15 がサポート終了に達すると、削除されます。

Openshift での 1.16.x オペレーターバージョンへのアップグレードに関する重要な情報

バージョン 1.16.0 以降で、オペレーターのステータスで条件が表される方法を変更しました。この変更により、Operator Lifecycle Manager が既存の CR に対して新しい CRD をチェックする方法が原因で、アップグレードプロセス中に古い条件のいずれかが設定されている場合、Openshift でオペレーターのアップグレードがハングアップする可能性があります。

Openshift に 1.16.x をインストールする前に、オペレーターの 1.15.x またはそれ以前のバージョンからアップグレードする場合、既存のバージョンのオペレーターをアンインストールし、すべてのステータスから古い条件をすべて削除することをお勧めします。Kubernetes クラスターの既存の EDB Postgres 。これは、Kubernetes クラスターの既存の EDB Postgres の操作性には影響しません。

既存の条件を削除するには、次を実行します。

```
while IFS=  read  NS CLUSTER; do
  kubectl -n ${NS} patch --type=json ${CLUSTER} --subresource=status -p=[{"op":
↪"remove", "path": "/status/conditions"}];
done < <(kubectl get cluster -A --no-headers=true -o jsonpath={range .items[*]}{.
↪metadata.namespace}{" cluster/"}{.metadata.name}{"\n"}{end})
```

重要

kubectl コマンドはバージョン 1.24 以降である必要があります。出力 `The request is invalid` を取得した場合、それはターゲットクラスターに条件がなかったことを意味します。

このコマンドは、Openshift クラスター内の Kubernetes クラスターのすべての EDB Postgres からすべての条件を削除します。コマンドが完了すると、バージョン 1.16.x を安全にインストールできます。

すでに 1.15.x またはそれ以前から 1.16.x にアップグレードしようとしており、1.16.x のインストールは「保留中」と表示され、以前のバージョンは「更新できません」と表示されている場合は、オペレーターの両方のバージョンをアンインストールして実行しますステータスを削除するコマンド。

1.2.79 EDB Postgres for Kubernetes 1.15.5 release notes

リリース日 2022 年 10 月 07 日

!!!警告バージョン 1.15 はサポート終了 EOL に達しました。バージョン 1.15.5 は、1.15 マイナーバージョンの最後のリリースです。Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.15.5 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。
Enhancement	コントローラマネージャーに `leaseDuration`および `renewDeadline`パラメーターを導入して、オペレーター展開でのリーダー選択の構成を強化します。
Enhancement	初めて WAL ファイルをアーカイブする前に、バックアップオブジェクトストアが空であることを確認するメカニズムを改善します。 `.check-empty-wal-archive`という新しいファイルは、クラスターがブートストラップされた直後に `PGDATA`に配置されます。最初の WAL ファイルが正常にアーカイブされた後、削除されます。
Security	`nonroot:nonroot`ユーザーを使用して、`distroless/static:nonroot`コンテナイメージにコピーされるインスタンスマネージャーバイナリの権限を明示的に設定します。
Bug fix	クラスターの条件を `metav1.Conditions`構造体と互換性のあるものにします。
Bug fix	プライマリに昇格した後、スタンバイでアクティブな接続をドロップします。
Bug fix	ネイティブ Prometheus エクスポータでの `MAPPEDMETRIC`および `DURATION`メトリックタイプ変換を優先します。

1.2.80 EDB Postgres for Kubernetes 1.15.4 release notes

リリース日:2022 年 9 月 06 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.15.4 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.81 EDB Postgres for Kubernetes 1.15.3 release notes

リリース日 2022 年 8 月 12 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.15.3 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

アップグレードポリシーの重要な変更

CloudNativePG コミュニティと合わせて、1.16 で始まる製品の最後の 2 つのマイナーバージョンをサポートする新しいポリシーを採用します。

その結果、OpenShift の新しい OLM チャネルに次のヘッドバージョンを導入して、Kubernetes の EDB Postgres の更新ストリームを管理します。

- `fast` 最新の利用可能なマイナーリリースの最新の利用可能なパッチリリース
- `stable-v1.16` 1.16 マイナーリリースの最新の利用可能なパッチリリース
- `stable-v1.15` 1.15 マイナーリリースの最新の利用可能なパッチリリース

1. 16 のリリース前は、サポートしていた唯一のチャンネルは `stable` チャンネルでした。このチャンネルは現在廃止されています。ただし、下位互換性を保つために、現在は `stable-v1.15` チャネルのエイリアスとして設定されており、バージョン 1.15 がサポート終了になると削除されます。

1.2.82 EDB Postgres for Kubernetes 1.15.2 release notes

リリース日 2022 年 7 月 7 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.15.2 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。
Enhancement	OpenShift で利用可能な新しい「stable-v1.15」チャネル、Kubernetes 用 EDB Postgres の 1.15 リリースブランチで利用可能な最新のパッチリリースへの更新パスを提供します。「stable」チャネルは「stable-v1.15」の同義語です。詳細については、次のセクションを参照してください。

アップグレードポリシーの重要な変更

CloudNativePG コミュニティと合わせて、1.16 で始まる製品の最後の 2 つのマイナーバージョンをサポートする新しいポリシーを採用します。

その結果、OpenShift の新しい OLM チャネルに次のヘッドバージョンを導入して、Kubernetes の EDB Postgres の更新ストリームを管理します。

- `fast` 最新の利用可能なマイナーリリースの最新の利用可能なパッチリリース
- `stable-v1.16` 1.16 マイナーリリースの最新の利用可能なパッチリリース
- `stable-v1.15` 1.15 マイナーリリースの最新の利用可能なパッチリリース

1. 16 のリリース前は、サポートしていた唯一のチャンネルは `stable` チャンネルでした。このチャンネルは現在廃止されています。ただし、下位互換性を保つために、現在は `stable-v1.15` チャネルのエイリアスとして設定されており、バージョン 1.15 がサポート終了になると削除されます。

1.2.83 EDB Postgres for Kubernetes 1.15.1 release notes

リリース日 2022 年 5 月 27 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.15.1 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.84 EDB Postgres for Kubernetes 1.15.0 release notes

リリース日 2022 年 4 月 21 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Upstream merge	コミュニティ CloudNativePG 1.15.0 とマージされました。コミュニティ *Introduction and Release Notes* を参照してください。

1.2.85 EDB Postgres for Kubernetes 1.14.0 release notes

リリース日 2022 年 3 月 25 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Feature	Barman Cloud 2.19 で導入された機能を利用して、バックアップとリカバリーのための Google Cloud Storage をネイティブにサポートします。
Feature	Cluster オブジェクトの LastBackupSucceeded 条件の導入により、バックアップの可観測性が向上しました。
Feature	ホットスタンバイ依存のパラメーター max_connections、max_prepared_transactions、max_locks_per_transaction、max_wal_senders、max_worker_processes の更新をサポートします。
Feature	クラスターオブジェクトに Online upgrade in progress フェーズを追加して、オペレーターのオンラインアップグレードが進行中の場合を表示します。
Feature	S3 オブジェクトストレージの資格情報を提供する別の方法として、AWS IAM ロールを継承する機能。
Feature	プーラーの authQuerySecret と証明書の不透明なシークレットのサポート。
Feature	デフォルトの PostgreSQL バージョンを 14.2 に更新しました。
Feature	repair という名前の kubectl cnp プラグインに新しいコマンドを追加して、Kubernetes クラスター全体の 1 つまたはすべての名前空間のクラスターにメンテナンスウィンドウを設定します。
Container images	最新の PostgreSQL および EDB Postgres Advanced Server コンテナには、 Barman Cloud 2.19 が含まれます。
Security fix	OpenShift を含む Operator Lifecycle Manager を使用した名前空間オペレーターインストールの RBAC 適用の強化。 OpenShift ユーザーはこのバージョンに更新することをお勧めします。
Bug fix	インスタンスマネージャーが元の pg_control ファイルのコピーを保存することにより、中断された pg_rewind を再試行できます。
Bug fix	pg_rewind を実行する前に、古い PID ファイルをクリーンアップします。
Bug fix	primary_conninfo のキーによるソートを強制して、13 より前のバージョンの PostgreSQL でのランダムな再起動を回避します。
Bug fix	調整時にサービスアカウントの変更ラベル、注釈などを保存します。
Bug fix	imagePullPolicy のデフォルト値の適用を無効にします。
Bug fix	WAL セグメントサイズの initDB 検証を改善します。
Bug fix	指定された LSN を使用してクラスターをリカバリーするときに、targetLSN オプションを適切に処理します。
Bug fix	サーバー証明書と CA 証明書の両方で証明書チェーンを許可することにより、カスタム TLS 証明書の検証を修正します。

1.2.86 EDB Postgres for Kubernetes 1.13.0 release notes

リリース日 2022 年 2 月 17 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Feature	Snappy 圧縮のサポート。 Snappy は、バックアップの高速圧縮オプションであり、より低い圧縮率を使用してオブジェクトストアへのアップロード速度を向上させます。
Feature	Barman オブジェクトストアにアップロードされたファイルのタグ付けのサポート。この機能には、オペランドイメージに Barman 2.18 が必要です。クラスター削除後のバックアップの数。
Feature	status.conditions を使用したクラスターのステータスの拡張。 ContinuousArchiving 条件は、クラスターが WAL ファイルのアーカイブを開始したことを示します。
Feature	追加情報を使用して kubectl の cnp プラグインの status コマンドを改善します。クラスターのステータスを表示するクラスターの概要セクションと、クラスターで使用される証明書のステータスと有効期限までの残り時間を含む証明書ステータスセクションを追加します。
Feature	新しい barman-cloud-check-wal-archive コマンドをサポートして、新しいクラスターの作成時に空でないバックアップ先を検出します。
Feature	シークレットを使用して、MONITORING_QUERIES_SECRET 構成変数を介してデフォルトのモニタリングクエリを追加するサポートを追加します。
Feature	ユーザーが AppArmor を使用してコンテナの権限を制限できるようにします AppArmor サポートで展開された Kubernetes クラスターで
Feature	kubectl の cnp プラグインに Windows プラットフォームのサポートを追加し、プラグインは Windows x86 および ARM で利用できるようになりました。
Feature	Kubernetes 1.18 および非推奨の API バージョンのサポートを終了
Container images	PostgreSQL コンテナには、 Barman 2.18 が含まれます。
Security fix	所有者およびスーパーユーザーのシークレット内のユーザー名フィールドの一貫性チェックを追加します。以前は、悪意のあるユーザーがシークレットを使用して、PostgreSQL ユーザーのパスワードを変更することができました。
Bug fix	Postgres ポッドからステータスを取得するコードのメモリリークを修正します。
Bug fix	クラッシュ後の PostgreSQL の自己再起動を無効にします。インスタンスコントローラは、PostgreSQL インスタンスのライフサイクルを処理します。
Bug fix	検証 Webhook の spec.postgresUID および spec.postgresGID フィールドの変更を防止します。クラスターの作成後にこれらのフィールドを変更すると、PostgreSQL は起動できなくなります。
Bug fix	バックアップおよび WAL アーカイブ処理コードからログの詳細度を減らします。
Bug fix	まだ初期化されていないときにクラスターが正常としてマークされるバグを修正します。
Bug fix	非常に高い WAL 生産レートのクラスター内のスタンバイサーバーは、調整されるとストリーミングに切り替えることができます。
Bug fix	まれにクラッシュにつながる可能性がある PostgreSQL ポッドの起動中に競合状態を修正します。
Bug fix	クラスター内の最初の PVC の初期化障害を引き起こす可能性のある競合状態を修正します。

次のページに続く

表 90 – 前のページからの続き

Type	Description
Bug fix	レプリカとしてクラスターに参加する前に、降格したばかりのプライマリポッドの余分な再起動を削除します。
Bug fix	バックアップから復旧するときに、レプリケーション依存の PostgreSQL 構成パラメーターを正しく処理します。
Bug fix	クラスターの作成中に PostgreSQL 構成の検証の欠落を修正します。

1.2.87 EDB Postgres for Kubernetes 1.12.0 release notes

リリース日 2022 年 1 月 11 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Feature	サポートされている Kubernetes ディストリビューションのリストに Kubernetes 1.23 を追加し、2020 年 12 月に Kubernetes プロジェクトによるサポートを終了した 1.17 のエンドツーエンドテストを削除します。
Feature	2 秒の接続タイムアウトと 30 秒の通信タイムアウトを追加することにより、ネットワークに問題が発生した場合のポッドステータスチェックの応答性を向上させます。この変更では、ポッドが障害が発生したと宣言する前にそのステータスを報告するのをオペレーターが待機する時間に制限を設定し、フェールオーバー操作の堅牢性と予測可能性が向上します。
Feature	.spec.inheritedMetadata フィールドをクラスターに導入し、ユーザーがクラスターによって生成されたすべてのオブジェクトに適用されるラベルとアノテーションを指定できるようにします。
Feature	インスタンスのステータスを計算するときに実行されるクエリの数を削減します。
Feature	PgBouncer の準備状況プローブを追加します。
Feature	Azure プロトコルを使用する場合、Barman のバックアップオブジェクトストアのエンドポイントのカスタム認証局のサポートを追加します。
Bug fix	フェイルオーバー中に、すべての WAL ストリーミング接続が閉じるまで、新しいプライマリの選択を待ちます。オペレーターは、デフォルトで wal_sender_timeout および wal_receiver_timeout を 5 秒に設定して、プライマリにネットワークの問題があるかどうかをスタンバイノードがすぐに認識できるようにします。
Bug fix	レプリカクラスター内の PostgreSQL インスタンスの「archive_mode」を「always」に設定することにより、レプリカクラスターの WAL アーカイブ戦略を変更して、ローリング更新を修正します。次に、現在およびターゲット指定されたプライマリからのみ WAL のアップロードを制限します。 WAL は、スイッチオーバー中に 2 回アップロードされる場合がありますが、これは問題ではありません。
Bug fix	レプリカクラスターソースにある Barman のバックアップオブジェクトストアのエンドポイントのカスタム認証局のサポートを修正しました。

次のページに続く

表 91 – 前のページからの続き

Type	Description
Bug fix	クラスター名前空間のデフォルトの監視構成マップの固定名を使用します。
Bug fix	デフォルトの Webhook が何らかの理由で機能していない場合、オペレーターは調整サイクル中にデフォルトでクラスターを更新するようになりました。
Bug fix	リソース要求と制限の比較を修正して、調整サイクルごとにすべてのポッドの更新につながるまれな問題を修正します。
Bug fix	Webhook からのログメッセージを改善して、オブジェクト名前空間も含めます。
Bug fix	すべての調整ループの開始時に「デフォルト」メッセージのロギングを停止します。
Bug fix	EnablePodMonitor が false の場合、調整サイクルごとに PodMonitor 削除のログを停止します。
Bug fix	ポッドに到達できない場合、アーキテクチャの不一致の可能性について文句を言わないでください。

1.2.88 EDB Postgres for Kubernetes 1.11.0 release notes

リリース日 2021 年 12 月 15 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Feature	パラレル WAL のアーカイブと復元 WAL アーカイブ PostgreSQL の archive_command と WAL リストアの両方で実行するパラレルジョブの最大数を設定する backupObjectStore.maxParallel オプションを導入することにより、データベースが大量の書き込みシステムで WAL 生成に追いつくことができます。 repair_command による)。並列復元オプションを使用すると、必要な WAL ファイルを取得して、シークエンシャルではなくパラレルで再生することにより、新しく昇格したスタンバイをより早く準備ができた状態にできます。
Feature	モニタリングのデフォルトのメトリックセット default-monitoring と呼ばれる新しい ConfigMap は、オペレーターの同じ名前空間に自動的に展開され、デフォルトで、既存の Postgres クラスターに追加されます。このような動作は、オペレーターの構成で MONITORING_QUERIES_CONFIGMAP パラメーターを設定することにより、グローバルに変更できます。または.spec. モニタリングを介してクラスターレベルで。
Feature	disableDefaultQueries オプションデフォルトで false に設定されます。
Feature	クラスターのモニタリングセクションに enablePodMonitor オプションを導入して、PodMonitor リソースを自動的に管理し、Prometheus とシームレスに統合します。
Feature	ポッドが Kubernetes によって強制終了される前に、目的の stopDelay 時間の前半でスマートシャットダウンを実行し、残りの半分で高速シャットダウンを実行して、PostgreSQL のシャットダウンプロシージャを改善します。

次のページに続く

表 92 – 前のページからの続き

Type	Description
Feature	switchoverDelay オプションを追加して、以前のプライマリが正常にシャットダウンし、新しいプライマリを昇格する前にすべての WAL ファイルをアーカイブするために与えられた時間を制御しますデフォルトでは、クラウドネイティブ PostgreSQL はデータの耐久性を特権にするために無期限に待機します。
Feature	ローリング更新を発行することにより、PostgreSQL クラスターのリソース要求と制限の変更を処理します。
Feature	ストリーミングレプリケーションステータス、データベースの合計サイズ、クラスター内のインスタンスのロールなどの追加情報を使用して、kubectl の cnp プラグインの status コマンドを改善します。
Feature	PostgreSQL の共有メモリの管理の dynamic_shared_memory_type および shared_memory_type パラメーターの構成を有効にすることにより、多くのパラレルワーカーを使用したワークロードのサポートを強化します。
Feature	クラスターレベルで定義されたラベルとアノテーションを、ポッドを含む関連リソースに伝播します削除はサポートされていません。
Feature	Kubelet によって追い出されたポッドを自動的に削除します。
Feature	ENABLE_AZURE_PVC_UPDATES オペレーター構成オプションを介して、必要に応じてクラスターのローリング更新を発行することにより、Azure の永続ボリュームの自動サイズ変更を管理しますデフォルトでは無効。
Feature	k8s.enterprisedb.io/reconciliationLoop アノテーションを導入します。これは、特定の Postgres クラスターで無効に設定されている場合、調整ループの実行を防ぎます。
Feature	initdb ブートストラップ方法の一部として postInitApplicationSQL オプションを導入して、クラスターが作成された直後にスーパーユーザーとしてメインアプリケーションデータベースで実行する SQL クエリーのリストを指定します。
Feature	EDB Postgres Advanced 14.2 のサポート。
Bug fix	Liveness プローブは、PostgreSQL サーバーの起動プロセスを正しく処理するようになりました。これは、少数の顧客から報告された問題を修正し、一貫した状態に到達するために WAL ファイルを復旧する必要がある再起動したスタンバイサーバーに影響を与えますが、liveness プローブのタイムアウトがキックインする前にそれを行うことができず、ポッドを CrashLoopBackOff にしたままですステータス。
Bug fix	Liveness プローブは、タイムラインの流用後に pg_rewind を使用して現在のプライマリと再調整する必要がある元のプライマリの場合を正しく処理するようになりました。これにより、新しいスタンバイのポッドが Kubernetes によって繰り返し強制終了されないように修正されます。
Bug fix	Postgres ポッドからクライアントサイドの調整を減らしますたとえば、優先順位と公平性ではなく、クライアントサイドの調整により 1.182388649s 待機しました、要求 GET。
Bug fix	提供されたものを使用して、OpenShift および OLM インストールで公開キーインフラストラクチャ PKI の初期化を無効にします。

次のページに続く

表 92 – 前のページからの続き

Type	Description
Bug fix	再起動が必要な構成パラメーターを変更する場合、常にプライマリを最後のままにしてください。
Bug fix	ジョブが正常に完了した後にのみ PVC を準備ができているようにマークし、PVC の初期化の競合状態を防ぎます。
Bug fix	有効期限が切れた Webhook TLS シークレットを更新するときは、正しい公開キーを使用します。
Bug fix	LSN を解析する際のオーバーフローを修正しました。
Bug fix	起動時に古い PID ファイルを削除します。
Bug fix	Pooler リソースに、オペレーターで定義された imagePullSecret が存在する場合、それを継承させます。

1.2.89 EDB Postgres for Kubernetes 1.10.0 release notes

リリース日 2021 年 11 月 11 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Feature	PgBouncer を使用した接続プーラー ローカルの PostgreSQL クラスターの接続プーラーとして使用される PgBouncer 展開を自動的に管理するためのプーラーリソースとコントローラを紹介します。この機能には、TLS クライアント/サーバー接続、パスワード認証、高可用性、ポッドテンプレートのサポート、主要な PgBouncer パラメーターの構成、PAUSE/RESUME、JSON 形式のロギング、統計、プール、リストの Prometheus エクスポーターが含まれます。
Feature	バックアップリテンションポリシー バックアップのリカバリウィンドウリテンションポリシーの定義をサポートしますたとえば、30 日のリカバリウィンドウを保証するための `30d`。
Feature	オペレーターのインプレース更新 インスタンスマネージャーのインプレースオンライン更新を導入します。これにより、オペレーターの更新に続いてクラスター全体のローリング更新を実行する必要がなくなります。デフォルトでは、このオプションは無効になっています。詳細については、ドキュメントを参照してください。
Feature	initdb ブートストラップ方法でカスタマイズできるオプションのリストを dataChecksums、エンコーディング、localeCollate、localeCType、walSegmentSize に制限します。これにより、オプション配列は時代遅れになり、v2 API で削除される予定です。
Feature	initdb ブートストラップ方法の一部として postInitTemplateSQL オプションを導入して、クラスターが作成された直後にスーパーユーザーとして template1 データベースで実行する SQL クエリのリストを指定します。この機能を使用すると、クラスター内に作成されるすべてのアプリケーションデータベースにデフォルトオブジェクトを含めることができます。

次のページに続く

表 93 – 前のページからの続き

Type	Description
Feature	インスタンス Prometheus エクスポーターに追加された新しいデフォルトメトリック Postgres バージョン、クラスター名、およびバックアップカタログによると、リカバリー可能の最初のポイント。
Feature	障害が発生した後、バックアップの取得を再試行します。
Feature	Barman Cloud 機能に関する認識を高めて、頻繁に更新されないオペランドイメージに存在しない最近導入された機能リテンションポリシーまたは Azure Blob Container ストレージなどをオペレーターが呼び出さないようにします。
Feature	cnp プラグインの status コマンドの出力をバックアップに関する情報と統合します。
Feature	PVC のステータス初期化または準備ができていることを報告する新しいアノテーションを導入します。
Feature	バックアップオブジェクトを含む、クラスターで生成されたすべてのオブジェクトの k8s.enterprisedb.io/clusterlabel にクラスター名を設定します。
Feature	Cluster、Backup、および ScheduledBackup の種類での非推奨の API バージョン postgresql.k8s.enterprisedb.io/v1alpha1 のサポートを削除します。
Feature	デフォルトのオペランドイメージを PostgreSQL 14.2 に設定します。
Security fix	オペレーターコンテナ securityContext の allowPrivilegeEscalation を false に設定します。
Bug fix	シングルインスタンスクラスターでのメンテナンス中に、プライマリ PodDisruptionBudget を無効にします。
Bug fix	復旧操作中に正しい証明書認証局 CA を使用します。
Bug fix	バックアップする前に WAL アーカイブステータスを確認する際の Postgres 接続リークを防止します。
Bug fix	一時的なエラーの後、WAL アーカイブ/復元を 100 ミリ秒スリープさせます。他の場合はログがフラッディングします。

1.2.90 EDB Postgres for Kubernetes 1.9.2 release notes

リリース日 2021 年 10 月 15 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Feature	2 つの新しいロガーで JSON ログを強化します。PostgreSQL の archive_command の wal-archive と、スタンバイの restore_command の wal-restore です。
Bug fix	スタンバイプロモーション中に WAL アーカイブを有効にします .history ファイルがアーカイブされないようにします。
Bug fix	Barman 2.13 以降を使用する場合にのみ、--cloud-provider オプションを Barman Cloud ツールに渡して、古いオペランドでのエラーを回避します。
Bug fix	バックアップをトリガーする前に、プライマリのポッドの準備が整うのを待ちます。

1.2.91 EDB Postgres for Kubernetes 1.9.1 release notes

リリース日 2021 年 9 月 30 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Feature	このリリースは、特定のイメージ名を定義せずに新しいクラスターが作成されるときに、それをデフォルトのメジャーバージョンにすることにより、PostgreSQL 14 のローンチを祝うものです。
Bug fix	プライマリポッドのログに表示されるバーマンエンドポイント CA シークレットメッセージを取得中にエラーが発生し、バックアップが正しく機能しない問題を修正します。
Bug fix	インスタンスマネージャーとの一時的な通信の問題が発生した場合、新しいバックアップの要求を適切に再試行します。

1.2.92 EDB Postgres for Kubernetes 1.9.0 release notes

リリース日 2021 年 9 月 28 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Feature	サポートされている Kubernetes ディストリビューションのリストに Kubernetes 1.22 を追加し、1.16 を削除します。
Feature	必要に応じてバックアップアーカイブから WAL ファイルを取得するために、pg_rewind の--restore-target-wal オプションのサポートを導入します PostgreSQL / EPAS 13 以降でのみ使用可能。
Feature	バージョン 1.9.0 は、バージョン 1.9.1 のリリースプロセスとその後のリリースの遅延により、OpenShift では利用できません。
Feature	各データベースの pg_catalog.pg_largeobject テーブルのページ数を推定する Prometheus エクスポーターのデフォルトメトリックを公開します。
Feature	ローカルのインメモリキャッシュを介して、WAL のアーカイブと取得のパフォーマンスを向上させます。
Bug fix	pg_isready を呼び出すときに postgres ユーザーを明示的に設定します- OpenShift の制限付き SCC で必要です。
Bug fix	ステータスの FirstRecoverabilityPoint を適切に更新します。
Bug fix	バックアップが要求された場合、指定されたプライマリで archive_mode = Always に設定します。

1.2.93 EDB Postgres for Kubernetes 1.8.0 release notes

リリース日 2023 年 9 月 13 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Feature	外部クラスターセクションで定義されたオブジェクトストアから直接、フルまたはポイントインタイムリカバリーを介して新しいクラスターをブートストラップし、バックアップ CR を定義するための以前の要件を排除します。
Feature	スケジュールされたバックアップに即時オプションを導入して、最初の Postgres インスタンスが実行された直後にバックアップを要求し、ポイントインタイムリカバリが構成されているときにクラスターの先頭に巻き戻す機能を追加します。
Feature	クラスターステータスに firstRecoverabilityPoint を追加して、バックアップオブジェクトストアのコンテンツに基づいてリカバリを要求する最も古い一貫したポイントを報告します。
Feature	次の新しいメトリックを公開することにより、PostgreSQL インスタンスのデフォルトの Prometheus エクスポータを強化します。ディスク上の WAL ファイルの数と計算された合計サイズ、ディスク上の WAL ファイルの数と計算された合計サイズ、アーカイブステータスの.ready および.done ファイルの数フォルダー、レプリカモードのフラグ、要求された最小/最大同期レプリカの数、および予想されるおよび実際に観察されたレプリカ。
Feature	Prometheus エクスポーターでカスタムメトリックを定義するときに runonserver オプションのサポートを追加して、メトリックの収集を PostgreSQL バージョンの範囲に制限します。
Feature	Barman Cloud の Barman 2.13 で導入された機能を利用して、バックアップとリカバリーのための Azure Blob Storage をネイティブにサポートします。
Feature	liveness プローブは pg_isready に依存します。
Feature	リカバリターゲット時間のタイムスタンプ仕様の RFC3339 形式をサポートします。
Feature	.spec.imagePullPolicy を導入して、クラスター用に作成されたすべてのポッドとジョブのイメージコンテナのプルポリシーを制御します。
Feature	OpenShift 4.5 を置き換える OpenShift 4.8 のサポートを追加します。
Feature	PostgreSQL 14 ベータ版をサポートします。
Feature	ストリーミング接続を必要とせずに、外部クラスターセクションで定義されたオブジェクトストアからのクラスター間レプリケーションにより、レプリカクラスター機能を強化します。実験。
Feature	クラスターの仕様に logLevel オプションを導入して、error、info、debug、または trace のいずれかのレベルを指定します。
Security fix	.spec.enableSuperuserAccess を導入して、パスワード認証を介して postgres ユーザーとのネットワークアクセスを有効/無効にします。
Security fix	spec.licenseKeySecret を使用して、シークレットでのライセンスキーの指定を有効にします。
Bug fix	クラスターが回復不能な状態になり、人間の介入が必要になると、ユーザーに適切に通知します。

1.2.94 EDB Postgres for Kubernetes 0.8 release notes

リリース日 2021 年 1 月 29 日

Type	Description
Feature	API バージョンを v1 にアップグレードしました。
Feature	`nodeSelector`を介してノードアフィニティを実装します。
Feature	デフォルトで`AntiAffinity`をアクティブにします。
Feature	完了した`Jobs`をクラスターから削除します。
Feature	コントローラランタイムを v0.8.1 にアップグレードします。
Change	スキーマバージョン 2 を使用してコンテナイメージをビルドします。
Change	より多くのケース、堅牢性、信頼性の向上をカバーすることにより、E2E テストを強化します。
Fix	バグ修正とコードの改善。

1.2.95 EDB Postgres for Kubernetes 1.7.1 release notes

リリース日 2021 年 8 月 11 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Feature	同期レプリケーションに関して構成より自己修復を優先し、クラスターが自己修復操作中に要件を満たせない場合に、オペレーターが minSyncReplicas および maxSyncReplicas 設定を一時的にオーバーライドできるようにします。
Feature	initdb ブートストラップ方法の一部として postInitSQL オプションを導入して、クラスターが作成された直後にスーパーユーザーとして実行する SQL クエリのリストを指定します。
Bug fix	プライマリの準備ができていないときにオペレーターがフェイルオーバーできるようにします 1.7.0 で導入されたバグ。
Bug fix	LOCAL 同期コミットレベルを使用して管理クエリーを実行します。
Bug fix	PGAudit の複数行のログエントリを正しく解析します。

1.2.96 EDB Postgres for Kubernetes 1.7.0 release notes

リリース日 2021 年 7 月 28 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Feature	JSON 出力で直接使用可能な pgaudit と呼ばれる新しいタイプのロガーを使用して、PGAudit にネイティブサポートを追加します。
Feature	pg_stat_statements および auto_explain 拡張機能のネイティブサポート。

次のページに続く

表 100 – 前のページからの続き

Type	Description
Feature	1 つ以上のデータベースでユーザー定義のメトリッククエリーを実行するための Prometheus エクスポーターの target_databases オプションシェルのようなパターンマッチングを介したデータベースの自動検出を含む。
Feature	manual_switchover_required メトリックを公開して、primaryUpdateStrategy が監視対象に設定されているクラスターに手動スイッチオーバーが必要かどうかを迅速に報告します。
Feature	pg_audit、auto_explain、および pg_stat_statements の shared_preload_libraries を透過的に処理します。
Feature	pg_stat_statements、pgaudit、または auto_explain オプションが postgresql パラメーターセクションに追加される場合、PostgreSQL の shared_preload_libraries の自動構成。
Feature	k8s.enterprisedb.io/reload ラベルをサポートして、Prometheus エクスポーターのカスタムモニタリング/アラートメトリックに使用される構成マップとシークレット、または証明書の保存に使用される構成マップとシークレットの自動リロードを細かく制御します。
Feature	kubectl の cnp プラグインに reload コマンドを追加して、インスタンスで調整ループをトリガーします。
Feature	advanced PodAffinity および AdditionalPodAntiAffinity を介して、ポッドアフィニティおよびアンチアフィニティ構成の制御を向上させます。
Feature	任意の時点で、少なくとも 1 つのプライマリインスタンスを実行する必要があることにより、プライマリインスタンス用に別個の PodDisruptionBudget を導入します。
Security fix	.spec.certificates.clientCASecret および spec.certificates.replicationTLSSecret オプションを追加して、カスタムクライアント証明書を定義します。
Security fix	PostgreSQL サーバーの権限と証明書。クライアントを認証し、PostgreSQL ノード間の安全な通信を行うために使用されます。
Security fix	.spec.backup.barmanObjectStore.endpointCA オプションを追加して、Barman のバックアップオブジェクトストアのエンドポイントのカスタム認証局バンドルを定義します。
Bug fix	Prometheus エクスポーターでヒストグラムを正しく解析します。
Bug fix	クラスター用にオペレーターが作成したサービスを調整します。

1.2.97 EDB Postgres for Kubernetes 1.6.0 release notes

リリース日 2021 年 7 月 12 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Feature	レプリカモード実験的 クラスターをソースクラスターのレプリカとして作成できます。レプリカクラスターには、指定されたプライマリと任意の数のスタンバイがあります。

次のページに続く

表 101 – 前のページからの続き

Type	Description
Feature	EDB Postgres Advanced Server イメージでの EDB Audit のサポート。
Feature	.spec.postgresql.promotionTimeout パラメーターを追加して、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。デフォルトは 40000000 秒です。
Feature	.spec.affinity.podAntiAffinityType パラメーターを追加します。これは、preferred デフォルトに設定して preferredSchedulingIgnoredDuringExecution が使用されるか、required に設定して requiredDuringSchedulingIgnoredDuringExecution になります。
Security fix	ライセンスキーがログに表示されないようにします。
Change	PVC とポッドを削除する際の競合状態が修正され、オペレーターが新しいポッドを作成できなくなりました。
Change	マネージャーが構成変更時に PostgreSQL の再起動の必要性を検出できない競合状態を修正しました。
Change	アノテーションのないクラスターでの kubectl-cnp のパニックを修正しました。
Change	デバッグする一部のログメッセージのレベルを下げました。
Change	サーバー CA および TLS インジェクションの E2E テスト。

1.2.98 EDB Postgres for Kubernetes 1.5.1 release notes

リリース日 2021 年 6 月 11 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Change	Red Hat OpenShift でのオペレーター展開での自動更新を妨げる CRD 検証のバグを修正します。
Change	シークレットを使用してオペレーターの構成を渡すことを許可します。

1.2.99 EDB Postgres for Kubernetes 1.5.0 release notes

リリース日 2021 年 6 月 17 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Feature	pg_basebackup ブートストラップ方法を導入して、Kubernetes の外部でも、同じメジャーバージョンの既存の PostgreSQL インスタンスのコピーとして新しい PostgreSQL クラスターを作成します。

次のページに続く

表 103 – 前のページからの続き

Type	Description
Feature	クラスターリソースの Affinity セクションに Kubernetes の許容範囲のサポートを追加し、ユーザーは必要なテイントを持つ Kubernetes ノードに PostgreSQL インスタンスを分散できます。
Feature	より決定的で反復可能な展開を行うには、`<image>:<tag>@sha256:<digestValue>`形式を介してイメージ名へのダイジェストの指定を有効にします。
Security fix	サーバー証明書と、それに署名した関連認証局のシークレットを定義することにより、PostgreSQL サーバーを認証するように TLS 証明書をカスタマイズします。
Security fix	内部および自動的に管理されるストリーミングレプリカの WAL レシーバープロセスの ssl モードを require から verify-ca に上げます。
Change	kubectl の cnp プラグインの promote サブコマンドを強化して、ポッドの完全な名前ではなくノード番号のみを受け入れます。
Change	クラスター名に DNS-1035 検証スキームを採用しますサービス名が継承されます。
Change	スタンバイインスタンスのクローンを作成するとき、または pg_basebackup メソッドを使用してブートストラップするときに、ストリーミングレプリケーション接続を強制します。
Change	物理ベースのバックアップに関して、Backup リソースを beginWal、endWal、beginLSN、endLSN、startedAt、および stoppedAt と統合します。
Documentation fix	オペレーターとオペランドコンテナによって公開されるポートのリストを提供します。
Documentation fix	データベースワークロードのストレージと PostgreSQL をベンチマークするための cnp-bench helm チャートとガイドラインを紹介します。
E2E test fix	Kubernetes 1.21 をテストします。
E2E test fix	オペレーターの高可用性のテストを追加します。
E2E test fix	ノードドレインのテストを追加します。
Bug fix	リカバリ操作中の pg_ctl start のタイムアウトが短すぎます。
Bug fix	オペレーターは PVC での直接イベントを監視していません。
Bug fix	barmanObjectStore バックアップの immediateCheckpoint と jobs パラメーターの処理を修正しました。
Bug fix	バックアップから復元するときに空のログ。

1.2.100 EDB Postgres for Kubernetes 1.4.0 release notes

リリース日 2021 年 5 月 18 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Feature	JSON 形式の PostgreSQL エラーメッセージの標準出力ログ。

次のページに続く

表 104 – 前のページからの続き

Type	Description
Feature	Prometheus エクスポーターの PostgreSQL メトリックの基本セットを提供します。
Feature	kubectl の cnp プラグインに restart コマンドを追加して、特定の PostgreSQL クラスターのポッドをロールアウト方法で再起動します。
Security fix	ポッドの readOnlyRootFile システムセキュリティコンテキストを設定します。
Change	重要 YAML マニフェストを使用してクラウドネイティブ PostgreSQL オペレーターを展開したことがある場合、新しいバージョンをインストールする前に既存のオペレーター展開を削除する必要があります。これは、オペレーターによって直接管理されるラベルとラベルセレクターの変更に伴う他の Kubernetes API との競合を回避するために必要です。 1.4.0 へのアップグレードの詳細については、クラウドネイティブの PostgreSQL ドキュメントを参照してください。
Change	オペレーターによって自動的に定義されるラベルを修正し、名前を control-plane:controller-manager から app.kubernetes.io/name:cloud-native-postgresql に変更します。
Change	メトリック名を Prometheus エクスポーターの TCP ポートに割り当てます。
Change	PostgreSQL のメトリックエクスポーター接続の application_name として cnp_metrics_exporter を設定します。
Change	可能な場合は、postgres データベースの代わりに Prometheus エクスポータのモニタリングクエリーにアプリケーションデータベースを使用します。
Documentation fixes	モニタリングクエリのカスタマイズ。
Documentation fixes	オペレーターのアップグレード手順。
Bug fix	pg_basebackup を呼び出すときは、-R の使用を避けてください。
Bug fix	ステータスを取得するときにエラーログからスタックトレースを削除します。

1.2.101 EDB Postgres for Kubernetes 1.3.0 release notes

リリース日 2021 年 4 月 23 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Feature	ラベルとアノテーションの継承。
Feature	コンテナごとにリソース制限を設定します。
Security fix	Red Hat OpenShift での制限付きセキュリティコンテキスト制約のサポートにより、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します。
Security fix	非ルート、非特権、および特権エスカレーションなしで実行するようにオペレーターによって明示的に定義されたポッドセキュリティコンテキスト。
Change	ポート 9187 でリッスンする Prometheus エクスポータエンドポイントポート 8000 は、API サーバーとのインスタンス調整のために予約されています。

1.2.102 EDB Postgres for Kubernetes 1.2.1 release notes

リリース日 2021 年 4 月 6 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Feature	ScheduledBackup はバックアップの所有者ではなくなりました。つまり、ScheduledBackup オブジェクトが削除されてもバックアップは削除されません。
Security fix	RHSA-2021:1024 セキュリティアドバイザリ重要) を解決するための ubi8-minimal イメージの更新。

1.2.103 EDB Postgres for Kubernetes 1.2.0 release notes

リリース日 2021 年 3 月 31 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Feature	Prometheus の postgres_exporter と互換性のある構文を使用して、ConfigMap および Secret オブジェクトとしてカスタムモニタリングクエリの実験的なサポートを導入します。
Feature	Operator Lifecycle ManagerOLM 展開をサポートし、その後の OperatorHub.io でのプレゼンスを提供します。
Feature	全社的に使用できるようにライセンスキーのサポートを拡張します以前の制限は単一のクラスター名前空間にのみ制限されていました。
Feature	米国国防総省 DoD の国防情報システム局 DISA およびインターネットセキュリティセンター CIS のガイドラインを適用し、Dockle とのパイプラインで直接検証することにより、コンテナのセキュリティを強化します。

1.2.104 EDB Postgres for Kubernetes 1.1 release notes

リリース日 2021 年 3 月 3 日

Kubernetes 用 EDB Postgres のこのリリースには、次のものが含まれます。

Type	Description
Feature	kubectl cnp status を追加して、JSON および YAML 出力を含むクラスターのステータスをきれいに印刷します。
Feature	kubectl cnp 証明書を追加して、クライアントアプリケーションの TLS 認証を有効にします。

次のページに続く

表 108 – 前のページからの続き

Type	Description
Feature	-ro サービスを追加して、使用可能なホットスタンバイレプリカのみに接続をルーティングし、クラスターのプライマリインスタンスからの読み取り専用クエリのオフロードを有効にします。
Feature	クラスターを maxSyncReplicas より低い値にスケールダウンしたロールバック。
Feature	元のプライマリを降格する前にチェックポイントを要求します。
Feature	SIGINT 信号高速シャットダウンを SIGTERM の PostgreSQL プロセスに送信します。

1.2.105 EDB Postgres for Kubernetes 1.0 release notes

リリース日 2021 年 2 月 4 日

最初のメジャーリリースは、API グループ postgresql.k8s.enterprisedb.io/v1 にクラスター、バックアップ、および ScheduledBackup を実装します。これらのリソースを使用して、次の主な機能を使用して Kubernetes 内で PostgreSQL クラスターを作成および管理します。

Type	Description
Feature	外部ツールを必要とせずに、高可用性を実現する Kubernetes API サーバーとの直接統合。
Feature	最も調整されたレプリカを昇格させることによるプライマリインスタンスのフェールオーバー。
Feature	レプリカの自動再作成。
Feature	選択したレプリカを昇格させることによるプライマリインスタンスの計画的なスイッチオーバー。
Feature	スケールアップ/ダウン機能。
Feature	任意の数のインスタンスの定義最小 1 - 1 つのプライマリサーバー。
Feature	アプリケーションをクラスターの唯一のプライマリサーバーに接続するための読み取り/書き込みサービスの定義。
Feature	アプリケーションを読み取りワークロードのインスタンスに接続するための読み取りサービスの定義。
Feature	PVC テンプレートを使用したローカル永続ボリュームのサポート。
Feature	ポッドの永続ボリュームストレージの再利用。
Feature	PostgreSQL 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更新。
Feature	TLS 接続とクライアント証明書認証。
Feature	S3 互換オブジェクトストアへの継続的なバックアップ。
Feature	S3 互換オブジェクトストアバックアップからのフルリカバリーとポイントインタイムリカバリー。
Feature	同期レプリカのサポート。
Feature	nodeSelector プロパティを介したノードアフィニティのサポート。
Feature	PostgreSQL エラーメッセージの標準出力ログ。

1.2.106 EDB Postgres for Kubernetes 0.7 release notes

リリース日 2020 年 12 月 31 日

Type	Description
Feature	フェイルオーバーまたはスイッチオーバーに続いて、必要に応じて`pg_rewind`をサポートして、元のプライマリをスタンバイとして機能させます。
Feature	ストレージクラスで許可されている場合、永続ボリューム拡張サポートを追加します。
Enhancement	PostgreSQL のメトリックエクスポーターを強化します。
Feature	Kubernetes バージョン 1.20 のサポートを追加します。
Feature	Kubernetes バージョン 1.15 のサポートを終了します。
Feature	Github アクションへの変換を含む E2E テストのリファクタリング。
Fix	バグ修正とコードの改善。

1.2.107 EDB Postgres for Kubernetes 0.6 release notes

リリース日 2020 年 12 月 4 日

Type	Description
Feature	`recoveryTarget`と呼ばれる新しいブートストラップ方法オプションを介して、タイムスタンプ、ターゲット名、またはトランザクション ID、およびタイムラインの仕様に基づいて、ポイントインタイムリカバリーを追加します。
Feature	`minSyncReplicas`および`maxSyncReplicas`クラスターオプションを介して同期ストリーミングレプリケーションのサポートを追加し、任意の時点での同期スタンバイサーバーの予想最小および最大数をそれぞれ定義しますデフォルトでは無効
Feature	EDB Postgres Advanced Server をサポートします。
Feature	空のクラスター`initDb`のブートストラップの`initdb`オプションを構成します。
Feature	EPAS との Redwood 互換性レベルを有効/無効にします。
Feature	Prometheus のメトリックのエクスポートのための新しいフレームワークでインスタンスマネージャーを拡張します。現在 pg_stat_archiver のみをサポートしています。
Feature	init コンテナの代わりに Kubernetes ジョブを使用して、クラスター初期化手順スタンバイの作成とリカバリを実行し、可観測性を向上させます。
Feature	`kubectl describe`および`kubectl get events`が使用する Kubernetes イベントを記録します。
Feature	ポッド、PVC、ジョブに Kubernetes の期待を導入して、競合状態を防止します。
Feature	PostgreSQL の`application_name`をポッド/インスタンスの名前に設定します。
Feature	`fullRecovery`ブートストラップモードは`recovery`に名前が変更され、ポイントインタイムリカバリーにも対応

次のページに続く

表 111 – 前のページからの続き

Type	Description
Fix	バグ修正とコードの改善。

1.2.108 EDB Postgres for Kubernetes 0.5 release notes

リリース日 2020 年 11 月 20 日

Type	Description
Feature	各 PostgreSQL クラスターの独立した認証局 CA の自動プロビジョニング。
Feature	クライアント/サーバー通信を暗号化するための TLS / SSL 接続の透過的かつネイティブサポート。
Enhancement	唯一の `REPLICATION` 特権と II を持つ `streaming_replica` と呼ばれる専用の固定データベースユーザーを介して、スタンバイストリーミングレプリケーションチャネルのセキュリティが向上します。 `streaming_replica` ユーザーを認証するためにクラスター認証局によって署名された自動管理 X.509 TLS 証明書。
Enhancement	`streaming_replica` ユーザーを認証するためにクラスター認証局によって署名された自動管理 X.509 TLS 証明書を介して、スタンバイストリーミングレプリケーションチャネルのセキュリティが向上します。
Enhancement	変更可能な Webhook を介して PostgreSQL の構成機能を向上させ、オペレーターによって直接管理されるパラメーターをユーザーが変更できないようにします。
Enhancement	ユーザーが指定した構成オプションをクラスター状態のデフォルト値と統合するデフォルト Web フックを介して、PostgreSQL の構成機能を向上させます。
Enhancement	PostgreSQL インスタンスのリロードと再起動の自動管理を介して、PostgreSQL の構成機能を向上させます。
Feature	バックアップからのリカバリーに続いて、PostgreSQL クラスターのカスタムおよび独立した構成を有効にします。
Feature	クラスターの現在のステータスつまり正常、フェールオーバー進行中、スイッチオーバー進行中などを伝えます。
Feature	k8s.2ndq.io から k8s.enterprisedb.io への API 変更。

1.2.109 EDB Postgres for Kubernetes 0.4 release notes

リリース日 2020 年 11 月 5 日

Type	Description
Feature	バックアップからの完全なリカバリーのサポート。
Feature	`bootstrap` セクションを追加して、クラスター `initDB` または `fullRecovery` を開始する方法を構成します。

次のページに続く

表 113 – 前のページからの続き

Type	Description
Feature	デフォルトと検証の Webhook を導入します。
Feature	構成を簡素化します構成よりもコンベンション。
Feature	ローリングアップグレードを同じ PostgreSQL メジャーバージョンに制限します。
Feature	エンドツーエンドテストは、GKE、AKS、および GKS で実行されます。
Enhancement	ドキュメントの改善。
Enhancement	バグ修正とマイナーな改善。

1.2.110 EDB Postgres for Kubernetes 0.3 release notes

リリース日 2020 年 9 月 25 日

Type	Description
Feature	PostgreSQL 13 のサポート。
Feature	`emptyDir`ボリュームストレージのサポートを削除します。
Feature	`nodeMaintenanceWindow`パラメーターを介したローカルストレージを使用した PostgreSQL クラスターのノードメンテナンスのサポート。
Enhancement	デフォルト、必須、および固定値をサポートする辞書を使用して、PostgreSQL 構成管理を向上させます。
Feature	クラスター間通信の認証方法として MD5 を使用し、シークレットでパスワードを自動作成します。
Feature	「trust」認証方法の必要性を削除します。
Feature	`spec.postgresql.pg_hba`はオプショナルになり、ポッド間の通信に必要な MD5 認証とポッド内通信の `peer`がデフォルトになります。
Feature	「使用不可」のアノテーションサポートを削除します。PVC を削除した後、対応するポッドを削除するだけで済みます。
Feature	暗黙の 30 日間の試用版を含むライセンスキーのサポート。
Fix	バグ修正とマイナーな改善。
Feature	以前のバージョンからの更新はサポートされていません。

1.2.111 EDB Postgres for Kubernetes 0.2 release notes

リリース日 2020 年 8 月 11 日

Type	Description
Feature	PostgreSQL 12 に加えて、PostgreSQL 10 および 11 がサポートされるようになりました。
Feature	オペレーターのベースイメージとしての UBI の使用 OpenShift サポート。

次のページに続く

表 115 – 前のページからの続き

Type	Description
Feature	新しい Backup と ScheduledBackup CRD により、ユーザーは S3 プロトコル AWS S3、MinIO、MinIO Gateway などに準拠したオブジェクトストアでクラスターの物理的バックアップを取得できます。
Feature	S3 プロトコルに準拠したオブジェクトストアへの WAL アーカイブのサポート。
Enhancement	E2e テストインフラの改善。
Fix	バグ修正とマイナーな改善。

1.2.112 EDB Postgres for Kubernetes 0.1 release notes

リリース日 2020 年 4 月 3 日

Type	Description
Feature	ポッドによる既存の永続ストレージの再利用ローリング更新に必要。
Feature	オペレーターコンテナイメージと PostgreSQL コンテナイメージ間の独立性ローリング更新で必要 - これにより、コミュニティ PostgreSQL イメージの使用が有効になります。
Feature	最初にすべてのレプリカを更新することによる、オペレーターとクラスター全体の PostgreSQL の新しいバージョンへの更新のローリング更新。レプリカが更新されるか `unsupervised`オプション 、ユーザーが手動でトリガーする `supervised`オプション と、スイッチオーバーは Kubernetes によって完全に管理できます。
Feature	ポッドの進行中の更新のフレームワークを確認します。
Enhancement	クラスターステータス情報の改善。
Enhancement	Kubernetes 1.18 の E2E テスト
Enhancement	フェイルオーバーの E2E パフォーマンステスト 5 秒未満。
Enhancement	スイッチオーバーの E2E テストの改善。
Feature	PostgreSQL の `cluster_name` GUC のサポート。
Feature	新しいドキュメントセクション `Exposing Postgres services`。
Feature	新しいドキュメントセクション `Security`。

1.2.113 EDB Postgres for Kubernetes 0.0.1 release notes

リリース日 2020 年 3 月 5 日

Type	Description
Feature	PostgreSQL 12.2 コンテナイメージ。
Feature	最も調整されたレプリカを昇格させることによる、プライマリインスタンスのフェイルオーバーを介した自己修復機能。
Feature	レプリカの自動再作成を介した自己修復機能。

次のページに続く

表 117 – 前のページからの続き

Type	Description
Feature	選択したレプリカを昇格させることによる、プライマリインスタンスの計画的なスイッチオーバー。
Feature	スケールアップ/ダウン機能。
Feature	任意の数のインスタンスの定義最小 1 - 1 つのプライマリサーバー。
Feature	アプリケーションをクラスターの唯一のプライマリサーバーに接続するための **読み取り-書き込み** サービスの定義。
Feature	アプリケーションを読み取りワークロードのインスタンスに接続するための **読み取り専用** サービスの定義。
Feature	PVC テンプレートを使用したローカル永続ボリュームのサポート。
Feature	PostgreSQL エラーメッセージの標準出力ログ。

第2章　Getting Started

2.1 Architecture

> **ヒント**
>
> より深く理解するには、*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。

2.1.1 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 に基づく**

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。 Shipping** 、これは 10 年以上、世界中の何百万ものユーザーが本番環境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。

重要

PostgreSQL を使用したストレージレベルのレプリケーションをお勧めしませんが、CloudNativePG™ クラスターの EDB Postgres ® AI では、その戦略を採用できます。詳細については、KubeCon NA 2022 での Chris Milsted と Gabriele Bartolini による Data On Kubernetes, Deploying And Running PostgreSQL And Patterns For Databases In a Kubernetes Cluster というタイトルの講演を参照してください。

このトピックが詳細にカバーされている場所。

2.1.2 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。

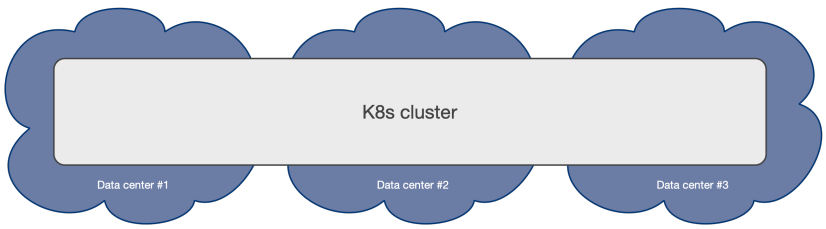


図 1: Kubernetes cluster spanning over 3 independent data centers

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、*Scheduling* が含まれます宣言的な方法でのワークロード、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。

重要

各オペレーター展開は、ローカル Kubernetes クラスター内の操作のみを管理できます。制御されたスイッチオーバーや予期しないフェイルオーバーなどの Kubernetes クラスターにわたる操作の場合、GitOps などを介して手動で、または高レベルのクラスター管理ツールを使用して、調整を処理する必要があります。

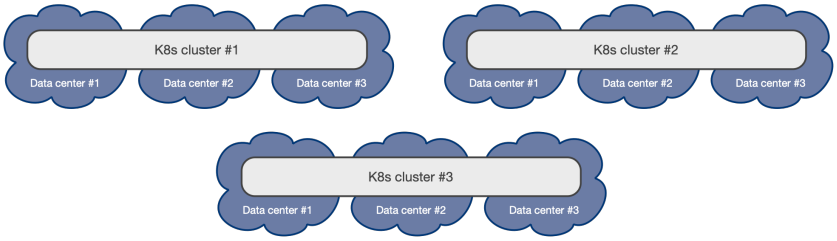


図 2: Example of a multiple Kubernetes cluster architecture distributed over 3 regions each with 3 independent data centers

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 をレベルに押し上げます可能な限りゾーンを停止する必要があります。つまり、CloudNativePG™ Cluster の EDB Postgres ® AI クラスター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。

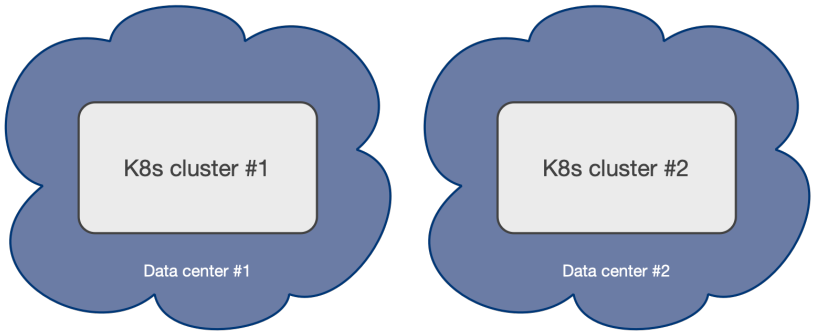


図 3: Example of a Kubernetes architecture with only 2 data centers

ヒント

Kubernetes の取り組みの初期段階にある場合は、このドキュメントをインフラストラクチャチームと共有してください。 2 つのデータセンターのセットアップは、従来のベアメタルまたは VM ベースのインフラストラクチャから Kubernetes への「リフトアンドシフト」移行の結果である可能性があり、3+ ゾーンシナリオで Kubernetes が提供するメリットは知られていない可能性があります、またはインフラストラクチャアーキテクチャの設計時に対処されました。最終的に、他の 2 つに接続された 3 番目の物理的な場所は、日常の複雑さをアプリケーションレベルから物理的インフラストラクチャレベルに移動することにより、インフラストラクチャの全体的なコストを削減するため、組織にとって検討すべき有効なオプションを表す可能性があります。

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 EDB Postgres ® AI for CloudNativePG™ Cluster のスコープ内ではありません。

重要

CloudNativePG™クラスターの EDB Postgres ® AI は、高レベルのオペレーターまたは管理ツールを介してさまざまな Kubernetes クラスターにわたって PostgreSQL アクティブ/パッシブトポロジを調整するために必要なすべてのプリミティブとプローブを提供します。

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。

ヒント

一般的な経験則として、Postgres ノードは 3 の倍数で展開します。理想的には、アベイラビリティーゾーンごとに 1 つのノードを使用します。 3 つのノード 1 つのプライマリと 2 つのスタンバイレプリカを含む PostgreSQL クラスターが異なるノードに分散され、フォールトトレランスと可用性が向上するため、3 つのノードが最適な数です。

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。

重要

この方法は、コンピューティングリソースと、ローカルに接続されたディスクを使用する場合はストレージの両方の点で、PostgreSQL ワークロードを他のワークロードから分離する最も簡単な方法です。異なる PostgreSQL クラスターが同じノードを共有する場合がありますが、ラベルとテイントを使用して、ノードが特定の `Cluster` の単一のインスタンス専用になるようにすることにより、これをさらに進めることができます。

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 テイントの使用をお勧めします。

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

2.1.3 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

重要

Cluster 構成の `managed.services` セクションを介して上記のサービスを構成できます。これは、サービスの数を減らし、タイプを選択することにより実行できますデフォルトは `ClusterIP` です。詳細については、 *Service Management* を参照してください。

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。

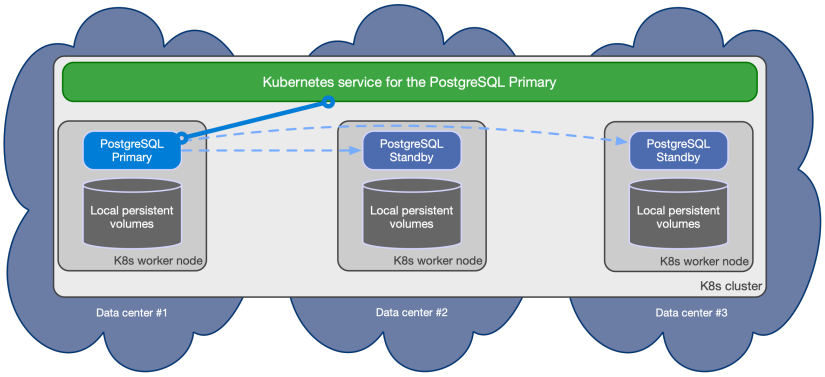


図 4: Bird-eye view of the recommended shared nothing architecture for PostgreSQL in Kubernetes

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。

参考

CloudNativePG™クラスターの EDB Postgres ® AI が同期設定を含む PostgreSQL レプリケーションに依存する方法の詳細については、 *Logical Replication* を参照してください。

参考

同じ Kubernetes クラスター内のステートレスアプリケーションから CloudNativePG™クラスターの EDB Postgres ® AI に接続する方法については、 *Connecting from an application* を参照してください。

参考

接続プーラーとして PgBouncer を利用し、アプリケーションと PostgreSQL クラスターの間にアクセスレイヤーを作成する方法については、 *PgBouncer* を使用した接続プーリング を参照してください。

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。

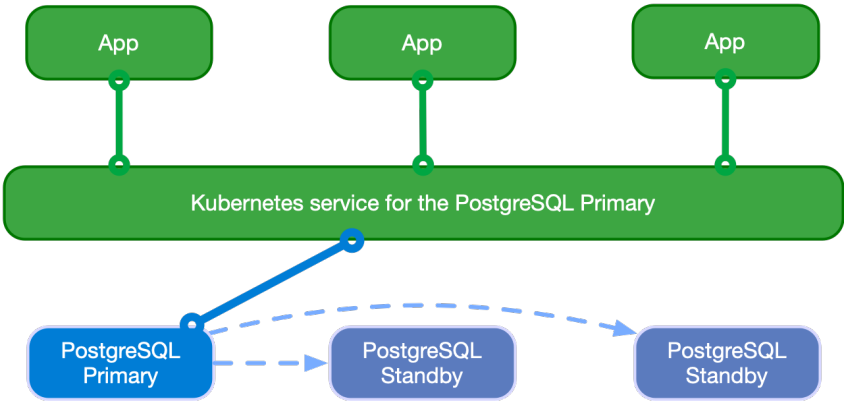


図 5: Applications writing to the single primary

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、高可用性の為、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。

読み取り専用ワークロード

重要

アプリケーションは制限に注意する必要があります Hot Standby

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。

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。

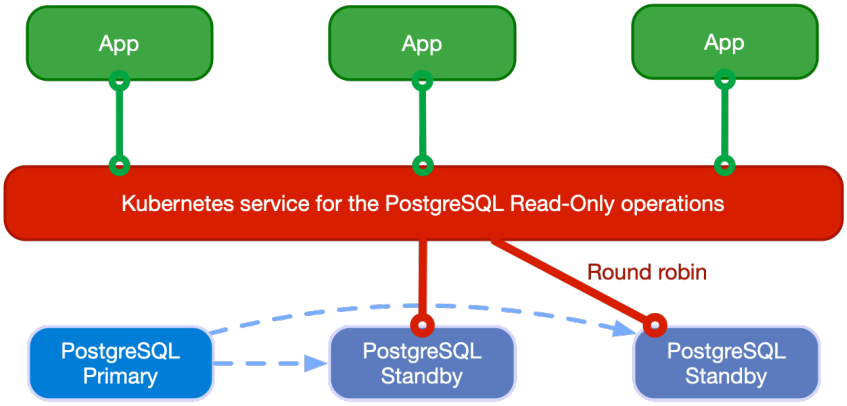


図 6: Applications reading from hot standby replicas in round robin

2.1.4 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 クラスターにわたる 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** バックアップ頻度と *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。

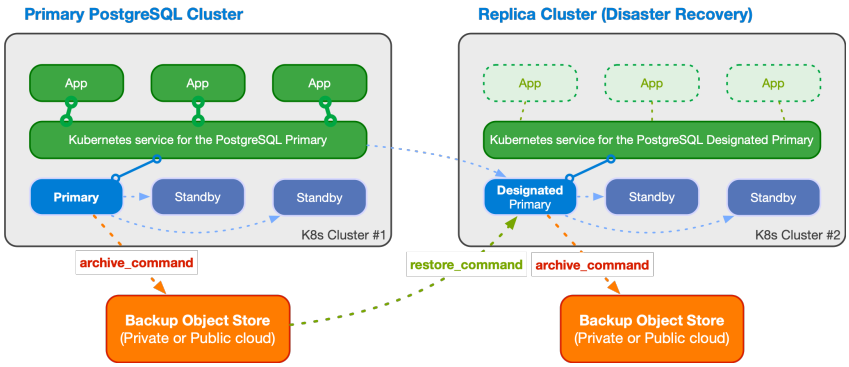


図 7: An example of multi-cluster deployment with a primary and a replica cluster

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。

> **重要**
>
> CloudNativePG™クラスターの EDB Postgres ® AI を使用すると、宣言的構成を介して分散トポロジを制御でき、GitOps を含むインフラストラクチャ・アズ・コード IaC プロセスの一部としてこれらの手順を自動化できます。

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。

> **参考**
>
> 物理レプリカクラスターがどのように動作するか、およびさまざまな Kubernetes クラスターにわたって読み取り専用クラスターを使用した分散トポロジを定義する方法の詳細については、 **レプリカクラスター** を参照してください。このアプローチは、グローバルディザスターリカバリーと高可用性 HA 戦略を大幅に強化できます。

2.2 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。

2.2.1 Kubernetes の用語

Node

ノードは、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 **ノード**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

ポッドは、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

シークレットは、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

ストレージクラス 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、または**ストレージクラス**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、**永続的なボリューム要求**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

永続的なボリューム要求 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

名前空間は、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**仮想クラスター**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

ロールベースセキュリティ 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

オペレーターは、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。

2.2.2 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。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ます。レプリカは、**スタンバイ** または **セカンダリ** サーバーとしても知られています。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 **レプリカ**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 クラスターとして意図され、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、通常は 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、 **スーパーユーザー** 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 `storage` を含むメインボリュームと WAL 用ボリューム `walStorage` に属する関連 PVC のグループです。

RTO 「recovery Time Objective」の頭字語、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。

RPO 「目標復旧ポイント」の頭字語で、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。

2.2.3 クラウド用語

リージョン クラウドの **リージョン** は、 **アベイラビリティーゾーン**で編成された隔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** ゾーン*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。

2.2.4 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。

2.3 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。

2.3.1 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。

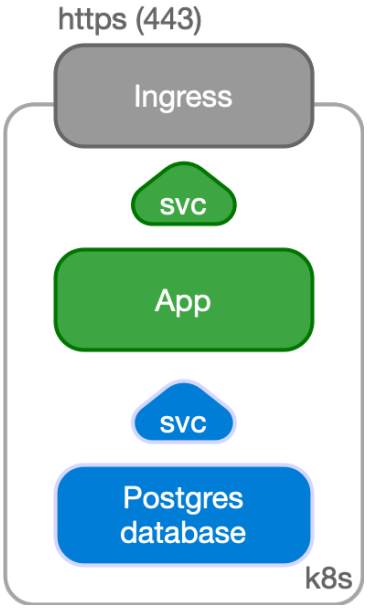


図 8: Application and Database inside Kubernetes

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。

2.3.2 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。

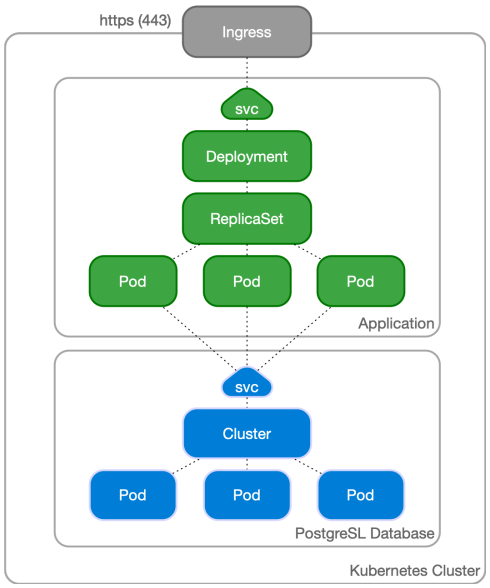


図 9: Close-up view of application and database inside Kubernetes

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。

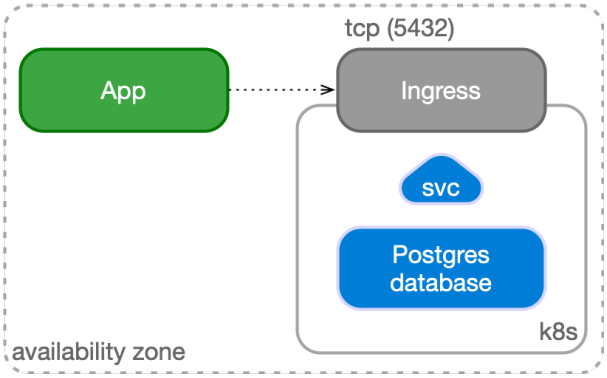


図 10: Application outside Kubernetes

2.4 Installation and upgrades

> **参考**
>
> Red Hat OpenShift Container Platform に Cloud Native PostgreSQL をインストールする方法については、*Red Hat OpenShift* を参照してください。

セクション。

> **警告**
>
> OLM operatorhub.io 経由は、CloudNativePG™クラスターの EDB Postgres AI のインストール方法としてサポートされなくなりました。

2.4.1 Kubernetes へのインストール

EDB サブスクリプショントークンの取得

重要

EDB サブスクリプショントークンを取得して、 CloudNativePG™クラスターに EDB Postgres AI をインストールする必要があります。トークンがないと、EDB プライベートソフトウェアリポジトリにアクセスできません。

CloudNativePG™クラスターに EDB Postgres ® AI をインストールするには、EDB Repos 2.0 トークンが必要です。EDB プライベートソフトウェアリポジトリへのアクセス。

EDB Account Profile にアクセスして、トークンを取得できます。まだログインしていない場合は、サインインする必要があります。

アカウントプロフィールページの **Repos 2.0 Token** label の横に使用するトークンが表示されます。デフォルトでは、トークンは隠されています。[Show] ボタン目のアイコンをクリックして、表示します。

トークンにより、スタンダードまたはエンタープライズという 2 つのリポジトリのいずれかにアクセスできます。

- `standard` - オペレーターと EDB Postgres 拡張オペランドイメージが含まれます。
- `enterprise` - オペレーターと EDB Postgres Advanced および EDB Postgres Extended オペランドイメージが含まれます。

サブスクリプションによって決定される関連する値を環境変数 `EDB_SUBSCRIPTION_PLAN` として設定します。

```
EDB_SUBSCRIPTION_PLAN=enterprise
```

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

```
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。

これで、インストールを続行できます。

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。

オペレーターマニフェストを直接使用する

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_$EDB_SUBSCRIPTION_PLAN \
  --docker-password=$EDB_SUBSCRIPTION_TOKEN
```

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。

サブスクリプションプランに応じて、2 つの異なるマニフェストを使用できます。

- 標準 latest standard operator manifest 。
- Enterprise latest enterprise operator manifest 。

次のコマンドを実行して、オペレーターの最新バージョンのマニフェストをインストールできます。

```
kubectl apply --server-side -f \
  https://get.enterprisedb.io/pg4k/pg4k-$EDB_SUBSCRIPTION_PLAN-1.27.0.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
```

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、 :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>` 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、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公式 docs

そしてこれによって issue 。 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。

2.4.2 デプロイメントの詳細

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デプロイメントの名前をカスタマイズできます。

`kubectl` の `describe` コマンドを使用して、詳細情報を取得できます。

```
$ 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。

> **参考**
>
> いくつかのデフォルトオプションをオーバーライドすることにより、オペレーターのデフォルト動作を変更できます。詳細については、 *Operator configuration* セクションを参照してください。

2.4.3 アップグレード

> **重要**
>
> *Introduction and Release Notes* をよくお読みください

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。

CloudNativePG™クラスターオペレーターの EDB Postgres ® AI のアップグレードは、2 段階のプロセスです。

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。

> **参考**
>
> このプロセスは、 *Rolling Updates* ページで詳細に説明されています。

> **重要**
>
> *primaryUpdateStrategy* が `unsupervised` のデフォルト値に設定されている場合、オペレーターのアップグレードは PostgreSQL クラスターでスイッチオーバーをトリガーし、通常は無視できるダウンタイムを発生します。 PostgreSQL クラスターにインスタンスが 1 つだけある場合、 *supervised* 値は `primaryUpdateStrategy` でサポートされていないため、インスタンスは自動的に再起動されます。いずれの場合も、アプリケーションは PostgreSQL に再接続する必要があります。

デフォルトのローリング更新動作は、インスタンスマネージャーのインプレース更新に置き換えることができます。このアプローチでは、PostgreSQL インスタンスの再起動が必要ないため、クラスター内のスイッ

チオーバーを回避します。この機能はデフォルトで無効になっていますが、以下で詳細に説明します。

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` 。

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。

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。

Introduction and 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。

1.27.0 または 1.26.1 へのアップグレード

重要

すべての EDB Postgres ® AI for CloudNativePG™ Cluster ユーザーは、バージョン 1.27.0、または少なくとも現在のマイナーリリース 1.26.1 などの最新の安定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。

バージョン 1.27 では、*Liveness Probe* のデフォルト動作の変更を導入しました。 *shutdown of an isolated primary*

`livenessProbeTimeout` 30 秒以内。

この動作がご使用の環境に適していない場合は、次の構成で liveness プローブの**分離チェック** を無効にできます。

```
spec:
  probes:
    liveness:
      isolationCheck:
        enabled: false
```

以前のマイナーバージョンから 1.26 へのアップグレード

重要

すべての EDB Postgres ® AI for CloudNativePG™ Cluster ユーザーは、バージョン 1.26.1、または少なくとも現在のマイナーリリース 1.25.x などの最新の安定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。

警告

マネージャーコンポーネント #6623 の起動プローブの変更のため、オペレーターのアップグレードは、インプレース更新が有効になっている *ENABLE_INSTANCE_MANAGER_INPLACE_UPDATES=true* 場合でも、PostgreSQL クラスターの再起動がトリガーされます。アップグレード後に、アプリケーションは PostgreSQL に再接続する必要があります。

`Cluster .status` のバックアップメトリックとフィールドの非推奨

Barman Cloud のバージョン 1.26.0 から始まった EDB Postgres ® AI for CloudNativePG™ Cluster の CNP-I プラグインに基づいたバックアップとリカバリーにとらわれないアプローチへの移行に伴い、`.status` の次のフィールドの非推奨期間を開始します `Cluster` リソースのセクション

- `firstRecoverabilityPoint`
- `firstRecoverabilityPointByMethod`
- `lastSuccessfulBackup`
- `lastSuccessfulBackupByMethod`
- `lastFailedBackup`

次の Prometheus メトリックも非推奨です。

- `cnp_collector_first_recoverability_point`
- `cnp_collector_last_failed_backup_timestamp`
- `cnp_collector_last_available_backup_timestamp`

警告

Barman Cloud などのプラグインベースのバックアップおよびリカバリーソリューションに移行した場合、これらのフィールドとメトリックは同期されず、更新されません。 Barman Cloud とボリュームスナップショットのインコアサポートにまだ依存しているユーザーは、当面はこれらのフィールドを使用し続けることができます。

新しいプラグインベースのアプローチでは、複数のバックアップ方法が同時に動作でき、それぞれがバックアップとリカバリーの独自のタイムラインで動作します。たとえば、一部のプラグインは WAL アーカイブなしでスナップショットを提供する場合がありますが、他のプラグインは継続的なアーカイブをサポートします。

この柔軟性のため、`Cluster` リソースで一元化されたステータスフィールドを維持することは、構成されているすべてのバックアップ方法の状態を正確に表していないため、誤解を招くまたは混乱を招く可能性があります。このため、これらのフィールドは非推奨になります。

代わりに、各プラグインは、独自のバックアップステータス情報を公開し、監視と操作の認識のためにインスタンスマネージャーにメトリックを提供します。

cnp プラグインの宣言的ハイバネーション

このリリースでは、 kubectl の cnp プラグインは、命令型から *declarative approach for cluster hibernation* に移行します。 hibernate on および hibernate off コマンドは、宣言的な変更を適用してハイバーネーションを有効または無効にする便利なショートカットになりました。hibernate status コマンドは、その目的が標準の status コマンドによって満たされるため、削除されました。

以前のマイナーバージョンから 1.25 へのアップグレード

> **警告**
>
> 上位のマイナーリリースにアップグレードするたびに、すべての中間マイナーリリースのリリースノートとアップグレード手順を確認してください。たとえば、1.23.x から 1.25 に移行する場合は、1.24 および 1.25 のリリースノートとアップグレード手順を必ず読んでください。

1. 25 にアップグレードするときに、既存の 1.24 クラスター構成を変更する必要はありません。

以前のマイナーバージョンから 1.24 へのアップグレード

レプリカクラスターから分散トポロジへ

CloudNativePG™クラスター 1.24.0 の EDB Postgres ® AI の重要な機能強化の 1 つは、レプリカクラスター機能のアップグレードです。

以前のレプリカクラスター機能は、現在「スタンドアロンレプリカクラスター」と呼ばれていますが、複数の Kubernetes クラスターにまたがるディザスターリカバリー DR および高可用性 HA シナリオでは推奨されなくなりました。スタンドアロンレプリカクラスターは、レポート、OLAP、テストデータを使用した開発環境の作成などの読み取り専用ワークロードに最適です。

DR および HA の目的で、CloudNativePG™クラスターの EDB Postgres ® AI は、レプリカクラスターの分散トポロジ戦略を導入しました。この新しい戦略では、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環境、複数のリージョン、および異なるクラウドプロバイダーにまたがる PostgreSQL クラスターを構築できます。また、スイッチオーバー操作を制御する API も提供し、任意の時点で 1 つのクラスターのみがプライマリとして機能します。

この分散トポロジ戦略は、復元力とスケーラビリティを強化し、多様なインフラストラクチャ設定にわたって高可用性とディザスターリカバリー機能を必要とする最新の分散アプリケーションのための堅牢なソリューションになります。

分散 PostgreSQL セットアップに関係するすべての Cluster リソースを変更することにより、以前のレプリカクラスター構成から分散トポロジにシームレスに移行できます。次の手順が実行されていることを確認します。

- externalClusters セクションを構成して、分散トポロジに関係するすべてのクラスターを含めます。保守性と一貫性を維持するために、すべての Cluster リソースで同じ構成を使用することを強くお勧めします。
- 分散トポロジを反映するように.spec.replica スタンザの primary および source フィールドを構成します。primary フィールドには、分散トポロジの現在のプライマリクラスターの名前が含まれている必要がありますが、source フィールドには、各 Cluster リソースの複製元のクラスターの名

前が含まれている必要があります。`enabled` フィールドは、以前に `true` または `false` に設定されていましたが、設定を解除するデフォルトにする必要があることに注意してください。

詳細については、 レプリカクラスターから分散トポロジへ を参照してください。

2.5 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説明します。

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。

重要

Kubernetes クラスターに接続するためには、マシンに `kubectl` がインストールされていること、または OpenShift Local を使用する場合は `oc` がインストールされていることを確認します。 *how to install `kubectl` <https://kubernetes.io/docs/tasks/tools/install-kubectl/>`_* の Kubernetes ドキュメントに従ってください

または how to install `oc <Red Hat OpenShift>` 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。

2.5.1 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。

2.5.2 パート 2 CloudNativePG™クラスターに 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™クラスターの EDB Postgres ® AI のインストールに進むことができます。

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。

2.5.3 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>
```

重要

ラベルとして `k8s.enterprisedb.io/cluster` を使用していることに注意してください。過去に `postgresql` を見たり使用したりしたことがあるかもしれません。このラベルは非推奨であり、将来的に削除される予定です。 *k8s.enterprisedb.io/cluster* を使用してください。

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 16.4 をインストールするには、次を使用できます。

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
   # [...]
spec:
   # [...]
     imageName: docker.enterprisedb.com/k8s_enterprise/edb-postgres-advanced:16
   #[...]
```

EDB Postgres Extended 16 をインストールするには、次を使用できます。

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
   # [...]
spec:
   # [...]
     imageName: docker.enterprisedb.com/k8s_enterprise/edb-postgres-extended:16
   #[...]
```

重要

不変のインフラストラクチャパラダイムでは、常にコンテナイメージの特定のバージョンをポイントする必要があります。クラスター内の更新ポリシーとバージョンの一貫性の点で予期せぬシナリオが発生する可能性があるため、実稼働環境では `latest` や `13` のようなタグを使用しないでください。厳密な決定論的で反復可能な展開の場合、 *<image>:<tag>@sha256:<digestValue>* 形式を介してイメージ名にダイジェストを追加できます。

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

重要

Prometheus と Grafana のインストールは、このプロジェクトの範囲を超えています。このセクションの手順は、実験と説明のみを目的として提供されています。

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のための役立つ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* **スタック**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/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。

参考

上記の helm コマンドの詳細については、 helm install を参照してください。

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 を介して監視できます。

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

  monitoring:
    enablePodMonitor: true
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。

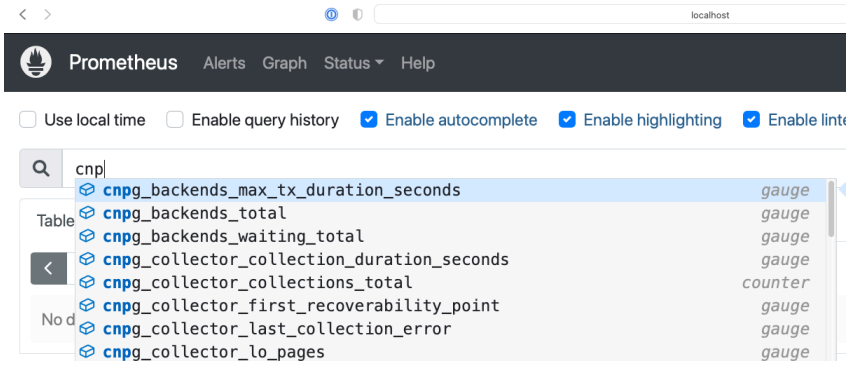


図 11: local prometheus

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 を作成することにより、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/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 12: 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。

第3章　Configuration

3.1 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。

3.1.1 `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"
# ...
```

> **参考**
>
> GUC Grand Unified Configuration としても知られる more information on the available parameters の PostgreSQL ドキュメントを参照してください。 CloudNativePG™クラスターの EDB Postgres ® AI は、PostgreSQL パラメーターの文字列のみを受け入れることに注意してください。

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
```

(次のページに続く)

(前のページからの続き)

```
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、*Logical Replication* 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。

次のパラメーターは **固定**で、オペレーターによって排他的に制御されます。

```
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` から `logical` を設定して、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 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のみて `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
recovery_target_timeline = latest
```

STANDBY_TCP_USER_TIMEOUT が指定されている場合、オペレーターによって管理されているすべてのスタンバイインスタンスに `tcp_user_timeout` パラメーターを設定します。

`tcp_user_timeout` パラメーターは、TCP 接続が強制的にクローズされるまでに、送信データが未確認のままでいられる時間を決定します。この値を調整すると、ネットワークの中断に対するスタンバイインスタンスの応答性を微調整できます。詳細については、PostgreSQL documentation を参照してください。

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。このため、PostgreSQL のすべての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。

> **重要**
>
> 指定されたライブラリが見つからない場合、サーバーは起動に失敗し、CloudNativePG™クラスターの EDB Postgres ® AI の自己修復の試行を妨げ、手動介入が必要になります。コンテンツを直接管理する予定がある場合は、 *shared_preload_libraries* の拡張機能と設定の両方を常にテストしてください。

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。

> **重要**
>
> *shared_preload_libraries* からライブラリを削除するには、クラスター内のすべてのインスタンスを再起動する必要があることに常に注意してください。

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。

重要

declarative extensions の導入により

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"
```

pg_failover_slots の有効化

pg_failover_slots

EDB による拡張機能により、論理レプリケーションスロットがフェールオーバーシナリオでも生き残ることができます。フェールオーバーは、CloudNativePG™クラスターの EDB Postgres の場合と同様、物理的ストリーミングレプリケーションを使用して実装されます。

pg_failover_slots. で始まるパラメーターを構成に追加することにより、pg_failover_slots を有効にできます。上記で説明したように、オペレーターは、これに応じて shared_preload_libraries オプションの pg_failover_slots エントリーを透過的に管理します。

:ref:``Database` マニフェストの必須フィールド <*Database* マニフェストの必須フィールド>` を参照してください。

この拡張機能の詳細については、

さらに、 pg_failover_slots で使用するデータベースごとに、各レプリカがプライマリに接続できるようにする pg_hba セクションにエントリを追加する必要があります。たとえば、 app データベースを pg_failover_slots で使用する場合、次のエントリを pg_hba セクションに追加する必要があります。

```
postgresql:
  pg_hba:
    - hostssl app streaming_replica all cert
```

3.1.2 pg_hba セクション

pg_hba は、ポッドが使用する pg_hba.conf を作成するために使用される PostgreSQL ホストベースの認証ルールのリストです。

重要

*more information on `pg_hba.conf* <https://www.postgresql.org/docs/current/auth-pg-hba-conf.html>`_ については、PostgreSQL ドキュメントを参照してください。

最初に一致するルールが認証に使用されるため、オペレーターによって生成された pg_hba.conf ファイルは 4 つのセクションで構成されていると考えられます。

1. 固定ルール
2. ユーザー定義のルール
3. オプションの LDAP セクション
4. デフォルトルール

固定ルール

```
local all all peer

hostssl postgres streaming_replica all cert map=cnp_streaming_replica
hostssl replication streaming_replica all cert map=cnp_streaming_replica
hostssl all cnp_pooler_pgbouncer all cert map=cnp_pooler_pgbouncer
```

デフォルトルール

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

PostgreSQL 14 から、password_encryption データベースパラメーターのデフォルト値は scram-sha-256 に設定されます。そのため、この PostgreSQL バージョンからのデフォルトの認証方法は scram-sha-256 です。

PostgreSQL 13 以前では、デフォルトの認証方法として md5 を使用します。

結果の pg_hba.conf は次のようになります。

```
local all all peer

hostssl postgres streaming_replica all cert map=cnp_streaming_replica
hostssl replication streaming_replica all cert map=cnp_streaming_replica
hostssl all cnp_pooler_pgbouncer all cert map=cnp_pooler_pgbouncer

<user defined rules>
<user defined LDAP>

host all all all scram-sha-256 # (or md5 for PostgreSQL version <= 13)
```

クラスターマニフェスト内では、次の抜粋にあるように、pg_hba 行が.spec.postgresql.pg_hba のリスト項目として追加されます。

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

上記の例では、安全なチャネル hostssl を介して MD5 パスワード認証必要に応じて scram-sha-256 を使用して、app ユーザーの app データベースへのアクセスを有効にしています。

LDAP 構成

クラスター仕様の postgres セクションの下には、pg_hba.conf ファイルに追加されるルールに変換する LDAP 構成を定義するオプションの ldap セクションがあります。

これは、2 つのモードをサポートします。LDAP セクションで server 、prefix 、および suffix を指定する必要がある simple bind モードと、 server 、baseDN 、binDN 、および ldap パスワードを含むシークレットである bindPassword を指定する必要がある search+bind モード。さらに、 search+bind モードでは、 searchFilter または searchAttribute を指定するオプションがあります。 searchAttribute が指定されていない場合、 uid のデフォルトの 14 が使用されます。

さらに、両方のモードで、 ldapscheme の scheme と port の指定ができます。ただし、スキームもポートも必要ありません。

search+bind のために入力されたこのセクションは次のようになります。

```
postgresql:
  ldap:
    server: openldap.default.svc.cluster.local
    bindSearchAuth:
```

(次のページに続く)

(前のページからの続き)

```
    baseDN: ou=org,dc=example,dc=com
    bindDN: cn=admin,dc=example,dc=com
    bindPassword:
      name: ldapBindPassword
      key: data
    searchAttribute: uid
```

3.1.3 pg_ident セクション

pg_ident は、 CloudNativePG™クラスターの EDB Postgres ® AI がデータディレクトリ内の ident マップファイル pg_ident.conf と呼ばれるを生成および維持するために使用する PostgreSQL ユーザー名マップのリストです。

重要

*more information on `pg_ident.conf* <https://www.postgresql.org/docs/current/auth-username-maps.html>`_ の PostgreSQL ドキュメントを参照してください。

オペレーターが書き込んだ pg_ident.conf ファイルは、次の 2 つのセクションで構成されています。

1. 固定ルール
2. ユーザー定義のルール

現在、オペレーターによって自動的に生成される唯一の固定ルールは次のとおりです。

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

インスタンスマネージャーは、PostgreSQL インスタンスを実行しているユーザーを検出し、データベースの postgres ユーザーにマップするルールを自動的に追加します。

postgres ユーザーがコンテナ内で適切に構成されていない場合、インスタンスマネージャーはローカルユーザーの接続を許可し、次のような警告メッセージをログに記録します。

```
Unable to identify the current user. Falling back to insecure mapping.
```

結果の pg_ident.conf は次のようになります。

```
local <postgres system user> postgres

<user defined lines>
```

クラスターマニフェスト内で、 pg_ident 行が.spec.postgresql.pg_ident のリスト項目として追加されます。例

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

3.1.4 構成の変更

Cluster リソースの postgresql セクションを編集することにより、構成変更を適用できます。

変更後、クラスターインスタンスはすぐに構成をリロードして変更を適用します。変更に再起動が必要なパラメーターが含まれる場合、オペレーターはローリングアップグレードを実行します。

3.1.5 ALTER SYSTEM の有効化

CloudNativePG™クラスターの EDB Postgres ® AI は、PostgreSQL クラスターの構成を変更するための排他的な方法として、クラスターマニフェストを使用することを強くお勧めします。このアプローチは、高可用性クラスター全体の一貫性を保証し、インフラストラクチャ・アズ・コードのベストプラクティスと連携しています。

CloudNativePG™クラスターの EDB Postgres ® AI では、デフォルト構成により、新しい Postgres クラスターで ALTER SYSTEM の使用が無効になっています。この決定は、このコマンドに関連する潜在的なリスクの認識に基づいています。ALTER SYSTEM の使用を有効にするには、.spec.postgresql.enableAlterSystem を true に明示的に設定します。

> **⚠ 警告**
>
> *ALTER SYSTEM* を利用するときは注意してください。このコマンドは、接続されたインスタンスで直接動作し、レプリケーションは行いません。 CloudNativePG™クラスターの EDB Postgres ® AI は、特定の固定パラメーターを担当し、他を完全に制御し、注意深い検討の必要性を強調します。

PostgreSQL 17 以降、.spec.postgresql.enableAlterSystem 設定は allow_alter_system を直接制御します

— CloudNativePG™クラスターの EDB Postgres ® AI から PostgreSQL に直接コントリビュートされた機能。

PostgreSQL 17 より前では、 .spec.postgresql.enableAlterSystem が false に設定されている場合、postgresql.auto.conf ファイルは読み取り専用になります。したがって、ALTER SYSTEM コマンドを実行しようとするとエラーになります。エラーメッセージは次のようになります。

```
ERROR:  could not open file "postgresql.auto.conf": Permission denied
```

3.1.6 動的共有メモリ設定

PostgreSQL は、dynamic_shared_memory_type を介して動的共有メモリ管理のいくつかの実装をサポートしています

構成オプション。 CloudNativePG™クラスターの EDB Postgres ® AI では、次の 2 つの値のいずれかに制限することをお勧めします。

- posix shm_open デフォルト設定を使用して割り当てられた POSIX 共有メモリに依存します
- sysv shmget を介して割り当てられた System V 共有メモリに基づいています

PostgreSQL では、この設定はパラレルクエリーでのメモリ割り当てにとって特に重要です。詳細は、*thread from the `pgsql-general* mailing list <https://www.postgresql.org/message-id/CA%2BhUKGJOj7qzDLxeFPVvto8YEWop6FSQoTYPO9Z6Ee%3Di-nPS_Q%40mail.gmail.com>`_ を参照してください。

POSIX 共有メモリ

オペレーターが/dev/shm の下に shm と呼ばれる**メモリバインドの** *``EmptyDir``* **ボリューム**を自動的にマウントすることを考慮すると、ほとんどの場合、 posix のデフォルト設定で十分です。次の方法で、実行中の Postgres コンテナ内のこのようなボリュームのサイズを確認できます。

```
mount | grep shm
```

次の出力のようなものを取得するはずです。

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

shm ボリュームの最大サイズを設定する場合は、Cluster リソースの.spec.ephemeralVolumesSizeLimit.shm フィールドを設定することにより設定できます。例

```
spec:
  ephemeralVolumesSizeLimit:
    shm: 1Gi
```

System V 共有メモリ

Kubernetes クラスターの SHMMAX および SHMALL パラメーターの十分な値がある場合、次のことを設定することもできます。

```
dynamic_shared_memory_type: "sysv"
```

次のコマンドを実行して、PostgreSQL コンテナ内から SHMMAX /SHMALL を確認できます。

```
ipcs -lm
```

例

```
- ----- Shared Memory Limits --------
max number of segments = 4096
max seg size (kbytes) = 18014398509465599
max total shared memory (kbytes) = 18014398509481980
min seg size (bytes) = 1
```

ご覧のとおり、max total shared memory の数が非常に多いため、dynamic_shared_memory_type を sysv に設定することをお勧めします。

別の方法は次を実行することです。

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

3.1.7 固定パラメーター

一部の PostgreSQL 構成パラメーターは、オペレーターが排他的に管理する必要があります。オペレーターは、ユーザーが Webhook を使用してそれらを設定できないようにします。

ユーザーは postgresql セクションで次の構成パラメーターを設定することはできません。

- 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
- 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
- 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
- unix_socket_directories
- unix_socket_group
- unix_socket_permissions

3.2 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、構成パラダイム上の規則に従います。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。

重要

構成の `ConfigMap` /*Secret* への変更は、オペレーターによって自動的に検出されません。そのため、リロードする必要があります以下を参照してください。さらに、変更は、構成がリロードされた後に作成されたリソースにのみ適用されます。

重要

オペレーターは、最初に ConfigMap 値を処理し、次に Secret をこの順序で処理します。その結果、パラメーターが両方の場所で定義されている場合、シークレットにあるものが使用されます。

3.2.1 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
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。
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:`archive_timeout <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

次のページに続く

表 1 – 前のページからの続き

Name	Description
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。
MONITORING_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。
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>`__ を定義します。デフォルトは 0 システムのデフォルトです。
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、 *kubectl* プラグインのサポート 、 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。

`INHERITED_ANNOTATIONS` および `INHERITED_LABELS` の値は、パスのようなワイルドカードをサポートしています。たとえば、値 `example.com/*` は、値 `example.com/one` および `example.com/two` の両方と一致します。

`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。

3.2.2 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
```

3.2.3 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
  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
```

3.2.4 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。

3.2.5 pprof HTTP Server

オペレーターは、`localhost:6060` の次のエンドポイントで PPROF HTTP サーバーを公開できます。

- `/debug/pprof/` . 使用可能なプロファイルをリストする HTML ページで「/debug/pprof/」の要求に応答します
- `/debug/pprof/cmdline` . 実行中のプログラムのコマンドラインに、NULL バイトで区切られた引数を使用して応答します。
- `/debug/pprof/profile` . pprof 形式の cpu プロファイルで応答します。プロファイリングは、GET パラメーターで指定された期間、または指定しない場合は 30 秒間持続します。
- `/debug/pprof/symbol` . 要求にリストされているプログラムカウンターを検索し、プログラムカウンターをファンクション名にマッピングするテーブルで応答します。
- `/debug/pprof/trace` . バイナリ形式の実行トレースで応答します。トレースは、秒 GET パラメーターで指定された期間、または指定しない場合は 1 秒間持続します。

オペレーターを有効にするには、オペレーター展開を編集し、フラグ`--pprof-server=true` を追加する必要があります。

これは、次のコマンドを実行して行うことができます。

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

次に、この例のように、編集ページでコンテナ引数を下にスクロールし、`--pprof-server=true` を追加します。

```
containers:
- args:
  - controller
  - --enable-leader-election
  - --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
```

変更を保存します。ここでの展開ではロールアウトが実行され、新しいポッドでは PPROF サーバーが有効になります。

ポッドが実行されたら、次の方法でコンテナ内で実行できます。

```
kubectl exec -ti -n postgresql-operator-system <pod name> -- bash
```

内部に入ったら、次を実行します。

```
curl localhost:6060/debug/pprof/
```

3.3 Instance pod configuration

3.3.1 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 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:
```

(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
  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。

3.3.2 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。

3.3.3 環境変数

環境変数を使用して、一部のシステム動作をカスタマイズできます。一例は、カスタム LDAP 構成ファイルをポイントできる LDAPCONF 変数です。別の例は、PostgreSQL コンテナが使用するタイムゾーンを表す TZ 環境変数です。

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。

```
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。

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

  envFrom:
```

(次のページに続く)

(前のページからの続き)

```
  - 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。

3.4 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。

重要

これらの例は、デモンストレーションと実験を目的としています。 クイックスタート で説明しているように、Minikube または Kind を使用して個人の Kubernetes クラスターで実行できます。

参考

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

3.4.1 基本

基本クラスター *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 拡張 PGE 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。

カスタマイズされたストレージクラスを使用するクラスター *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的なクラスター。

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的なクラスター。

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。

``pg_hba`` 構成をカスタマイズ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。

3.4.2 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こと。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：**前提条件** *cluster-storage-class-with-backup.yaml*

正常に稼働 *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 用に構成されたバックアップを含む単純なクラスター： **前提条件**構成は、minio が実行され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：**前提条件**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。

3.4.3 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：前提条件 *cluster-storage-class-with-backup.yaml*

適用され、正常、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を介したレプリカクラスター： 前提条件 *cluster-example-with-volume-snapshot.yaml*

適用され、正常な、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：前提条件 *cluster-example.yaml*

正常に稼働: *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。

3.4.4 PostGIS

PostGIS の例 *postgis-example.yaml* PostGIS クラスターの例。詳細は、*PostGIS コンテナイメージ* を参照してください。

3.4.5 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。

3.4.6 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。

3.4.7 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： 前提条件 構成は、minio が実行され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のテーブルスペースを含む復元されたクラスター：前提条件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* を参照してください。

3.4.8 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*

3.4.9 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、**アプリユーザー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のある宛先クラスター：**前提条件**上記のように定義されたパブリケーションのあるソースクラスター。*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するプレーンパブリケーション： **前提条件** 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使用したパブリケーション： **前提条件** 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 前提条件既存のクラスター `cluster-example` がソースとして設定され、パブリケーション `pub-all` を使用します。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。

3.4.10 Postgres データベースの宣言的管理

プレーンなデータベース：**前提条件**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： **前提条件** 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*

第4章　Using

4.1 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。

> **重要**
>
> 既存のクラスターからのブートストラップにより、**レプリカクラスター**の作成が可能になります。これは、継続的リカバリーのままで、ソースクラスターとの同期を維持し、読み取り専用接続を受け入れる独立した PostgreSQL クラスターです。詳細については、*Replica Cluster section* 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。

4.1.1 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 ファイルまたは特定の**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。

EDB Postgres ® AI for CloudNativePG™クラスターオペレーターが `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 **バックアップ リカバリー** を参照してください。

> **参考**
>
> "API reference for the `bootstrap section <API Reference>` を参照してください。

詳細については、

4.1.2 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**ソース** 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 externalClusters	pg_basebackup	recovery
Only streaming	✓	
Only object store		✓

次のページに続く

表 1 – 前のページからの続き

Content of external-Clusters	pg_basebackup	recovery
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。

参考

"API reference for the `externalClusters section <API Reference>` を参照してください。

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。

4.1.3 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
```

(次のページに続く)

(前のページからの続き)

```
    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 **非特権** 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。

構成パラダイムに関する条約のおかげで、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スーパーユーザー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 のアプリケーションユーザー。

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
  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。

> **重要**
>
> 追加のユーザーを作成する必要がある場合は、 *Declarative database role management* を参照してください。

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、 **デフォルト**

の *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継続的な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。

builtinLocale `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。

dataChecksums `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` 。

fence `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。

icuLocale `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。

icuRules `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 Locale Support 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。

locale `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。

localeCollate `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。

localeCType `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。

localeProvider `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 EDB Postgres ® AI for CloudNativePG™クラスターが実装する 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。

```
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**スーパーユーザー** `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、 EDB Postgres ® AI for CloudNativePG™クラスターは、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。

重要

postInitSQLRefs 、*postInitTemplateSQLRefs* 、および `postInitApplicationSQLRefs` で指定された ConfigMaps または Secrets 内のエントリの存在を確認します。それ以外の場合、ブートストラップは失敗します。これらの SQL ファイルにエラーがあると、ブートストラップフェーズが正常に完了できません。

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。

```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
      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。

```yaml
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>

  bootstrap:
    initdb:
      database: app
      owner: app
      redwood: false
  storage:
    size: 1Gi
```

4.1.4 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バージョン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、既存の物理的な**ベースバックアップ** `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。

重要

デフォルトでは、 *recovery* メソッドは`externalClusters` セクションのクラスターの `name` を厳密に使用して、通常サーバーの名前に予約されているオブジェクトストア内のバックアップデータのメインフォルダーを見つけます。 Backup プラグインは、別のプラグインを指定する方法を提供します。例、Barman Cloud プラグインは、 serverName

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 EDB Postgres ® AI for CloudNativePG™クラスターオペレーター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 バックアップ 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、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**ソース** の正確な物理コピーとして、新しいクラスター **ターゲット**を作成できます。有効な **ストリーミングレプリケーション**接続を使用して。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™クラスタークラスターの 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

> **重要**
>
> すべての 要件 が満たされ、操作が完全にテストされていることが完全に確認できない限り、Kubernetes の外部の既存の PostgreSQL クラスターを EDB Postgres ® AI for CloudNativePG™ クラスターに移行するために、物理的レプリケーションに基づくこの方法の使用は避けてください。 EDB Postgres ® AI for CloudNativePG™ Cluster コミュニティは、このようなユースケースについてこのアプローチを承認しておらず、代わりに論理インポートを使用することをお勧めします。 CloudNativePG™ Cluster の EDB Postgres ® AI とシームレスに動作する方法で、物理レプリケーションのすべての要件が満たされることは、非常にまれです。

> **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*スナップショット*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。

重要

ターゲットインスタンスとソースインスタンス間のネットワーク接続の構成は、特定のコンテキストと環境に大きく依存するため、 EDB Postgres ® AI for CloudNativePG™ Cluster ドキュメントの範囲外です。

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

参考

詳細については、 Planning 、 pg_basebackup を参照してください。

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。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: target-db
spec:
  instances: 3
  imageName: quay.io/enterprisedb/postgresql:17.5

  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 17.5 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 17.5 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: quay.io/enterprisedb/postgresql:17.5

  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。

> **重要**
>
> *Cluster* がリカバリモードである間、カタログを含むデータベースへの変更は許可されません。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。復旧フェーズでは、ロールはソースクラスターで定義されたとおりに残ります。

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。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
[...]
spec:
  bootstrap:
    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。

> **重要**
>
> 移行を試行する前に、プロシージャーとアプリケーションの両方をテストする必要があります。特に、必要なだけ移行手順を実行して、運用中のアプリケーションのダウンタイムを体系的に測定することが重要です。

4.2 Importing Postgres databases

このセクションでは、CloudNativePG™ Cluster クラスター用の新しい 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 13.x からバージョン 17.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。

> **重要**
>
> このため、バックアップの開始後にソースデータベースに行われた変更は宛先クラスタにないため、*Cluster* リソースの最終インポートの前にソースの書き込み操作を停止することをお勧めします。したがって、この機能が参照される理由「オフラインインポート」または「オフラインメジャーアップグレード」として。

4.2.1 その仕組み

概念的には、インポートには、*BootstrapInitDB* 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、EDB Postgres ® AI for CloudNativePG™クラスタープロジェクトの推奨に従って、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、

4.2.2 `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` を使用した選択したデータベース `initdb.import.databases` 内のエクスポート
- `initdb.owner` ユーザーが所有する `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
- 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。

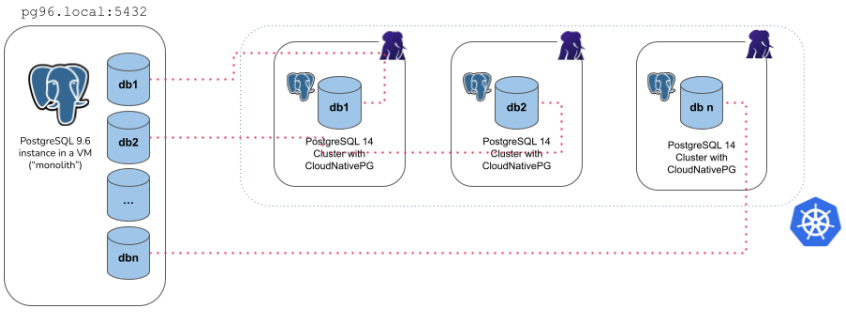


図 1: Example of microservice import type

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。

```
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す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

> **ヒント**
>
> マイクロサービスアプローチは、 EDB Postgres ® AI for CloudNativePG™クラスター規約と宛先クラスターのデフォルトに従います。宛先クラスターに `initdb.database` または `initdb.owner` を設定しない場合、両方のパラメーターはデフォルトの `app` になります。

4.2.3 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
- pg_dump -Fd を使用して、選択したデータベース initdb.import.databases 内を一度に 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 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
```

(次のページに続く)

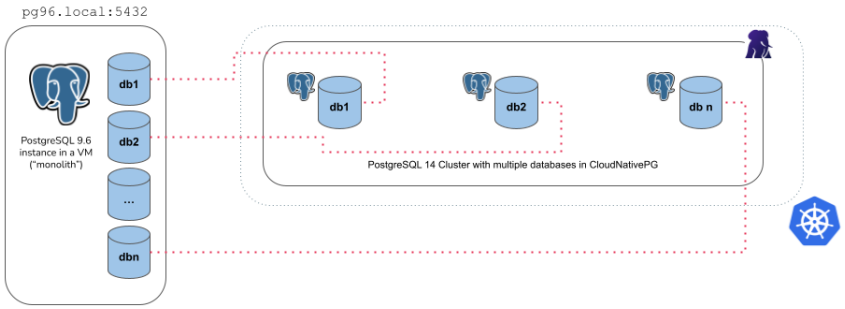


図 2: Example of monolith import type

(前のページからの続き)

```
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

> **ヒント**
>
> データベースとその所有者は、ソースクラスターに存在するものとまったく同じように保持されます。インポート中に `app` データベースもユーザーも作成されません。 *bootstrap.initdb* スタンザが、インポートされるデータベースまたはユーザーと一致しないカスタム `database` および `owner` 値を指定している場合、インスタンスマネージャーは、インポートされたデータベースと所有者を変更しないまま、これらの指定された名前で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。

4.2.4 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
```

(次のページに続く)

(前のページからの続き)

```
metadata:
  name: cluster-example
spec:
  instances: 1

  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

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

4.2.5 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。

重要

WAL アーカイブ要求された場合、およびデータベースのインポート処理が完了した後に WAL レベルが受け入れられます。同様に、レプリカはブートストラップフェーズの後に、実際のクラスターリソースが開始されるときにクローン作成されます。

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。

4.2.6 pg_dump および pg_restore の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。たとえば、次の例に示すように、パラレルジョブを有効にしてデータのインポート/エクスポートプロセスを高速化できます。

```
#  <snip>

bootstrap:
  initdb:
    import:
      type: microservice
      databases:
        - app
      source:
        externalCluster: cluster-example
      pgDumpExtraOptions:
        - "--jobs=2"
      pgRestoreExtraOptions:
        - "--jobs=2"

#  <snip>
```

警告

pgDumpExtraOptions および `pgRestoreExtraOptions` フィールドは、自己の責任で注意して使用してください。これらのオプションはオペレーターによって検証または検証されておらず、一部の構成は意図

された機能または動作と競合する場合があります。運用に適用する前に、安全で制御された環境で完全にテストしてください。

4.2.7 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 *Logical Replication* セクションを参照してください。

4.3 Security

このセクションには、CloudNativePG™クラスターの EDB Postgres ® AI のセキュリティに関する情報が含まれており、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。

参考

The 4C’ s Security Model in Kubernetes を参照してください。

ブログ記事 CloudNativePG™クラスターの EDB Postgres AI のセキュリティで EDB が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します。

4.3.1 コード

EDB Postgres ® AI for CloudNativePG™ Cluster ’ ソースコードは、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。

4.3.2 コンテナ

CloudNativePG™クラスターの EDB Postgres ® AI のすべてのコンテナイメージは、すべてのコミットに続いて、CI / CD パイプラインを介して自動的にビルドされます。これらのイメージには、オペレーターのイメージだけでなく、特にサポートされているすべての PostgreSQL バージョン EDB Postgres Extended および EDB Postgres Advanced を含むオペランドのイメージが含まれます。CI / CD プロセス中に、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。
- ** Snyk :** コンテナ内のセキュリティ問題を検出し、GitHub インターフェイスを介して結果を報告します。

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日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で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ているコンテナイメージに**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。

4.3.3 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。

ロールベースのアクセス制御 RBAC

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* 、*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、最も重要な理由を提供します。

`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、信頼できるステータスを継続的に提供できるようにするサービスアカウントを作成する必要があります。`Cluster` の。

`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イルオーバー/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割り当てることにより、適切なプライマリ PostgreSQL インスタンス)。

`validatingwebhookconfigurations` および `mutatingwebhookconfigurations` オペレーターは、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な両方の Webhook 構成に、自己署名 Webhook CA を注入します。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。

!!!Info OLM では、オペレーターを独自の名前空間に展開し、CloudNativePG™クラスタークラスターの 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。

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。 .. ::

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。

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 で定義されます

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 を必要とします。

clusters/status インスタンスマネージャーは、 update および patch に、自分自身の Cluster リソースのみのステータスを必要とします

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、update および patch に、名前空間内の Backup リソースのステータスを必要とします

ポッドセキュリティポリシー

重要

Kubernetes v1.21 から、*PodSecurityPolicy* の使用は非推奨になり、Kubernetes v1.25 の時点で、完全に削除されました。この非推奨にもかかわらず、オペレーターが現在 Kubernetes の古いサポートされていないバージョンでテストを受けていることを確認しています。したがって、このセクションはこれらの特定のシナリオのために保持されています。

Pod Security Policy

これは、クラスターで実行するためにポッドが満たす必要があるセキュリティルールと仕様を定義する Kubernetes 方法です。InfoSec の理由から、すべての Kubernetes プラットフォームはそれらを実装する必要があります。

CloudNativePG™クラスターの EDB Postgres ® AI は、コンテナの実行に **特権**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、ボリュームへのアクセスには、**特権** モードまたは `root` 特権も必要ありません。Kubernetes プラットフォームおよび/または管理者は、適切な権限を適切に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、必要なセキュリティコンテキストを明示的に設定します。

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。

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。

!!!「クラスターアノテーションの例」も参照してください

```
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
```

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	No
operand	5432	PostgreSQL instance	*postgresql*	Optional	Yes

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。[“Client TLS/SSL Connections”](ssl_connections.md#“Client TLS/SSL Connections”) ページ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。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。

4.4 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。

4.4.1 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。

> **重要**
>
> *.spec.probes.startup.maximumLag* オプションは、ポッドの起動フェーズ中にのみ検証および適用されます。つまり、レプリカが起動しているときにのみ適用されます。

> **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
```

4.4.2 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
```

(次のページに続く)

(前のページからの続き)

```
    timeoutSeconds: 3
    failureThreshold: 10
```

プライマリ分離

CloudNativePG™クラスター 1.27 の EDB Postgres 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
```

重要

デフォルトの liveness プローブ設定 `livenessProbeTimeout` から自動的に派生は、アグレッシブな 30 秒になる場合があることに注意してください。そのため、環境に合わせて liveness プローブ構成を明示的に設定することをお勧めします。

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"
```

4.4.3 Readiness Probe

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、readness 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
      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した。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。

重要

startup プローブとは異なり、 *.spec.probes.readiness.maximumLag* オプションは継続的に監視されます。この設定が適切に調整されていない場合、遅延レプリカの準備ができなくなる場合があります。

!!!警告

```
Incorrect configuration of the `maximumLag` option can lead to repeated
readiness probe failures, causing serious consequences, such as:


- Exclusion of the replica from key operator features, such as promotion
  during failover or participation in synchronous replication quorum.

- Disruptions in read/read-only services.

- In longer failover times scenarios, replicas might be declared unready,
  leading to a cluster stall requiring manual intervention.
```

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。

```yaml
#  <snip>

probes:
  readiness:
    type: streaming
    maximumLag: 64Mi
    failureThreshold: 30
    periodSeconds: 10
```

4.4.4 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 **スマート**シャットダウンを要求し、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。

> **重要**
>
> データベース リリース候補の目的 に影響を与える Postgres クラスターでのデータの損失を回避するには、プライマリインスタンスが実行されているポッドを削除しないでください。この場合、最初に別のインスタンスへのスイッチオーバーを実行します。

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。実際、オペレーターは、新しいプライマリですべてのデータが利用できるように保証に、選択した新しいプライマリを昇格させる前に、元のプライマリに **fast** シャットダウンを発行する必要があります。

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 バックアップ頻度と *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。

4.4.5 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 フェイルオーバーに対する復元力 を参照してください。

4.4.6 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 にも当てはまります。 モニタリングとメトリック

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。

重要

実稼働システムでは、データベースを継続的に監視することが重要です。ディスクストレージが使い果たされると、データベースサーバーがシャットダウンする場合があります。

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。

4.5 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。

> **参考**
>
> *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 EDB Postgres ® AI for CloudNativePG™ Cluster クラスターのインスタンス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

4.5.1 ポッドアフィニティとアンチアフィニティ

Kubernetes は、**アフィニティ**および**アンチアフィニティ**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**アフィニティ**、または特定のノードで回避するかどうか**アンチアフィニティ**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: quay.io/enterprisedb/postgresql:17.5

  affinity:
    enablePodAntiAffinity: true # Default value
    topologyKey: kubernetes.io/hostname # Default value
    podAntiAffinityType: preferred # Default value
```

(次のページに続く)

(前のページからの続き)

```
  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。

```
affinity:
  podAntiAffinity:
    preferredDuringSchedulingIgnoredDuringExecution:
      - podAffinityTerm:
          labelSelector:
            matchExpressions:
              - key: k8s.enterprisedb.io/cluster
                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**優先** 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に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、Kubernetes クラスターでの自動水平スケーリングに Cluster Autoscaler を使用する場合に特に関連します。

> **参考**
>
> 詳細については、 *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。

```yaml
additionalPodAntiAffinity:
  requiredDuringSchedulingIgnoredDuringExecution:
  - labelSelector:
      matchExpressions:
      - key: postgresql
        operator: Exists
        values: []
    topologyKey: "kubernetes.io/hostname"
```

4.5.2 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ように PostgreSQL クラスターを要求できます。

4.5.3 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。

> **参考**
>
> taint と toleration の詳細については、 *Kubernetes documentation* を参照してください。

4.5.4 PostgreSQL ワークロードの分離

> **重要**
>
> 続行する前に、ドキュメントの アーキテクチャ セクションを必ず読んでください。

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:
```

(次のページに続く)

(前のページからの続き)

```
    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。

4.6 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、2 つのノブを使用してマニフェストの `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。

- `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"
  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**
- **バースト可能**
- **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
```

(次のページに続く)

(前のページからの続き)

```
    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。

参考

リソース管理の詳細については、 Managing Compute Resources for Containers を参照してください。

Kubernetes ドキュメントのページ。

4.7 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 の動作に従っているため、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。EDB Postgres ® AI for CloudNativePG™クラスター固有の詳細については、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。

重要

本番で CloudNativePG™クラスターの EDB Postgres ® AI を実行している場合、 professional support を探すことを強くお勧めします。

4.7.1 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。

4.7.2 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。

> **重要**
>
> professional support なしで手動操作を実行しないでください。

リコンシリエーションの無効化

PostgreSQL クラスターのリコンシリエーションループを一時的に無効にするには、k8s.enterprisedb.io/reconciliationLoop アノテーションを使用します。

```yaml
metadata:
  name: cluster-example-no-reconcile
  annotations:
    k8s.enterprisedb.io/reconciliationLoop: "disabled"
spec:
  # ...
```

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、オペレーターはフェイルオーバーを含む自己修復アクションを実行できません。

4.8 Rolling Updates

オペレーターを使用すると、アプリケーションが実行されているときにクラスターで使用される PostgreSQL バージョンを変更できます。

重要

PostgreSQL マイナーリリースのアップグレードのみがサポートされています。

ローリングアップグレードは、次の場合に開始されます。

- ユーザーはクラスター仕様の `imageName` 属性を変更します。
- *Image Catalog* は、クラスターが使用するメジャーの新しいイメージで更新されます。
- PostgreSQL 構成を変更するには、再起動を適用する必要があります。
- `Cluster .spec.resources` 値の変更
- AKS の永続ボリューム要求のサイズの変更
- オペレーターが更新された後、ポッドが最新のインスタンスマネージャーを実行していることを確認します *in-place updates are enabled* の場合

オペレーターは、一度に 1 つのポッドずつ、すべてのレプリカのアップグレードを開始します。シリアルが最も高いものから開始します。

プライマリは、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。

4.8.1 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、選択された `primaryUpdateMethod` オペレーションがプライマリで開始されます。これ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。

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 **バックアップ頻度と** *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。

4.8.2 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、 :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>` を参照してください。

4.9 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。

重要

このセクションは、同じ Kubernetes クラスターで管理される同じ `Cluster` リソース内のレプリケーションに関するものです。異なる Kubernetes クラスター間でも、別の Postgres *Cluster* リソースを使用して複製する方法については、 レプリカクラスター セクションを参照してください。

4.9.1 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。

4.9.2 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 9.02010 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。 *Logical Replication* 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。

4.9.3 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
```

> **参考**
>
> EDB Postgres ® AI for CloudNativePG™ クラスターが証明書を管理する方法の詳細については、 *CertificatesConfiguration* を参照してください。

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。

> **参考**
>
> CloudNativePG™クラスターの EDB Postgres ® AI が高可用性レプリカのレプリケーションスロットを自動的に管理する方法の詳細については、 **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。

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。

4.9.4 同期レプリケーション

CloudNativePG™クラスターの EDB Postgres 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。

> **重要**
>
> *failover quorum* feature* 実験的は、同期レプリケーションとともに使用して、フェイ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: 1G
```

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: 1G

  postgresql:
    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。

重要

現在、この方法は、以下で説明する `maxStandbyNamesFromCluster` 、*standbyNamesPre* 、および `standbyNamesPost` オプションを使用して、現在のクラスターを超えて同期レプリケーションを拡張する場合に最も役立ちます。

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。

> **重要**
>
> *preferred* は、*standbyNamesPre* および`standbyNamesPost` が設定されていない場合にのみ使用できます。

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。

> **重要**
>
> レプリカにとって*ready/available*が何を意味するかを明確に理解し、それに応じて期待を設定します。デフォルトでは、レプリカは少なくとも 1 回ソースに正常に接続すると、準備ができているとみなされます。ただし、 CloudNativePG™クラスターの EDB Postgres ® AI を使用すると、最大遅延に基づいてレプリカの起動および準備状況プローブを構成できます。詳細については、 *Postgres instance manager* を参照してください。

この設定は、データの安全性と可用性のバランスをとり、アプリケーションは一時的にスタンバイが使用不可になっている間も書き込みを続行できます。したがって、**自己修復モード** 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。

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

(次のページに続く)

(前のページからの続き)

```
  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。

4.9.5 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 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。

重要

非推奨なメソッドと新しいメソッドは相互排他的です。

CloudNativePG™クラスターの EDB Postgres 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。

重要

デフォルトでは、同期レプリケーションは、使用可能なすべてのレプリカから区別なく選択します。次のセクションで説明するように`syncReplicaElectionConstraint` オプションを介してノードラベルを操作することにより、同期レプリカをスケジュールできるノードを制限できます。

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 `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、**メソッド ``*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リスト**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。

重要

オペレーターが同期レプリケーション設定の適用よりも自己修復を選択する場合でも、3 つ以上のインスタンスがあるクラスター、より一般的には`maxSyncReplicas < (instances - 1)` の場合にのみ同期レプリケーションを計画することをお勧めします。

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。

参考

一般的なポッドアフィニティおよびアンチアフィニティルールの詳細については、 *Scheduling* を確認してください。

> **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。

> **重要**
>
> 自己修復の適用は、同期レプリカ選択の追加制約を定義しているときに引き続き適用されます 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 を参照してください。

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。

4.9.6 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、機能は有効です `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:
```

(次のページに続く)

(前のページからの続き)

```
  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 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。

```
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。

```
# ...
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。

> **参考**
>
> CloudNativePG™クラスター展開の EDB Postgres ® AI をモニタする方法の詳細については、 **モニタリングとメトリック** を参照してください。

4.10 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。

重要

バージョン 1.26 から、ネイティブバックアップおよびリカバリー機能は、コアオペレーターの**段階的にフェーズアウト**され、公式 CNP-I プラグインに移動されます。この移行は、**WAL アーカイブ**の管理を標準化する拡張可能なインターフェイス**CNP-I**によって有効になる**バックアップに依存しないアーキテクチャ**への EDB Postgres ® AI for CloudNativePG™ Cluster のシフトと一致しています。 、**物理ベースバックアップ**、および対応する**リカバリプロセス**。

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** を公式にサポートしています

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

(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。

4.10.1 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

CNP-I は、WAL アーカイブアーカイブと復元操作の両方と、ベースバックアップと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。

> **重要**
>
> 常に本番で WAL アーカイブをセットアップすることをお勧めします。通常ステージング環境と開発環境が関係する既知のユースケースがあり、上記の利点はいずれも必要とされず、WAL アーカイブは必要ありません。この場合の RPO は、24 時間毎日のバックアップまたは無限バックアップなしなどの任意の値です。

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。

4.10.2 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

> **重要**
>
> CNP-I は、サードパーティが独自のバックアッププラグインをビルドおよび統合できるように設計されています。時間の経過とともに、サポートされているバックアップソリューションのエコシステムが成長すると予想されます。

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	OK
Hot backup	OK	OK
Incremental copy	NO	☒^2^
Differential copy	NO	☒^2^
Backup from a standby	OK	OK
Snapshot recovery	NO^3^	OK
Retention policies	OK	NO
Point-in-Time Recovery (PITR)	OK	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。

4.10.3 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** タイミングを定義します。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。

バックアップ頻度と RTO

> **ヒント**
>
> バックアップの頻度は、 **目標復旧時間** バックアップ頻度と *RTO* に直接影響します。

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

4.10.4 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:
  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
```

- **

重要

オンデマンドバックアップには、PostgreSQL スーパーユーザーまたはアプリケーションユーザーの Kubernetes シークレットは含まれません。これらのシークレットが、より広範な Kubernetes クラスター

バックアップ戦略に含まれていることを確認する必要があります。

4.10.5 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` – CNP-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 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。

4.10.6 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、 ** EDB Postgres ® AI for CloudNativePG™クラスターは、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取得をサポートしています。

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。

```yaml
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。

4.10.7 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 **CNP-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 。

> **重要**
>
> ユーザーは、使用しているバックアッププラグインが提供する保持メカニズムを信頼することをお勧めします。これにより、使用中のバックアップ方法の柔軟性と一貫性が保証されます。

4.11 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。

> **重要**
>
> PITR を実行するには、有効な WAL アーカイブが必要です。

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、以下を介したリカバリー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 CNP-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。

> **重要**
>
> 古いドキュメントについては、 *Appendix B – Recovery from an Object Store* を参照してください。

4.11.1 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。

> **重要**
>
> オブジェクトストアには、CloudNativePG™クラスター *Cluster* 用の EDB Postgres ® AI によって生成されたバックアップデータが含まれている必要があります。**非推奨のネイティブ Barman Cloud 統合**** **または** Barman Cloud Plugin**のいずれかを使用して。

> **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。

```
apiVersion: barmancloud.k8s.enterprisedb.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。

```
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
```

4.11.2 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` または `PersistentVolumeClaim` PVC 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。

4.11.3 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:
```

(次のページに続く)

(前のページからの続き)

```
instances: 3

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。

4.11.4 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。

> **重要**
>
> *barmanObjectStore.wal.maxParallel* オプションを使用して、リカバリオブジェクトストアからトランザクションログを同時にダウンロードすることにより、アーカイブからの WAL 取得を高速化することを検討します。

4.11.5 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。

重要

PITR では、リカバリーターゲット で説明したオプションを使用してリカバリターゲットを指定する必要があります。

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。

> **重要**
>
> このようなバックアップが存在し、アクセス可能であることを確認する必要があります。

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: "2023-07-06T08:00:39"
  externalClusters:
    - name: origin
      plugin:
        name: barman-cloud.cloudnative-pg.io
        parameters:
```

(次のページに続く)

(前のページからの続き)

```
        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 *Tablespaces* 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。

targetTime RFC 3339 形式で表現される、リカバリーが進行するタイムスタンプ。正確な停止ポイントは、`exclusive` オプションの影響も受けます。

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。

targetXID リカバリーが続行され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。

targetName リカバリが続行する名前付けのリストアポイント `pg_create_restore_point()` で作成されます。

targetLSN リカバリを続行す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。

重要

targetTime または `targetLSN` のいずれかを指定すると、オペレーターは最も近いバックアップを取得できます。ただし、これは残りのターゲット `targetName` 、*targetXID* 、および `targetImmediate` では不可能です。このような場合、*backupID* の指定は必須です。

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:
```

(次のページに続く)

(前のページからの続き)

```
    source: origin
    recoveryTarget:
      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
```

4.11.6 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、

重要

Cluster がリカバリモードである間、カタログを含むデータベースへの変更は許可されません。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。このフェーズでは、ユーザーはソースクラスターで定義されたままになります。

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。

```
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. 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。

4.11.7 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**ブートストラップ** 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。

> **重要**
>
> PostgreSQL PITR の方法に慣れていない場合

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。

> **重要**
>
> 新しい PVC のベースバックアップコピーの期間は、バックアップのサイズ、およびネットワークとストレージの両方の速度によって異なります。

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。

4.11.8 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。

重要

復旧したクラスターで `k8s.enterprisedb.io/skipEmptyWalArchiveCheck` アノテーションを `enabled` に設定することにより、この安全性チェックをバイパスできます。ただし、PostgreSQL の復旧プロセスに精通していない限り、これは強くお勧めしません。チェックを誤ってスキップすると、重大なデータの損失が発生する可能性があります。専門的なシナリオでのみ注意して使用してください。

4.12 Appendix A - Backup on volume snapshots

重要

サポートされているすべてのサービスのリストについては、Kubernetes の公式ドキュメントを参照してください Container Storage Interface (CSI) drivers

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 ネイティブの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。

4.12.1 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。

4.12.2 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。

> **重要**
>
> ボリュームスナップショットがサポートされていることをサードパーティベンダーに確認するのはあなたの責任です。 CloudNativePG™クラスターの EDB Postgres ® AI は、この問題に関して Kubernetes API とのみ対話し、特定の CSI ドライバーごとにストレージレベルでの問題をサポートすることはできません。

4.12.3 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。

4.12.4 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

重要

デフォルト値は、ほとんどの実稼働環境に適しています。ホットバックアップは一貫性があり、スナップショットリカバリーを実行するために使用できます。バックアップの開始から一時レプリケーションスロットを介して WAL の保持を保証します。ただし、そのためにはコールドバックアップに依存することをお勧めします。

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:
  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
```

4.12.5 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、

4.12.6 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...
```

4.12.7 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。

> **重要**
>
> 例のテストに興味がある場合は、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 Volume Snapshots を読んでください。

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:
```

(次のページに続く)

(前のページからの続き)

```
    storageClass: ebs-sc
    size: 10Gi
  walStorage:
    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。

4.13 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 `quay.io/enterprisedb/postgresql` を使用できます。

重要

Barman クラウドで導入された機能強化を利用するために、システムで最新バージョンのオペランドを実行していることを常に確認して、クラスターのセキュリティ面を向上させます。

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** 。

4.13.1 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。

4.13.2 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 。

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:
  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。

重要

デフォルトでは、CloudNativePG™クラスターの EDB Postgres ® AI は `archive_timeout` を `5min` に設定して、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 分ごとにクローズおよびアーカイブされ、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 Objective 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、オペレーターによって設定され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。

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。

4.13.3 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:
  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"
```

4.13.4 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 (ms)	Time	Restore (ms)	Time	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

4.13.5 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。

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
[...]
spec:
  backup:
    barmanObjectStore:
      [...]
      tags:
        backupRetentionPolicy: "expire"
      historyTags:
        backupRetentionPolicy: "keep"
```

4.13.6 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"
```

4.13.7 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:
      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。

重要

デフォルトでは、 *recovery* メソッドは、オブジェクトストア内のバックアップデータのメインフォルダーの名前として、 *externalClusters* セクションのクラスターの `name` を厳密に使用します。この名前は、通常、サーバーの名前用に予約されています。 *barmanObjectStore.serverName* プロパティを使用し

て、別のフォルダー名を指定できます。

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。

4.14 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出荷するプロセスです。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。

4.14.1 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。

4.14.2 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。

4.14.3 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。

重要

すべての新しい展開は、より柔軟で保守可能なアプローチを提供するプラグインベースのアーキテクチャを採用することを強くお勧めします。

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。

4.14.4 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 に設定して、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。

4.15 PostgreSQL Role Management

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。

クラスター仕様の managed スタンザにより、CloudNativePG™クラスターの EDB Postgres ® AI は、.spec.managed.roles で指定されたロールの完全なライフサイクル管理を提供するようになりました。

この機能により、既存のロールの宣言的管理と、データベースにまだ存在しない場合は新しいロールの作成が可能になります。ロールの作成は、データベースのブートストラップが完了した後*に発生します。

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
```

`.spec.managed.roles` のロール仕様は、PostgreSQL structure and naming conventions に従います。ロールごとに定義できる属性の完全なリストについては、*API Reference* を参照してください。

いくつかの点に注意してください。

1. `ensure` 属性は PostgreSQL の **ではありません**。これにより、宣言的なロール管理でロールを作成および削除できます。2 つの値は、`present` デフォルトと `absent` です。
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。

宣言的ロール管理により、PostgreSQL インスタンスが仕様に準拠していることが保証されます。ユーザーがデータベースでロール属性を直接変更した場合、EDB Postgres ® AI for CloudNativePG™ Cluster オペレーターは、次の調整サイクル中にこれらの変更を元に戻します。

4.15.1 パスワード管理

宣言的ロール管理機能には、ロールパスワードの調整が含まれます。パスワードは Kubernetes の世界と PostgreSQL では根本的に異なる方法で管理されており、その結果、いくつかの注意事項があります。

管理対象ロール構成は、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。Kubernetes で通常のように Base64 でエンコードされます。例

```
managed:
  roles:
  - name: dante
    ensure: present
    [··· snipped ···]
    passwordSecret:
      name: cluster-example-dante
```

これは、ユーザー名とパスワードを含む `cluster-example-dante` と呼ばれるシークレットの存在を前提としています。ユーザー名は、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
```

ロールに `passwordSecret` が指定されていない場合、インスタンスマネージャーはパスワードを使用してロールを作成/変更しようとしません。これは、`WITH PASSWORD` で指示されない限り、ALTER がパスワードを更新しない PostgreSQL 規約に従います。

ロールが最初にパスワードを使用して作成され、PostgreSQL でパスワードを NULL に設定したい場合、これは EDB Postgres ® AI for CloudNativePG™ Cluster のユーザー側で明示的にする必要があります。「仕様で提供されているパスワードなし」と「パスワードを NULL に設定する」を区別するには、フィールド `DisablePassword` を使用する必要があります。

データベースの `dante` ロールにパスワードを設定しないことにしたとします。このような場合、次を指定します。

```
managed:
  roles:
  - name: dante
    ensure: present
    [··· snipped ···]
    disablePassword: true
```

注特定のロールで `passwordSecret` と `disablePassword` の両方を設定することはエラーとみなされます。この構成は、検証 Webhook によって拒否されます。

パスワードの有効期限 VALID UNTIL

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。

警告

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。

パスワードがハッシュ化されました

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>
```

4.15.2 実現不可能なロール構成

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、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、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。どちらが続くか…

4.15.3 管理対象ロールのステータス

クラスターステータスには、以下に示すように、管理対象ロールのステータスのセクションが含まれます。

```
status:
  [···snipped···]
  managedRolesStatus:
    byStatus:
      not-managed:
      - app
      pending-reconciliation:
      - dante
      - petrarca
      reconciled:
      - ariosto
      reserved:
      - postgres
      - streaming_replica
    cannotReconcile:
      dante:
      - could not perform DELETE on role dante: owner of database inferno
      petrarca:
      - could not perform UPDATE_MEMBERSHIPS on role petrarca: role "poets" does not␣
↪exist
```

データベースおよび EDB Postgres ® AI for CloudNativePG™ Cluster 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
```

重要

下位互換性の観点から、宣言的ロール管理は、データベースには存在するが仕様に含まれていないロールを無視するように設計されています。これらのロールのライフサイクルは引き続き PostgreSQL 内で

管理され、CloudNativePG™ Cluster ユーザーの EDB Postgres ® AI は、自分の都合に合わせてこの機能を採用できます。

4.16 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。

重要

動的にプロビジョニングされるストレージに関しては、Kubernetes に独自の詳細があります。これらには、**ストレージクラス**、**永続ボリューム**、および*永続ボリューム要求 PVC*が含まれます。 VM および物理サーバー上のデータベースワークロードのストレージの観点から、長年構築してきた貴重な知識に加えて、これらの概念を所有する必要があります。

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™クラスター用に 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。

4.16.1 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。

> **重要**
>
> サポートされているすべてのサービスのリストについては、Kubernetes ドキュメントを参照してください container storage interface (CSI) drivers

スナップショット機能を提供します。

4.16.2 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

> **重要**
>
> データベースを本番環境に配置する前に、ストレージとデータベースのパフォーマンスの両方を測定する必要があります。これらの結果は、計画段階たとえば、キャパシティプランニングなどにおいてだけでなく、非常に価値があります。また、実稼働ライフサイクル、特にこの種のテストを実行する時間がない緊急事態においても貴重です。データベースは時間の経過とともに変更および進化し、データの分散も同様に、パフォーマンスに影響を与える可能性があります。シークエンシャルな読み取りまたは書き込みの理論的な最大スループットを知ることは、これらの状況において非常に役立ちます。これは、外部ワークロードの影響によって結果が変化しないシェアードナッシングコンテキストに特に当てはまります。

自分のシステムを知り、ベンチマークします。

4.16.3 保存時の暗号化

保存時の暗号化は、 CloudNativePG™クラスターの EDB Postgres AI で可能です。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。

4.16.4 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
- *Tablespaces* 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* **グループ** として定義されます。

4.16.5 ストレージクラスを介した構成

重要

CloudNativePG™クラスターの EDB Postgres ® AI は、すべてのストレージクラスと互換的に動作するように設計されています。いつものように、運用に展開する前に、制御された環境でストレージクラスを適切にベンチマークすることをお勧めします。

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:
  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
```

4.16.6 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
```

4.16.7 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。

> **参考**
>
> Reliability and the Write-Ahead Log を参照

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
```

> **重要**
>
> *walStorage* の削除はサポートされていません。 WAL の別のボリュームを追加すると、既存の Postgres クラスターから削除することはできません。

4.16.8 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、*Tablespaces* を参照してください。

4.16.9 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
    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
```

重要

専用の WAL ボリュームを作成した場合、このプロセス中に両方の PVC を削除する必要があります。WAL ボリューム PVC を再生成する場合にも、同じ手順が適用されます。これは、 *.spec.walStorage* セクションの `resizeInUseVolumes` も無効にすることにより実行できます。

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-4-join-v2     0/1     Completed   0          17s
cluster-example-4             1/1     Running     0          10s
```

4.16.10 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**事前プロビジョニング** としても知られています。

重要

ボリュームの事前プロビジョニングは、オペレーターの高可用性と自己修復機能に影響を与えるため、回避することをお勧めします。これは、CloudNativePG™クラスターの EDB Postgres ® AI が構築された完全な宣言的モデルを中断します。

CloudNativePG™クラスターの EDB Postgres 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` を作成できるようにします。

> **警告**
>
> 静的プロビジョニングでは、事前に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。

4.16.11 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することを検討します。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**シェアードナッシングアーキテクチャ** の原則に沿って、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、: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>`

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。

4.17 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、**ラベル** と **アノテーション**を介してリンクし、関係にすることができます。

> **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 で採用された技術です。

4.17.1 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 形式のバックアップの日付 `YYYYMMDD` 。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くて非推奨の 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/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かコールド。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をデータベースインスタンスに使用するポッドから区別します。

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 EDB Postgres ® AI for 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のいずれかに関連付けられている PostgreSQL ユーザーのタイプを指定し、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® CloudNativePG™クラスター用の AI 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が primary または replica であるかどうか。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が primary または replica かどうか。

4.17.2 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/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 Postgres プロセスのコアダンプを制御するフィルター、ビットマスクで表現されます。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 形式で表現された、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、バージョン 3.4 2023 年 1 月より前の--name オプションが使用できなかった場合の 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。許可される値は on および off です。

k8s.enterprisedb.io/managedSecrets オペレーターによって管理され、各 Postgres クラスターの ServiceAccount リソースに自動的に設定されるシークレットをプルします。

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/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 JSON-patch formatted patch, the patch will be applied on the instance␣
↪Pods.

 **☒ WARNING:**  This feature may introduce discrepancies between the
operator's 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してマークされた時間。

k8s.enterprisedb.io/validation CloudNativePG™クラスター管理の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、ボリュームスナップショットのバックアップが失敗したとみなす前に、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。

4.17.3 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。

4.17.4 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:
    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
```

4.17.5 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。

4.18 Monitoring

重要

Prometheus と Grafana のインストールは、このプロジェクトの範囲を超えています。これらはシステムに正しくインストールされていることを前提としています。ただし、実験のために、 *Part 4 of the Quickstart* で指示を提供します。

4.18.1 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。

重要

CloudNativePG™クラスターの EDB Postgres ® AI は、デフォルトで **メトリックのデフォルトセット** のセットをインストールします

`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 つのトランザクション
- `pg_monitor` ロールで実行
- `application_name` を `cnp_metrics_exporter` に設定して実行
- ユーザー `postgres` として実行

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**メインデータベース** 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。

参考

CloudNativePG™クラスターの EDB Postgres ® AI と Prometheus および Grafana との統合を評価することに興味がある場合は、*Part 4 of the quickstart* で簡単なセットアップガイドを見つけることができます。

Prometheus オペレーターを使用したモニタリング

特定の PostgreSQL クラスターは、 Prometheus Operator's リソース PodMonitor を使用してモニタできます。

クラスターリソース自分自身デフォルト false で.spec.monitoring.enablePodMonitor ～true を設定することにより、クラスターを正しくポイントする PodMonitor をオペレーターが自動的に作成できます。

重要

自動的に作成される `PodMonitor` への変更は、次の調整サイクルでオペレーターによってオーバーライドされます。カスタマイズが必要な場合は、以下の説明に従って行うことができます。

特定のクラスターに 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
```

重要

一意の名前、および正しいクラスターの名前空間とラベルたとえば *cluster-example* などを使用して上記の例を変更していることを確認します。

重要

このドキュメントの以前のバージョンで使用されていた `postgresql` ラベルは非推奨であり、将来的に削除されます。代わりに `k8s.enterprisedb.io/cluster` ラベルを使用して、インスタンスを選択してください。

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。

重要

.spec.monitoring.tls.enabled 設定を変更すると、クラスターのローリング再起動がトリガーされます。

`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。

```yaml
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:
          name: cluster-example-ca
          key: ca.crt
      serverName: cluster-example-rw
```

重要

一意の名前、および正しいクラスターの名前空間とラベルたとえば *cluster-example* などを使用して上記の例を変更していることを確認します。

重要

メトリックエンドポイントの`serverName` フィールドは、サーバー証明書で定義された名前のいずれかと一致する必要があります。デフォルトの証明書が使用されている場合、*serverName* 値は`<cluster-name>-rw` 形式である必要があります。

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,␣
```

(次のページに続く)

(前のページからの続き)

```
↪implying the absence of High Availability (HA). Ideally this value should match the␣
↪number of instances in the cluster.

#  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
```

(次のページに続く)

(前のページからの続き)

```
#  TYPE cnp_collector_replica_mode gauge

cnp_collector_replica_mode 0

#  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="17.5"} 17.5

#  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
```

(次のページに続く)

(前のページからの続き)

```
cnp_collector_first_recoverability_point 1.63238406e+09

#  HELP cnp_collector_lo_pages Estimated number of pages in the pg_largeobject table

#  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
```

(次のページに続く)

(前のページからの続き)

```
#  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_
↪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.
```

(次のページに続く)

(前のページからの続き)

```
# TYPE go_goroutines gauge

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.
```

(次のページに続く)

(前のページからの続き)

```
#  TYPE go_memstats_gc_sys_bytes gauge
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
```

(次のページに続く)

(前のページからの続き)

```
go_memstats_last_gc_time_seconds 1.6311727586032727e+09

#  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
```

(次のページに続く)

(前のページからの続き)

```
go_memstats_next_gc_bytes 8.689632e+06

#  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 **ベータ** 状態であり、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。

```
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。

重要

ユーザー定義メトリックが既存のメトリックを上書きすると、インスタンスマネージャーは json 警告ログを出力します。これには、メッセージ `Query with the same name already found. Overwriting the existing one.` と上書きされたクエリー名を含むキー `queryName` が含まれます。

ユーザー定義メトリックの例

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。

```yaml
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_is_in_recovery()
              THEN 0
              ELSE GREATEST (0,
                EXTRACT(EPOCH FROM (now() - pg_last_xact_replay_timestamp())))
              END AS lag,
              pg_is_in_recovery() AS in_recovery,
              EXISTS (TABLE pg_stat_wal_receiver) AS is_wal_receiver_up,
              (SELECT count(*) FROM 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*

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
     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 **シェルのようなパターン** つまり *、? または [] 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_database WHERE datallowconn AND NOT datistemplate 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。

> **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 current_database() ファンクションを使用します。

```
some_query: |
  query: |
    SELECT
     current_database() as datname,
     count(*) as rows
    FROM some_table
```

(次のページに続く)

(前のページからの続き)

```
  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
     current_database() as datname,
     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 edb_16

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 を実行するかどうかを評価します。 - 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。次のように定義されます。- <ColumnName> クエリーによって返された列の名前 - name メトリック内の列の ColumnName をオーバーライドします定義されている場合 - usage 以下に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

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。

重要

MONITORING_QUERIES_CONFIGMAP / MONITORING_QUERIES_SECRET を介して指定された ConfigMap または Secret は、常に固定名 `postgresql-operator-default-monitoring` でクラスターの名前空間にコピーされます。そのため、デフォルトのメトリックを使用する場合は、クラスターの名前空間にこの名前で ConfigMap を作成しないでください。

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、EDB Postgres ® AI for CloudNativePG™ Cluster ’ エクスポータに実装されていません。

4.18.2 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
```

4.18.3 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 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:8.2.1
    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
```

4.18.4 補助リソース

重要

これらのリソースは、説明と実験のために提供されており、実稼働システムへのいかなる種類の推奨を表すものではありません

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™クラスタークラスターとオペレーターの EDB Postgres の Grafana ダッシュボードは、専用のリポジトリに保持されます cloudnative-pg/grafana-dashboards

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。

4.18.5 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
```

(次のページに続く)

(前のページからの続き)

```
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。

重要

これにより、クラスター全体のモニタリングが有効になります。1 つの名前空間/プロジェクトでのみ必要な場合は、Red Hat の公式ドキュメントを参照するか、クラスター管理者に相談してください。

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。

4.19 Logging

CloudNativePG™クラスターの EDB Postgres ® AI は、セキュリティ上の理由から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。

重要

ログの長期保存と管理はオペレーターの範囲外であり、Kubernetes インフラストラクチャレベルで処理する必要があります。詳細については、 *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。

4.19.1 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。

重要

現在、ログレベルはインスタンスの起動時にのみ設定できます。クラスターが起動した後のクラスター仕様のログレベルの変更は、新しいポッドにのみ適用され、既存のポッドには適用されません。

4.19.2 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。

4.19.3 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,
```

(次のページに続く)

(前のページからの続き)


```
  "logger": "postgres",
  "msg": "record",
  "record": {
    "log_time": "2021-04-30 11:14:09.718 UTC",
    "user_name": "",
    "database_name": "",
    "process_id": "25",
    "connection_from": "",
    "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。

4.19.4 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。

重要

PGAudit ライブラリを `shared_preload_libraries` に追加する必要があります。 CloudNativePG™クラスターの EDB Postgres ® AI は、PostgreSQL 構成内の `pgaudit.*` パラメーターの存在に基づいて、これを自動的に管理します。オペレーターは、*shared_preload_libraries* からのライブラリの追加と削除の両方を処理します。

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。

重要

CloudNativePG™クラスターの EDB Postgres ® AI は、接続を受け入れるクラスター内のすべてのデータベースで `CREATE EXTENSION` および `DROP EXTENSION` コマンドを実行します。

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
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",
    "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",
}
```


レコードの各フィールドの詳細については、PGAudit documentation を参照してください。

4.19.5 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: quay.io/enterprisedb/edb-postgres-advanced:13
```

(次のページに続く)

(前のページからの続き)

```
  licenseKey: <LICENSE>

  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
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",
```

(次のページに続く)

(前のページからの続き)


```
    "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",
    "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、

4.19.6 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。

4.20 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. **オペレーター管理モード**–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™クラスターの 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。

4.20.1 オペレーター管理モード

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。

> **重要**
>
> 単純なリロード操作で十分であるため、証明書の更新は PostgreSQL サーバーのダウンタイムを発生させません。ただし、CloudNativePG™クラスターの EDB Postgres ® AI によって制御されないユーザー管理証明書は、更新プロセスの後に再発行する必要があります。

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。便利な代替方法は、*operator's configuration capability* を使用することです。

サーバー証明書

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。

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。

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。

クライアント証明書

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。

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。

4.20.2 ユーザー提供の証明書モード

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、引き続き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。

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。

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
```

(次のページに続く)

(前のページからの続き)

```
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:
    - 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
```

(次のページに続く)

(前のページからの続き)

```
    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。

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。

`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。

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
```

(次のページに続く)

(前のページからの続き)

```
  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
  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。

4.21 Client TLS/SSL connections

参考

CertificatesConfiguration を参照

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。

4.21.1 新しい証明書の発行

> **参考**
>
> *Certificates in the EDB Postgres ® AI for CloudNativePG™ Cluster plugin* を参照

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。

4.21.2 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
          readinessProbe:
            httpGet:
              port: 8080
              path: /tx
      volumes:
        - name: secret-volume-root-ca
          secret:
            secretName: cluster-example-ca
            defaultMode: 0600
```

(次のページに続く)

(前のページからの続き)

```
      - name: secret-volume-app
        secret:
          secretName: cluster-app
          defaultMode: 0600
```

このポッドは、 EDB Postgres ® AI for CloudNativePG™ Cluster オペレーター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 17.5 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 8.3.1 20191121 (Red Hat
8. 3.1-5), 64-bit
(1 row)
```

4.21.3 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。

4.22 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。

> **ヒント**
>
> アプリケーションでこれらのサービスを使用し、特定の PostgreSQL インスタンスに直接接続することを強くお勧めします。後者はクラスターの有効期間中に変更される可能性があります。

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。

> **参考**
>
> 接続プーラーとして PgBouncer を利用し、アプリケーションと PostgreSQL クラスターの間にアクセスレイヤーを作成する方法については、 *PgBouncer* **を使用した接続プーリング** を参照してください。

4.22.1 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。

4.22.2 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 **プライマリ**インスタンスを指すサービスの IP アドレス

4.22.3 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、データベースを**所有** してい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。

重要

ネットワーク上のスーパーユーザーのアクセスはデフォルトで無効になっています。

4.23 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。

4.23.1 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。

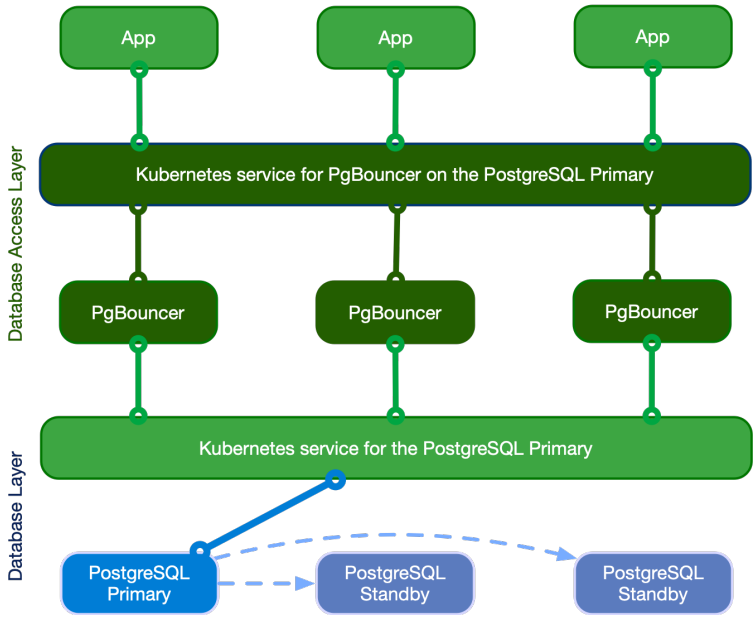


図 3: Applications writing to the single primary via PgBouncer

4.23.2 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
spec:
  cluster:
    name: cluster-example

  instances: 3
  type: rw
  pgbouncer:
    poolMode: session
    parameters:
      max_client_conn: "1000"
      default_pool_size: "10"
```

重要

プーラー名は、同じ名前空間内のクラスター名と同じにすることはできません。

この例では、 `cluster-example` と呼ばれる Postgres `Cluster` リソースに厳密に関連付けられる `pooler-example-rw` と呼ばれる `Pooler` リソースを作成します。これは、読み取り/書き込みサービス `rw` 、したがって `cluster-example-rw` によって識別されるプライマリを指します。

`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。

重要

Pooler リソースは、PgBouncer に `*` フォールバックデータベースのみを設定します。この設定は、クライアントから渡された接続文字列のすべてのパラメーターが PostgreSQL サーバーに中継されることを意味します。詳細については、 ["Section [databases]" in the PgBouncer documentation](https://www.pgbouncer.org/config.html#section-databases) を参照してください。

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。

> **参考**
>
> 詳細はこちら *PgBouncerSpec*

API リファレンスで。

4.23.3 リソースのライフサイクルのプール化

`Pooler` リソースはクラスター管理リソースではありません。プーラーは、必要に応じて手動で作成します。PostgreSQL クラスターごとに複数のプーラーを展開することもできます。

重要な点は、`Cluster` および `Pooler` リソースのライフサイクルが現在独立していることです。クラスターの削除はプーラーの削除を意味しません。その逆も同様です。

> **重要**
>
> プーラーがどのように動作するかを理解すると、使用可能なアーキテクチャの点で完全に自由になります。プーラーのないクラスター、単一のプーラーを含むクラスター、または複数のプーラー、つまりアプリケーションごとに 1 つを含むクラスターを持つことができます。

> **重要**
>
> オペレーターがアップグレードされると、プーラーポッドのローリングアップグレードが実行されます。これは、プーラーポッド内のインスタンスマネージャーもアップグレードされるようにするために必要です。

4.23.4 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側とサーバー PostgreSQL 側の両方で、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の EDB Postgres ® AI と透過的に統合されています。

具体的には、PgBouncer は PostgreSQL サーバーの証明書を再利用します。また、TLS クライアント証明書認証を使用して PostgreSQL サーバーに接続し、クライアントのパスワード認証のために `auth_query` を実行します 認証を使用する を参照してください。

コンテナは pgbouncer システムユーザーとして実行され、`pgbouncer` 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、クラスターが使用するものと同じ証明書を使用します。ただし、これらの証明書を提供した場合、プーラーは次の形式のシークレットを受け入れます。

1. ベーシック認証
2. TLS

3. 不透明

不透明の場合、使用する必要がある次の特定のキーを検索します。

- tls.crt
- tls.key

したがって、このシークレットを TLS シークレットとして扱い、そこから開始できます。

4.23.5 認証

パスワードベースの認証は、CloudNativePG™クラスターの EDB Postgres AI の PgBouncer のクライアントでサポートされている唯一の方法です。

内部的に、実装は PgBouncer の `auth_user` および `auth_query` オプションに依存しています。具体的には、演算子

- PostgreSQL サーバーに `cnp_pooler_pgbouncer` と呼ばれる標準ユーザーを作成します
- `postgres` データベースに lookup ファンクションを作成し、cnp_pooler_pgbouncer ユーザーに実行特権を付与します PoLA
- このユーザーの TLS 証明書を発行します
- `cnp_pooler_pgbouncer` を `auth_user` として設定します
- TLS 証明書を使用して PostgreSQL サーバーに対して `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たプーラーがないことを検出した場合、上記のすべてを削除します

重要

独自のシークレットを指定する場合、オペレーターはプーラーを自動的に統合しません。

独自のシークレットを指定した場合、プーラーを手動で統合するには、クラスター内から次のクエリを実行する必要があります。

最初に、ロールを作成する必要があります。

```
CREATE ROLE cnp_pooler_pgbouncer WITH LOGIN;
```

次に、アプリケーションデータベースごとに、`cnp_pooler_pgbouncer` に接続する権限を付与します。

```
GRANT CONNECT ON DATABASE { database name here } TO cnp_pooler_pgbouncer;
```

最後に、_superuser_として各アプリケーションデータベースに接続し、各アプリケーションデータベース内に認証ファンクションを作成します。

```
CREATE OR REPLACE FUNCTION public.user_search(uname TEXT)
  RETURNS TABLE (usename name, passwd text)
  LANGUAGE sql SECURITY DEFINER AS
  SELECT usename, passwd FROM pg_catalog.pg_shadow WHERE usename=$1;;

REVOKE ALL ON FUNCTION public.user_search(text)
  FROM public;

GRANT EXECUTE ON FUNCTION public.user_search(text)
  TO cnp_pooler_pgbouncer;
```

重要

user_search が `SECURITY DEFINER` ファンクションであることを考えると、 *postgres* ユーザーなどの `SUPERUSER` 特権を持つロールを介して作成する必要があります。

4.23.6 ポッドテンプレート

Pooler リソースの template セクションのポッドテンプレート仕様を利用できます。詳細については、API リファレンスの *PoolerSpec* を参照してください。

テンプレートを使用して、ポッドとノードのアフィニティおよびアンチアフィニティルールの微制御を含め、必要に応じてポッドを構成できます。デフォルトでは、コンテナは quay.io/enterprisedb/pgbouncer のイメージを使用します。

この例は、 `PodAntiAffinity` を指定する Pooler を示しています。

```
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:
```

(次のページに続く)

(前のページからの続き)

```
        requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
                - key: app
                  operator: In
                  values:
                    - pooler
            topologyKey: "kubernetes.io/hostname"
```

注釈

PodSpec の必須フィールドであるため、変更しない場合は`.spec.template.spec.containers` を`[]` に明示的に設定します。 *.spec.template.spec.containers* が設定されていない場合、Kubernetes API サーバーはマニフェストを適用しようとすると次のエラーを返します。*error validating "pooler.yaml": error validating data: ValidationError(Pooler.spec.template.spec): missing required field "containers"*

この例では、リソースを設定し、使用されるイメージを変更します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Pooler
metadata:
  name: pooler-example-rw
spec:
  cluster:
    name: cluster-example
  instances: 3
  type: rw

  template:
    metadata:
      labels:
        app: pooler
    spec:
      containers:
        - name: pgbouncer
          image: my-pgbouncer:latest
          resources:
            requests:
              cpu: "0.1"
              memory: 100Mi
            limits:
              cpu: "0.5"
              memory: 500Mi
```

4.23.7 サービステンプレート

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、またはサービスタイプの変更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。

4.23.8 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、それらの間の遅延が高い場合、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。

4.23.9 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**
>
> オペレーターは検証しないため、各オプションの値を正しく設定する責任があります。

これらはカスタマイズ可能な PgBouncer オプションであり、各パラメーターの PgBouncer ドキュメントへのリンクがあります。特に記載のない限り、デフォルト値は PgBouncer によって直接設定された値です。

- application_name_add_host
- autodb_idle_timeout
- cancel_wait_timeout
- client_idle_timeout
- client_login_timeout
- 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 デフォルトで無効になっています `0` 、以下の **モニタリングとメトリック** 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場合
- 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_tls_protocols
- 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。

4.23.10 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.
```

(次のページに続く)

(前のページからの続き)

```
#  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

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.
```

(次のページに続く)

(前のページからの続き)

```
#  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.

#  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
```

(次のページに続く)

(前のページからの続き)

```
# 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␣
↪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␣
```

(次のページに続く)

(前のページからの続き)

```
↪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

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
```

(次のページに続く)

(前のページからの続き)

```
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

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
```

(次のページに続く)

(前のページからの続き)

```
# 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

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
```

(次のページに続く)

(前のページからの続き)

```
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

#  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␣
```

(次のページに続く)

(前のページからの続き)

```
↪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.

#  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。

クラスターと同様に、 Prometheus operator's リソース PodMonitor を使用して特定のプーラーを監視できます。オペレーターが `Pooler` リソースの`.spec.monitoring.enablePodMonitor` から `true` を設定することにより、プーラーを正しくポイントする `PodMonitor` を作成できます。デフォルトは `false` です。

重要

自動的に作成される `PodMonitor` の変更は、次の調整サイクルでオペレーターによってオーバーライドされます。カスタマイズする必要がある場合は、次の例に示すように実行できます。

特定のプーラーに `PodMonitor` を手動で展開するには、次のように定義し、必要に応じて変更できます。

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: <POOLER_NAME>
spec:
  selector:
    matchLabels:
      k8s.enterprisedb.io/poolerName: <POOLER_NAME>
  podMetricsEndpoints:
    - port: metrics
```

4.23.11 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"
  }
}
```

4.23.12 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。 paused オプションを使用してこれを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。

> **参考**
>
> 詳細については、 PAUSE を参照してください。

重要

将来のバージョンでは、スイッチオーバー操作は PgBouncer プーラーと完全に統合され、 *PAUSE / RESUME* 機能を利用して、クライアントアプリケーションによる知覚的なダウンタイムを削減します。現在、 *paused* 属性を `true` に設定し、 :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>` を介して switchover コマンドを発行して、 *paused* 属性を `false` に復元することにより、同じ結果を得ることができます。

4.23.13 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と考えられます。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。

4.24 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。 EDB Postgres ® AI for CloudNativePG™ クラスター用語では、これは レプリカクラスターから分散トポロジへ 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。EDB Postgres ® AI for CloudNativePG™ クラスター用語では、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 — アーキテクチャ 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。

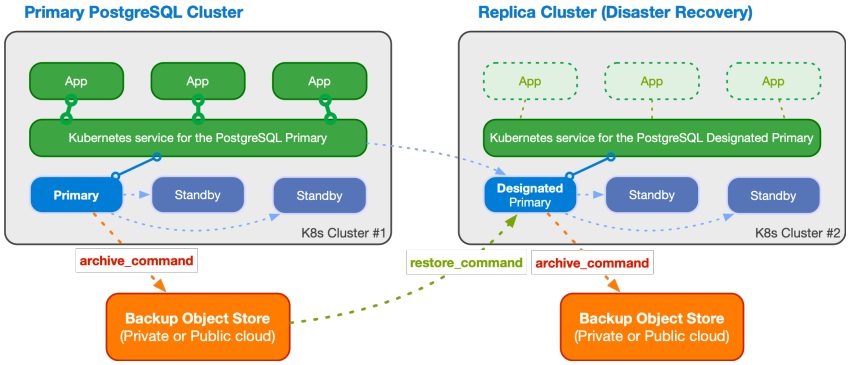


図 4: An example of multi-cluster deployment with a primary and a replica cluster

4.24.1 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プロモーション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、レプリカクラスターから分散トポロジへ 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 CNP-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、接続する WAL レシーバープロセスを開始します。ソースクラスターと受信データ。

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。

4.24.2 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` オプションとセットを使用して継続的リカバリーを有効にします `externalClusters` 名前をポイントする `source` フィールド。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。

4.24.3 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、南ヨーロッパの Kubernetes クラスターに CloudNativePG™クラスターの EDB Postgres® AI がインストールされており、`cluster-eu-south` という名前の PostgreSQL `Cluster` がプライマリとして機能していることを前提としています。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。

重要

続行する前に、必ずご確認ください *PostgreSQL* のロールについて

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。

4.24.4 スタンドアロンレプリカクラスター

重要

スタンドアロンレプリカクラスターは、CloudNativePG™クラスター 1.24 の EDB Postgres ® AI で分散トポロジ戦略を導入する前は、レプリカクラスターと呼ばれていました。

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。

重要

スタンドアロンレプリカクラスターは、主に読み取り専用のワークロードが関係するいくつかのユースケースに適しています。ディザスターリカバリーソリューションのセットアップを予定している場合は、上記の「分散トポロジ」を確認してください。

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
      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
    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。

4.24.5 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ています。

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"
```

(次のページに続く)

(前のページからの続き)

```
#  ...
```

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。

重要

常に目標を測定します。環境によっては、より迅速な結果を得るには、ボリュームスナップショットベースのリカバリーに依存する方が効率的である場合があります。独自の要件とインフラストラクチャに最適なアプローチを評価および選択します。

4.25 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。

4.25.1 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。

4.25.2 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。

4.25.3 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**ローカルストレージ** 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。

4.25.4 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、インスタンスが格納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* 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。

4.25.5 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。

4.25.6 ノードのメンテナンスウィンドウ

重要

CloudNativePG™クラスターの EDB Postgres ® AI は、ノードのメンテナンスウィンドウのサポートを継続しますが、以前のセクションで説明したように、現在ポッドの中断バジェットの直接制御に移行することをお勧めします。このセクションは、主に下位互換性のために保持されています。

リリース 1.23 より前は、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。標準の自己処理を回避するには、`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ているときに開始する修復プロシージャ。

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。

重要

PodDisruptionBudget 管理は、 *.spec.enablePDB* フィールドを `false` に設定することにより無効にできます。その場合、オペレーターは `PodDisruptionBudgets` を作成せず、以前に作成されている場合は削除します。

reusePVC が false に設定された単一インスタンスクラスター

重要

高可用性を保証するために、常に複数のインスタンスでクラスターを作成することをお勧めします。

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。

4.26 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。

4.26.1 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 1.27.0 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/v1.27.0/kubectl-
↪cnp_1.27.0_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 (1.27.0) ...
Setting up kubectl-cnp (1.27.0) ...
```

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 1.27.0 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/v1.27.0/kubectl-
↪cnp_1.27.0_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:
 cnp                x86_64              1.27.0-1                @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

EDB Postgres ® AI for CloudNativePG™クラスタープラグインは、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
```

重要

スクリプトの名前は、kubectl オートコンプリートプロセスで使用されるため、指定されたものと正確に一致する必要があります。

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

```

(次のページに続く)

(前のページからの続き)

```
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
```

重要

スクリプトの名前は、kubectl オートコンプリートプロセスで使用されるため、指定されたものと正確である必要があります

4.26.2 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。

重要

上記のステータス情報は、異なる時間に異なる場所で取得されたため、わずかに一貫性のない戻り値が発生します。たとえば、メインヘッダーの `Current Write LSN` 場所は、2 つの異なる時間間隔で取得されるため、インスタンスステータスの `Current LSN` フィールドとは異なる場合があります。

```
kubectl cnp status sandbox
```

```
Cluster Summary
Name:                default/sandbox
System ID:           7423474350493388827
PostgreSQL Image:    docker.enterprisedb.com/k8s_enterprise/edb-postgres-extended:16.4
Primary instance:    sandbox-1
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  1.27.0           k8s-eu-
↪worker
sandbox-2  0/604DE38    Standby (async)   OK      BestEffort  1.27.0           k8s-eu-
↪worker2
sandbox-3  0/604DE38    Standby (async)   OK      BestEffort  1.27.0           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
```

(次のページに続く)

(前のページからの続き)

```
System ID:           7423474350493388827
PostgreSQL Image:    docker.enterprisedb.com/k8s_enterprise/edb-postgres-extended:16.4
Primary instance:    sandbox-1
Primary start time:  2024-10-08 18:31:57 +0000 UTC (uptime 2m4s)
Status:              Cluster in healthy state
Instances:           3
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
- ---             ----     -------------  ---------------  -----------------------  --
↪-----------------
```

(次のページに続く)

(前のページからの続き)

```
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
- ---       -----------  ----------------  ------  ---         ---------------  ----
sandbox-1  0/6053720    Primary           OK      BestEffort  1.27.0           k8s-eu-
↪worker
sandbox-2  0/6053720    Standby (async)   OK      BestEffort  1.27.0           k8s-eu-
↪worker2
sandbox-3  0/6053720    Standby (async)   OK      BestEffort  1.27.0           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。

重要

Secrets と ConfigMap のすべての機密情報は編集されています。データマップには**キー**が表示されますが、値は空になります。フラグ `-S` / *--stopRedaction* は、リダクションを無効にし、値を表示します。自己の責任でのみ使用します。これはプライベートデータを共有します。

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.abbr}}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":"1.27.0",
↪"build":{"Version":"1.27.0+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":"1.27.0",
↪"build":{"Version":"1.27.0+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␣
```

(次のページに続く)

(前のページからの続き)

```
↪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␣
↪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。

ハイバネーションは、 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

休止している場合、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ーの一部であるため、フェンスされたインスタンスを持つクラスターは休止できません。

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 hibernate 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 C - Common object stores for backups 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。

> **重要**
>
> *postgres* ユーザーとして接続するため、運用環境では、この方法は許可された担当者のみが細心の注意を払って使用する必要があります。

```
$ 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)
```

(次のページに続く)

(前のページからの続き)

```
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 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。

重要

Kubernetes 運用環境での pgAdmin の展開は、このドキュメントの範囲を超えており、より広範には EDB Postgres ® AI for CloudNativePG™ Cluster プロジェクトの範囲を超えています。

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。

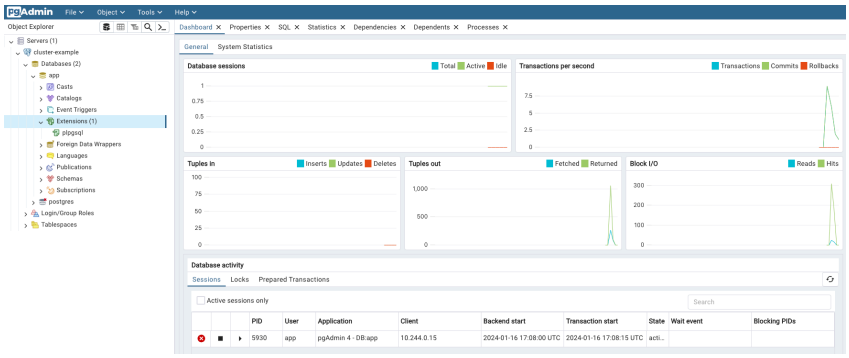


図 5: Screenshot of desktop installation of pgAdmin

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。

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ステップには、アプリケーションを新しいデータベース**カットオーバー** 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。

4.26.3 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。

4.26.4 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

次のページに続く

表 7 – 前のページからの続き

Command	Resource Permissions
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
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。

```
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
```

重要

resources ごとおよび `apiGroups` ごとに動詞を制限したままにすると、意図した以上の権限を誤って付与するのを防ぐことができます。

4.27 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。

> **重要**
>
> 2 フェーズプロシージャは、WAL レシーバーが正常に停止でき、障害が発生したプライマリが再起動時に WAL のストリーミングを再度開始しないことを保証します。これらの安全対策は、新しいプライマリとレプリカ間のタイムラインの不一致を防ぎます。

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**高速シャットダウン**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。

4.27.1 RTO と RPO のインパクト

フェイルオーバーにより、サービスが影響を受ける バックアップ頻度と *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。

4.27.2 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。

4.27.3 Failover Quorum クォーラムベースのフェールオーバー

> **警告**
>
> *フェールオーバークォーラム*は、バージョン 1.27.0 で導入された実験的な機能です。運用環境では注意して使用します。

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。

フェールオーバークォーラムは、 `Cluster` リソースにアノテーション `alpha.k8s.enterprisedb.io/failoverQuorum="true"` を設定することにより有効にできます。

> **注釈**
>
> この機能が実験フェーズを終了すると、アノテーション *alpha.k8s.enterprisedb.io/failoverQuorum* は`Cluster` リソースの構成オプションに置き換えられます。

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、 **プロモートブルレプリカ** 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。

> **重要**
>
> ユーザーは `FailoverQuorum` リソースを直接変更しないでください。 PostgreSQL の構成の変更中に、構成を決定できない場合、*FailoverQuorum* リソースがリセットされ、新しい構成が適用されるまでフェイルオーバーを防止します。

`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**
>
> *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、クライアントに返されたが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	OK
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	OK
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	OK
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	OK

次のページに続く

表 11 – 前のページからの続き

R	W	N	Failover
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	NO

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	NO

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	OK
1	1	2	NO

^1

4.28 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。

4.28.1 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"]
[...]
```

4.28.2 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 "*"
```

4.28.3 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 **準備完了** 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。

フェンスされたインスタンスが削除されると、ポッドは通常通り再作成されますが、ポストマスターは起動されません。これは、インスタンスが Crashlooping の場合に非常に役立ちます。

4.29 Declarative hibernation

CloudNativePG™クラスターの EDB Postgres ® AI は、PostgreSQL クラスターを常に起動、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。

4.29.1 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
```

休止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:   quay.io/enterprisedb/postgresql:17.5
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
[..]
```

4.29.2 リハイドレート

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。

4.30 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。

> **重要**
>
> このセクションでは、 PostGIS に精通していることを前提として、CloudNativePG™クラスターの EDB Postgres ® AI を介して Kubernetes で PostGIS データベースを使用して新しい PostgreSQL クラスターを作成する方法に関する基本情報を提供します。

CloudNativePG コミュニティは、公式の *PostGIS images hosted on DockerHub* 上にビルドされた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、PostGIS を含む EDB Postgres Advanced Server のコンテナイメージを提供し、公式 registry on Quay.io

`-postgis` サフィックスを使用します。

4.30.1 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
- インスタンスが複数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、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、`template1` データベースにインストールします

> **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* 。

4.30.2 PostGIS で新しい PostgreSQL クラスターを作成する

PostGIS 3.2 を使用して新しい PostgreSQL 14 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**
>
> 構成に関する規則は EDB Postgres ® AI for CloudNativePG™ クラスターに適用されますが、運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 14 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.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: postgis-example
spec:
  instances: 3
  imageName: ghcr.io/enterprisedb/postgresql:16-postgis
  bootstrap:
    initdb:
      postInitTemplateSQL:
        - CREATE EXTENSION postgis;
        - CREATE EXTENSION postgis_topology;
        - CREATE EXTENSION fuzzystrmatch;
        - CREATE EXTENSION postgis_tiger_geocoder;

  storage:
    size: 1Gi
```

この例では、 `app` と呼ばれるアプリケーションデータベースを実際に作成する前に、`template1` データベースに対してクエリーのリストを実行する `postInitTemplateSQL` オプションに依存しています。これは、マニフェストを適用し、クラスターが起動すると、上記の拡張機能がテンプレートデータベースとアプリケーションデータベースの両方にインストールされ、使用できる状態になることを意味します。

> **注釈**
>
> 時間をかけて、*postInitApplicationSQL* などの *API Reference* からで使用可能なオプションを確認します。

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 exec -ti postgis-example-1 -- psql app
Defaulted container "postgres" out of: postgres, bootstrap-controller (init)
psql (16.0 (Debian 16.0-1.pgdg110+1))
Type "help" for help.

app=# SELECT * FROM pg_available_extensions WHERE name ~ ^postgis ORDER BY 1;
           name           | default_version | installed_version |
     comment
- -------------------------+-----------------+-------------------+--------------------
↪---------------------------------------
 postgis                  | 3.2.2           | 3.2.2             | PostGIS geometry␣
↪and geography spatial types and functions
 postgis-3                | 3.2.2           |                   | PostGIS geometry␣
↪and geography spatial types and functions
 postgis_raster           | 3.2.2           |                   | PostGIS raster␣
↪types and functions
 postgis_raster-3         | 3.2.2           |                   | PostGIS raster␣
↪types and functions
 postgis_sfcgal           | 3.2.2           |                   | PostGIS SFCGAL␣
↪functions
 postgis_sfcgal-3         | 3.2.2           |                   | PostGIS SFCGAL␣
↪functions
 postgis_tiger_geocoder   | 3.2.2           | 3.2.2             | PostGIS tiger␣
↪geocoder and reverse geocoder
 postgis_tiger_geocoder-3 | 3.2.2           |                   | PostGIS tiger␣
↪geocoder and reverse geocoder
 postgis_topology         | 3.2.2           | 3.2.2             | PostGIS topology␣
↪spatial types and functions
 postgis_topology-3       | 3.2.2           |                   | PostGIS topology␣
↪spatial types and functions
(10 rows)
```

次のステップは、 `postInitTemplateSQL` セクション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 |   Schema   |                          Description
- ----------------------+---------+------------+-------------------------------------
```

(次のページに続く)

(前のページからの続き)

```
↪-----------------------
 fuzzystrmatch          | 1.1     | public     | determine similarities and distance␣
↪between strings
 plpgsql                | 1.0     | pg_catalog | PL/pgSQL procedural language
 postgis                | 3.2.2   | public     | PostGIS geometry and geography␣
↪spatial types and functions
 postgis_tiger_geocoder | 3.2.2   | tiger      | PostGIS tiger geocoder and reverse␣
↪geocoder
 postgis_topology       | 3.2.2   | topology   | PostGIS topology spatial types and␣
↪functions
(5 rows)
```

最後に

```
app=# SELECT postgis_full_version();
                                                                                postgis_
↪full_version
- ------------------------------------------------------------------------------------------
↪------------------------------------------------------------------------------------------
↪---
 POSTGIS="3.2.2 628da50" [EXTENSION] PGSQL="140" GEOS="3.9.0-CAPI-1.16.2" PROJ="7.2.1"
 LIBXML="2.9.10" LIBJSON="0.15" LIBPROTOBUF="1.3.3" WAGYU="0.5.0 (Internal)" TOPOLOGY
(1 row)
```

4.31 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

重要

PostgreSQL versions officially supported by PGDG のみが許可されます。

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。

EDB は public PostgreSQL container images を提供および保守します

CloudNativePG™クラスターの EDB Postgres と完全な互換性があります。これらの画像は quay.io で公開されています。

4.31.1 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適用されません。

4.32 Image Volume Extensions

CloudNativePG™クラスターの EDB Postgres ® AI は、ポッドのスタートアップに *Kubernetes `ImageVolume* feature <https://kubernetes.io/docs/tasks/configure-pod-container/image-volumes/>`_ を使用して `Cluster` への **PostgreSQL 拡張機能の動的ロード** をサポートします

そして、このプロジェクトが貢献した PostgreSQL 18 で導入された `extension_control_path` GUC。

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、既知のファイルシステムパスの実行中のポッド内の読み取り専用で不変ボリュームとしてマウントできます。

拡張機能は *shared_preload_libraries* を使用してグローバルに、または `CREATE EXTENSION` コマンドを使用してデータベースレベルで利用できるようにできます。後者の場合、 *PostgreSQL Database Management* を使用できます

PostgreSQL データベース内での一貫した自動拡張機能セットアップを保証します。

4.32.1 メリット

イメージボリューム拡張機能は、拡張機能のディストリビューションから PostgreSQL オペランドコンテナイメージのディストリビューションを分離します。これにより、PostgreSQL イメージ内のビルド時に拡張機能を定義して埋め込む必要がなくなります。これは、セキュリティとサプライチェーンの観点からなど、コンテナ化されたワークロードとしての PostgreSQL の主な採用のブロッカーです。

その結果、次のことができます。

- *official PostgreSQL `minimal* operand images <https://github.com/enterprisedb/docker-postgres?tab=readme-ov-file#minimal-images>`_ を使用します

CloudNativePG™クラスターの EDB Postgres AI によって提供されます。

- カスタム PostgreSQL イメージをリビルドまたはメンテナンスせずに、必要な拡張機能を `Cluster` 定義に動的に追加します。
- 不変の最小限のセキュアベースイメージを使用し、各ワークロードに必要な拡張機能のみを追加することにより、操作面を削減します。

拡張イメージは、 **画像の仕様** に従ってビルドする必要があります。

4.32.2 要件

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` のサポート。
- **Kubernetes 1.33** 、`ImageVolume` 機能ゲートが有効になっています。
- **EDB Postgres ® AI for CloudNativePG™クラスター互換の拡張コンテナイメージ** 、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一致します。
  - `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。

4.32.3 その仕組み

拡張機能イメージは、`Cluster` リソースの `.spec.postgresql.extensions` スタンザで定義され、PostgreSQL クラスターに追加する拡張機能の順序付けリストを受け入れます。

> **注釈**
>
> フィールドレベルの詳細については、 API reference for `ExtensionConfiguration <API Reference>` を参照してください。

各イメージボリュームは `/extensions/<EXTENSION_NAME>` にマウントされます。

デフォルトでは、CloudNativePG™クラスターの EDB Postgres ® AI は、関連する GUC を自動的に管理し、次の設定を行います。

- `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`

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。

> **注釈**
>
> 拡張機能コンテナイメージの構築方法とそのレイアウトによっては、イメージ構造と一致するようにデフォルトの `extension_control_path` および `dynamic_library_path` 値を調整する必要がある場合があります。

重要

拡張機能イメージに共有ライブラリが含まれる場合、クラスターで使用される PostgreSQL コンテナイメージと同じ 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 アーキテクチャでコンパイルして、互換性を保証し、ランタイムの問題を防ぐ必要があります。

4.32.4 新しい拡張機能を追加する方法

CloudNativePG™クラスターの EDB Postgres ® AI のデータベースに拡張機能を追加するには、いくつかの手順を必要とします。

1. PostgreSQL が検出してロードできるように、`Cluster` リソースに拡張イメージを定義します。
2. ライブラリを *shared_preload_libraries* に追加する

拡張機能で必要な場合。

3. 拡張機能が `CREATE EXTENSION` をサポートしている場合、インストールする `Database` リソースで拡張機能を宣言します。

警告

拡張機能イメージと PostgreSQL 構成設定 *shared_preload_libraries* などを同時に変更することは避けてください。最初に、新しい拡張機能イメージでポッドをロールアウトし、PostgreSQL 構成を更新します。この制限は、CloudNativePG™クラスターの EDB Postgres AI の将来のリリースで対処される予定です。

説明のために、このガイドでは、 `CREATE EXTENSION` をサポートする `foo` という名前の単純な架空の拡張機能を使用します。

新しい拡張機能の `Cluster` リソースへの追加

`.spec.postgresql.extensions` スタンザを使用して、`ImageVolume` ベースの拡張機能を `Cluster` に追加できます。例

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: foo-18
spec:
  # ...
  postgresql:
    extensions:
      - name: foo
        image:
```

(次のページに続く)

(前のページからの続き)

```
        reference: # registry path for your extension image
  # ...
```

name フィールドは **必須**で、マウントパスこの例では/extensions/foo を決定するため、**クラスター内で一意である必要があります** 。**小文字の英数字またはハイフン** `-` 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。

image スタンザは *Kubernetes `ImageVolume* API <https://kubernetes.io/docs/tasks/configure-pod-container/image-volumes/>`_ の後にあります。 reference は、拡張機能イメージの有効なコンテナレジストリパスを指している必要があります。

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 *Rolling Updates* 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。

マウントされると、CloudNativePG™クラスターの EDB Postgres ® AI は、以下を追加することにより PostgreSQL を自動的に構成します。

- /extensions/foo/share ～extension_control_path
- /extensions/foo/lib ～dynamic_library_path

これにより、次のセクションで説明するように、PostgreSQL コンテナがデータベースから要求されたときに foo 拡張機能を提供できる準備が保証されます。 CREATE EXTENSION foo コマンドは、reconciliation of the `Database resource <PostgreSQL Database Management>` 中に自動的にトリガーされ、PostgreSQL が次の場所を見つけるため、追加の構成なしで機能します。

- /extensions/foo/share/extension/foo.control の拡張制御ファイル
- /extensions/foo/lib/foo.so の共有ライブラリ

新しい拡張機能の Database リソースへの追加

PostgreSQL インスタンスで拡張機能が利用できるようになると、宣言的データベースを活用して *manage the lifecycle of your extensions* ことができます

ターゲットデータベース内で。

foo の例で続けると、次のリソース定義を使用して、foo-18 クラスターの app データベースへの foo 拡張機能のインストールを要求できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Database
metadata:
  name: foo-app
```

(次のページに続く)

(前のページからの続き)

```
spec:
  name: app
  owner: app
  cluster:
    name: foo-18
  extensions:
    - name: foo
      version: 1.0
```

CloudNativePG™クラスターの EDB Postgres ® AI は、このリソースを自動的にリコンサイルし、 app データベース内で CREATE EXTENSION foo コマンドがまだインストールされていない場合は実行し、手動介入なしで目的の状態が維持されるようにします。

4.32.5 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。

「構成よりコンベンション」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 インスタンスマネージャープロセスの 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。これにより、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。

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:
  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
```

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:
  # ...
  postgresql:
    extensions:
      - name: geospatial
        extension_control_path:
          - postgis/share
          - pgrouting/share
        dynamic_library_path:
          - postgis/lib
          - pgrouting/lib
        # ...
        image:
          reference: # registry path for your geospatial image
      # ...
    # ...
  # ...
```

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:
  # ...
  postgresql:
    extensions:
      - name: postgis
        # ...
        ld_library_path:
          - syslib
        image:
          reference: # registry path for your PostGIS image
      # ...
    # ...
  # ...
```

CloudNativePG™クラスターの EDB Postgres ® AI は、`/extensions/postgis/system` 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、**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。 *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。

4.32.6 画像の仕様

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 SQL ファイルが含まれています。
- `lib` 拡張機能の共有ライブラリたとえば `<EXTENSION>.so` および追加の必要ライブラリが含まれています。

この構造に従うことで、拡張機能は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 拡張機能開発者は、アーティファクト配布の一部として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し、Kubernetes 環境内で拡張機能を簡単に使用できるようにすることをお勧めします。理想的には、拡張機能イメージは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アーキテクチャをターゲットとし、特定の PostgreSQL バージョンに関連付けられ、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 Debian または RPM パッケージを使用してビルドする必要があります。このアプローチでは、クラスターで使用される PostgreSQL イメージとの一貫性、セキュリティ、および互換性が保証されます。

4.32.7 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 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。

4.33 CNP-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。

4.33.1 なぜ CNP-I なのでしょうか？

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。

CNP-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。

CNP-I は、コアを中断せずにバックアップ、リカバリー、サブリソースのリコンシリエーションなど、クラスターのライフサイクルのキーポイントで 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。プロジェクト。

CNP-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。

4.33.2 プラグインの登録

CNP-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。

CloudNativePG™クラスターの EDB Postgres ® AI は、 **起動時に**プラグインを検出します。次の 2 つの方法のいずれかで登録できます。

- 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

サイドカーとして実行する場合、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
```

(次のページに続く)

(前のページからの続き)

```
      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。

> **⚠ 警告**
>
> CloudNativePG™クラスターの EDB Postgres ® AI は、プラグインを動的に検出しません。新しいプラグインを展開する場合、それを検出するには**オペレーターを再起動**する必要があります。

展開例

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cnpg-i-plugin-example
spec:
  template:
    [...]
    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>` — EDB Postgres ® AI for CloudNativePG™ クラスター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

```yaml
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。

```yaml
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。

4.33.3 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

```
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

4.33.4 コミュニティプラグイン

CNP-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済みの信頼できる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。

CNP-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。

4.34 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 の実装の信頼性が向上し、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。

4.34.1 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。

4.34.2 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。

4.34.3 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。

4.34.4 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ーの 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。

4.35 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。

4.35.1 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。

4.35.2 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。

4.36 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

> **重要**
>
> *pgbench* および `fio` は、ステージングまたは実稼働前環境で実行する必要があります。運用環境ではこれらのプラグインを使用しないでください。データベースと、同じ共有環境で実行される他のワークロード/アプリケーションに壊滅的な結果をもたらす可能性があります。

4.36.1 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>
```

> **重要**
>
> pgbench を参照してください。

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>
```

4.36.2 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。

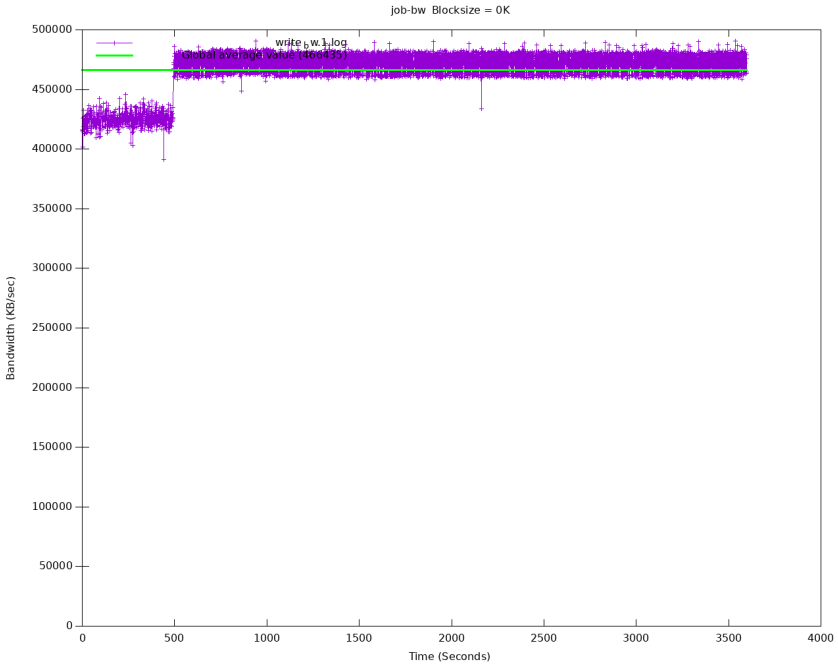


図 6: Sequential writes bandwidth

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5章　EDB Enhancements

5.1 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。

5.1.1 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。

PostgreSQL コンテナイメージは、 quay.io/enterprisedb/postgresql で入手できます。

5.2 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。

次のドキュメントは、ライセンスキーを使用してオペレーターをインストールしたユーザーのみを対象としています。

5.2.1 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。

> **参考**
>
> 詳細については、 *Operator configuration* を参照してください。

企業レベルライセンスがインストールされると、クラスターステータス内でライセンスキーの有効性を確認できます。

```
kubectl get cluster cluster_example -o yaml
[...]
status:
  [...]
  licenseStatus:
    licenseExpiration: "2021-11-06T09:36:02Z"
    licenseStatus: Trial
    valid: true
    isImplicit: false
    isTrial: true
[...]
```

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
```

5.2.2 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:
  [...]
```

(次のページに続く)

(前のページからの続き)

```
  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。

5.2.3 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/limited-use-license で入手可能な EDB 制限使用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.

5.2.4 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。

5.3 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 **アーキテクチャ** 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。

ほとんどの場合、CloudNativePG™クラスターの EDB Postgres® AI のデフォルトのクラスター全体のインストールが推奨され、PostgreSQL クラスターの集中管理または委任管理 RBAC 定義に従って特定のユーザー/プロジェクトに制限されます *OpenShift* **の重要な概念** を参照してください

および以下の **ユーザーと権限**)。

> **重要**
>
> インストールとアップグレードの両方のプロセスでは、 *cluster-admin* 権限を持つアカウントを使用して OpenShift Container Platform クラスターにアクセスする必要があります。 *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。"_。

5.3.1 アーキテクチャ

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。

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。

5.3.2 OpenShift の重要な概念

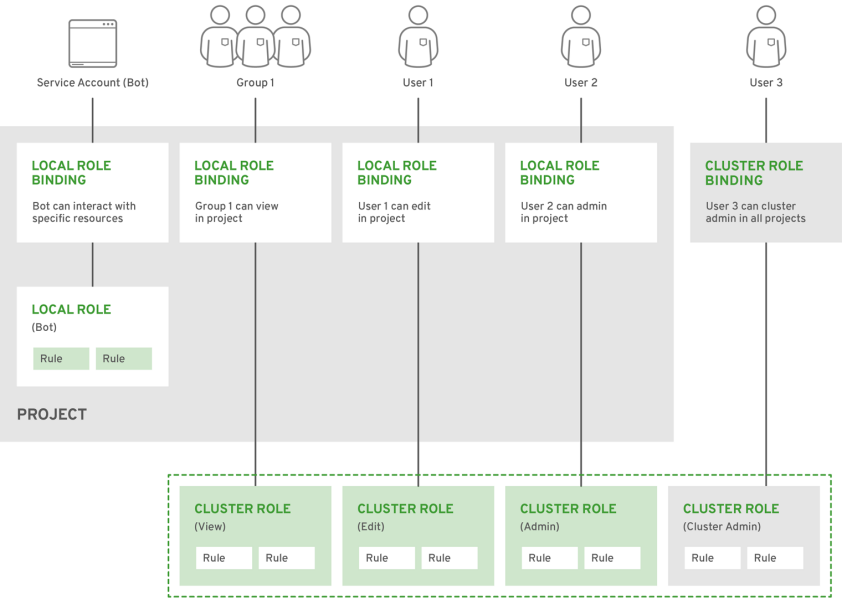
CloudNativePG™クラスターオペレーターの EDB Postgres ® AI が OpenShift 環境にどのように適合するかを理解するには、次の Kubernetes 関連のトピックに精通する必要があります。

- 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

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。

> **参考**
>
> *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。

マルチテナント管理の制限

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。

5.3.3 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™クラスター用の 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。

5.3.4 Web コンソールを介したインストール

EDB プライベートレジストリへのアクセスの保証

> **重要**
>
> オペレーターイメージとオペランドイメージの両方が保存されているプライベート EDB リポジトリへのアクセスが必要です。アクセスには、有効な EDB subscription plan が必要です。詳細は *Accessing EDB private image registries* を参照してください。

OpenShift インストールは、プライベートリポジトリに保持されているオペランドとオペレータイメージにアクセスするためにプルシークレットを使用します。

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` 、EDB Postgres ® AI for CloudNativePG™ クラスターオペレーターイメージの場合

次のように、`oc create` コマンドを介して各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="@@REPOSITORY@@" \
  --docker-password="@@TOKEN@@"
```

そこで

- `@@REPOSITORY@@` は、*Which repository to choose?* で説明したように、リポジトリの名前です。
- `@@TOKEN@@` は、**トークンの取得方法** で説明しているように、EDB アカウントのリポジトリトークンです。

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。
2. `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。
3. Operator に関する情報を読み、 `Install` を選択します。
4. 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 **単一プロジェクトのインストール** 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状況に従って 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

重要

オペレーターのアップグレードのプロセスは、 *PostgreSQL Upgrades* に説明されています。

重要

単一のプロジェクト、技術的には`OwnNamespace` インストールモードにオペレーターを同じクラスターに複数回インストールすることが可能です。 API が同じである限り、異なるアップグレードポリシーですべての名前空間にオペレーターインストールが存在します **マルチテナント管理の制限** を参照してください。

オペレーターのクラスター全体とローカルインストールの選択は、重要なターニングポイントです。既存のローカルインストール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ようとすると、以下のエラーがスローされることによりブロックされます。続行する場合は、最初にオペレーターのすべてのローカルインストールを削除する必要があります。

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™クラスターに 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。

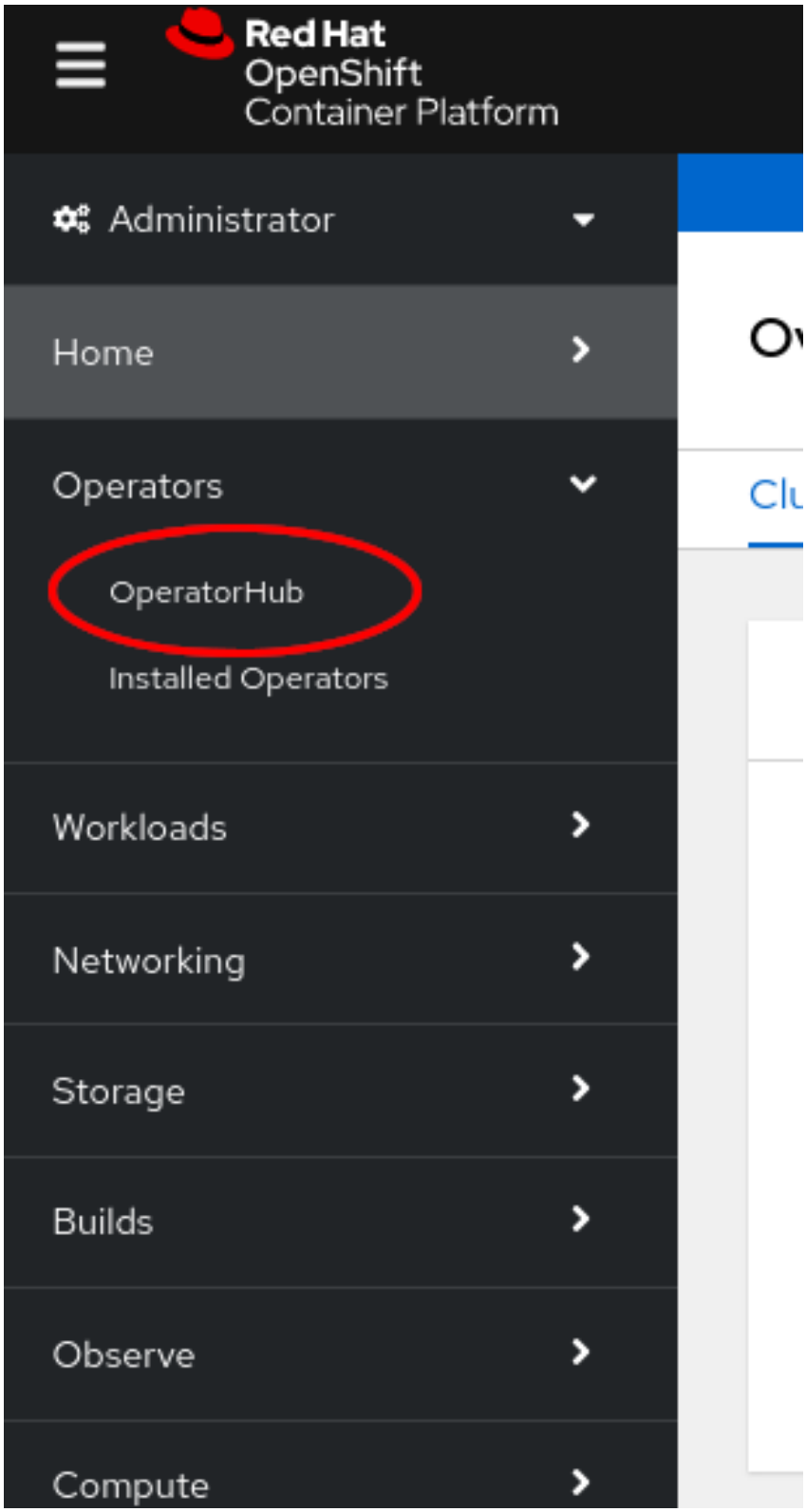


図 1: Menu OperatorHub

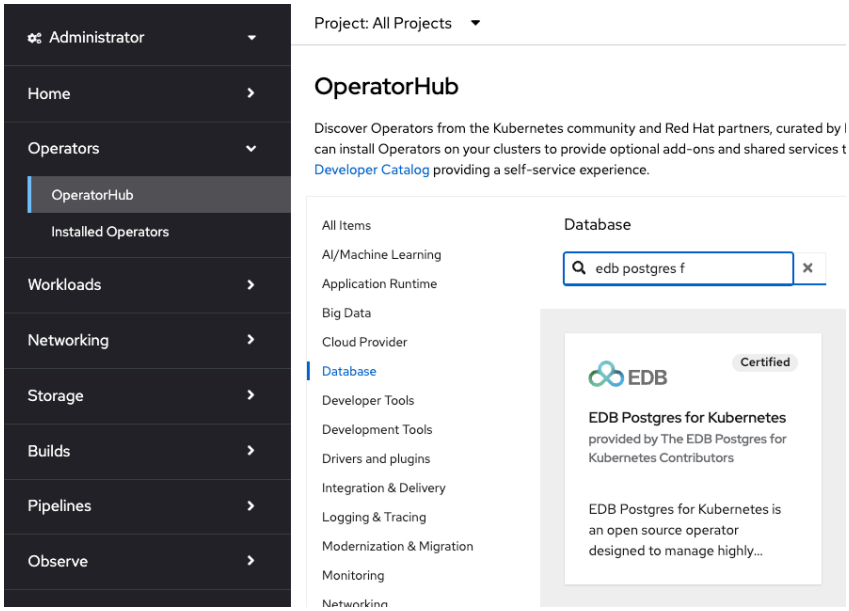


図 2: Install OperatorHub

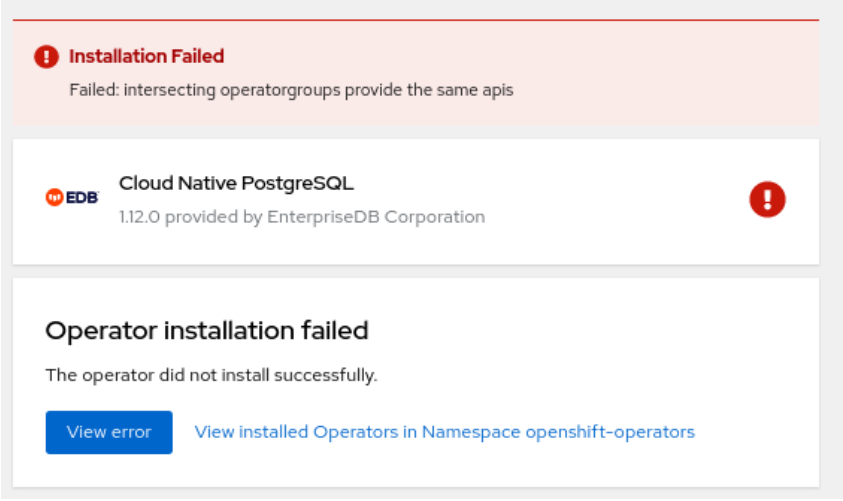


図 3: Install failing

> **警告**
>
> これは、OpenShift クラスター内のすべてのユーザーが Kubernetes オペレーターの EDB Postgres を使用、*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。 EDB Postgres ® AI for CloudNativePG™ Cluster の管理対象カスタムリソース主に `Cluster` もの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。

Web コンソールから、`Installation mode` として `All namespaces on the cluster (default)` を選択します。

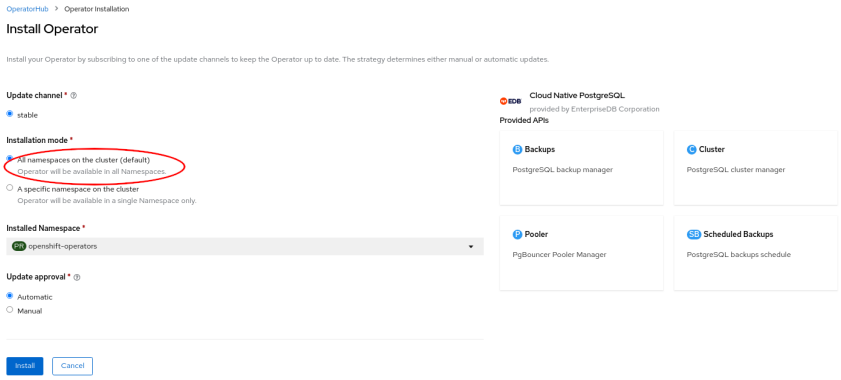


図 4: Install all namespaces

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

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 Kubernetes オペレーターの EDB Postgres を使用、*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、ユーザーが EDB Postgres ® AI for CloudNativePG™ Cluster の管理カスタムリソース主に `Cluster` ものと対話するためには、名前空間に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。

Web コンソールから、 `Installation mode` として `A specific namespace on the cluster` を選択し、ターゲット名前空間この例では `proj-dev` を選択します。

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。

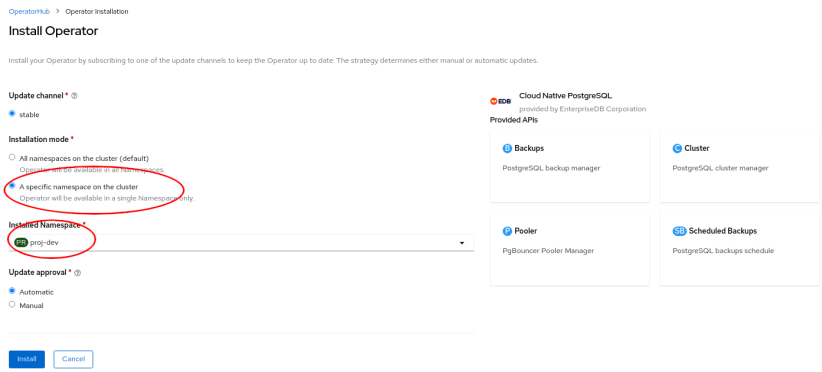


図 5: Install one namespace

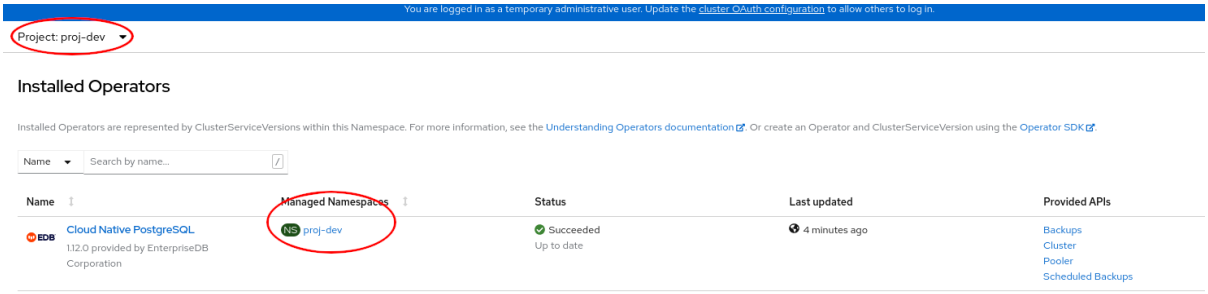


図 6: Install one namespace list

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。

このインストールプロセスは、同じ OpenShift クラスター内の複数の名前空間で繰り返すことができ、さまざまなプロジェクトにオペレーターを独立してインストールできます。この場合、**マルチテナント管理の制限** を必ず読んでください。

5.3.5 oc CLI を介したインストール

重要

OpenShift CLI `oc のインストール <OpenShift CLI *oc* のインストール>` を参照してください。

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、

OpenShift Container Platform Web コンソールを使用する代わりに、OperatorHub から CloudNativePG™ Cluster Operator の EDB Postgres ® AI をインストールし、 oc コマンドラインインターフェイスを使用してサブスクリプションを作成できます。oc CLI を介して、すべての名前空間、単一の名前空間または複数の名前空間にオペレーターをインストールできます。

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* .

このセクションでは、主に OperatorGroup および Subscription オブジェクトを作成することによる、簡単な例を使用して複数のプロジェクトへのオペレーターのインストールについて説明します。

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。

1. cloud-native-postgresql オペレーターが OperatorHub から利用できることを確認します。

```
oc get packagemanifests -n openshift-marketplace cloud-native-postgresql
```

2. 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
```

3. 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、my-operators 名前空間に OperatorGroup オブジェクトを作成します。

```
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
```

重要

また、*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。

4. my-operators 名前空間に Subscription オブジェクトを作成して、 openshift-marketplace の 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
metadata:
  name: cloud-native-postgresql
  namespace: my-operators
spec:
  channel: fast
  name: cloud-native-postgresql
  source: certified-operators
  sourceNamespace: openshift-marketplace
```

5. 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、 EDB Postgres ® AI for CloudNativePG™ Cluster の事前定義された ClusterRole s 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。

上記の操作の結果は、以下の画像に示すように、Web コンソールから確認することもできます。

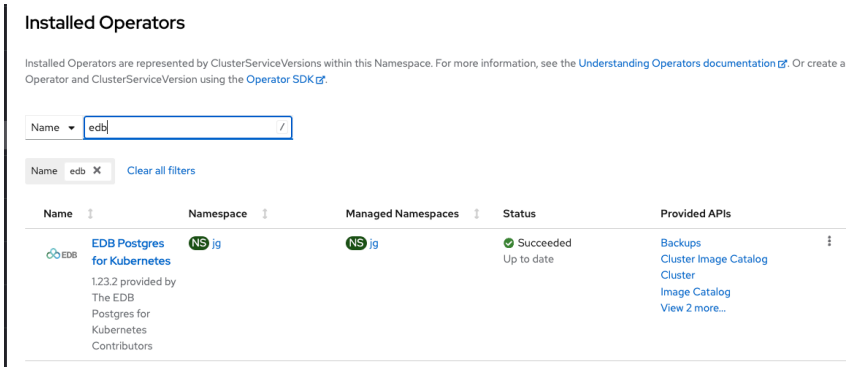


図 7: Multi namespace installation displayed in the web console

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:
  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。

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。

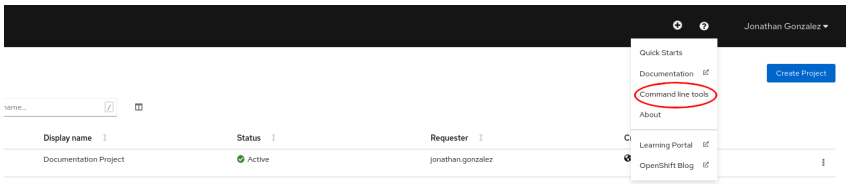


図 8: Command Line Tools menu

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。

> **参考**
>
> 詳細および更新情報については、公式 *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。

5.3.6 事前定義された RBAC オブジェクト

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担うリソースの事前定義されたセットが付属しています。

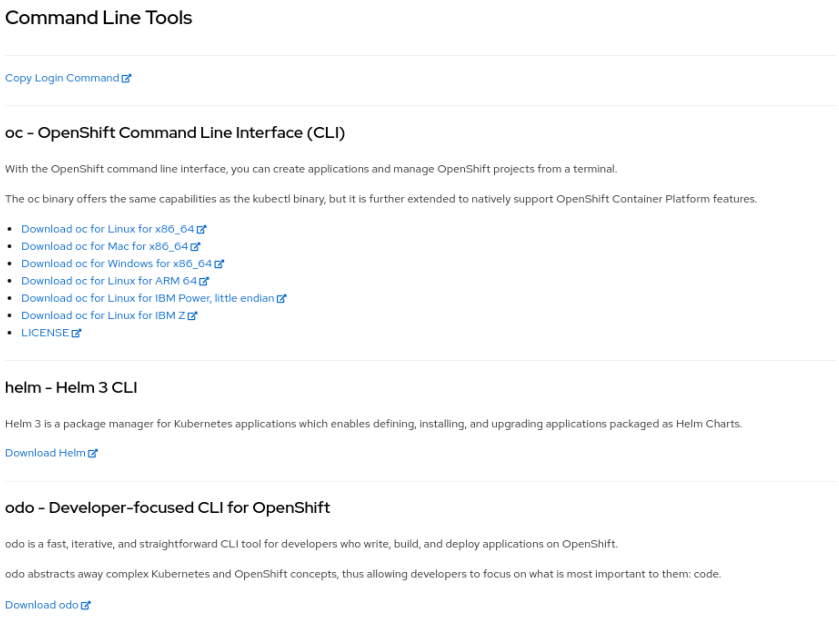


図 9: Command Line Tools

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`
- `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
```

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。

> **重要**
>
> Kubernetes のサービスアカウントは名前空間リソースです。明示的に許可されない限り、サービスアカウントは、定義された名前空間の外部でアクセスできません。

これは、次を実行して確認できます。

```
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。

> **重要**
>
> *postgresql-operator-manager* サービスアカウントは、オペレーターの作業を停止する可能性があるため削除しないでください。

クラスターロール

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、EDB Postgres ® AI for CloudNativePG™クラスター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。

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、顧客対応ユーザーと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

重要

クラスターロール自分自身はセキュリティ上の脅威ではありません。これらは、OpenShift で、特定のプロジェクトまたはグローバルで実際のユーザーに後で「バインド」されるロールのテンプレートを定義するための推奨される方法です。実際、クラスターロールは、グローバル権限の `ClusterRoleBinding` オブジェクトと組み合わせて、またはローカル権限の `RoleBinding` オブジェクトと組み合わせて使用できます。これにより、複数のプロジェクトでクラスターロールを再利用できると同時に、ローカルロールを介して個々のプロジェクト内のカスタマイズが可能になります。

次を実行して、事前定義されたクラスターロールのリストを確認できます。

```
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
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
```

(次のページに続く)

(前のページからの続き)

```
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:
      - "*"
```

参考

OperatorGroup による RBAC の実際の実装に興味がある場合は、 OperatorGroup: RBAC を参照してください。

Kubernetes CRD のクラスターロール

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]
```

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* 、および `ScheduledBackup` 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。 Cluster roles on EDB Postgres ® AI for CloudNativePG™ Cluster CRDs を参照してください。

オペレーターは、PostgreSQL 管理に関連する多くの操作を宣言的な方法で自動化します。それ以外の場合は、手動で命令的介入を必要とします。このような操作には、RBAC たとえばサービスアカウント、ポッ

ドたとえばセキュリティコンテキスト制約など、および Postgres レベルたとえば TLS 証明書などのセキュリティ関連事項も含まれます。

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。

5.3.7 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:
```

(次のページに続く)

(前のページからの続き)

```
      - 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
  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:
      - "*"
```

5.3.8 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 と連携します。

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。

5.3.9 プーラーイメージのカスタマイズ

デフォルトでは、 `Pooler` リソースは、`quay.io/enterprisedb/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。

高度なインターフェイスからイメージ名を変更できます。具体的には、_“Template”**セクションを開き、**”Spec > Containers”**の下で**”Add Container”_を選択します。

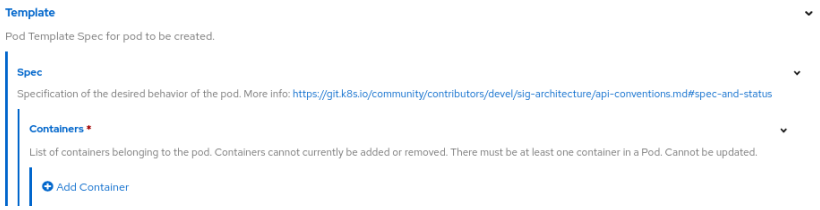


図 10: Pooler template

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

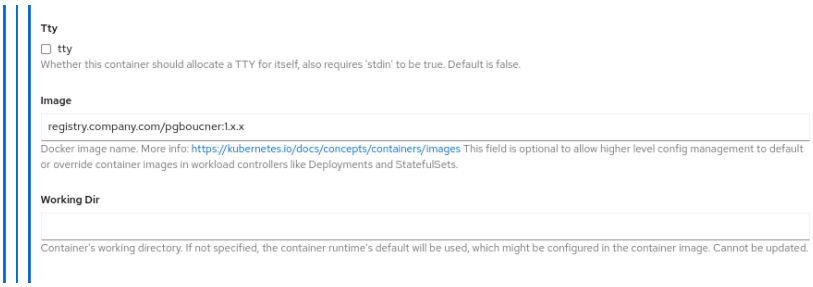


図 11: Pooler image

5.3.10 Velero の OADP

EDB Postgres ® AI for CloudNativePG™ クラスターオペレーターは、Velero の管理に *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 と統合され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。

5.3.11 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
- --
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。

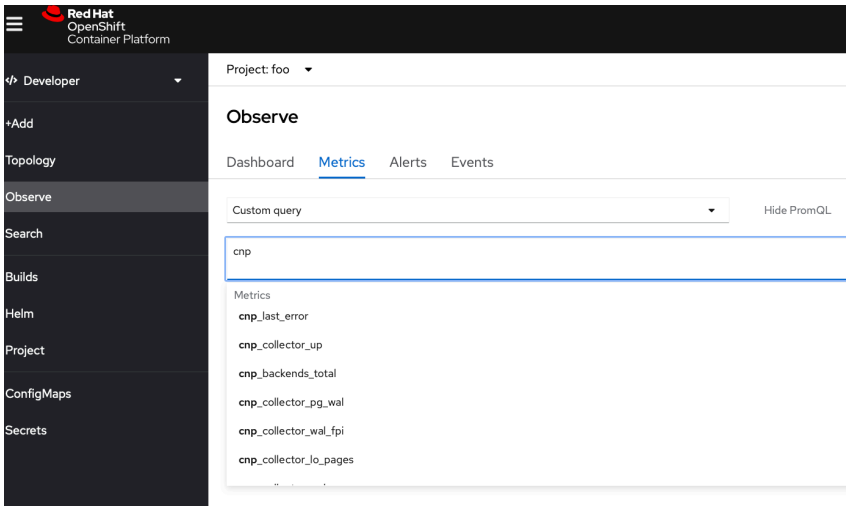


図 12: Prometheus queries

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。

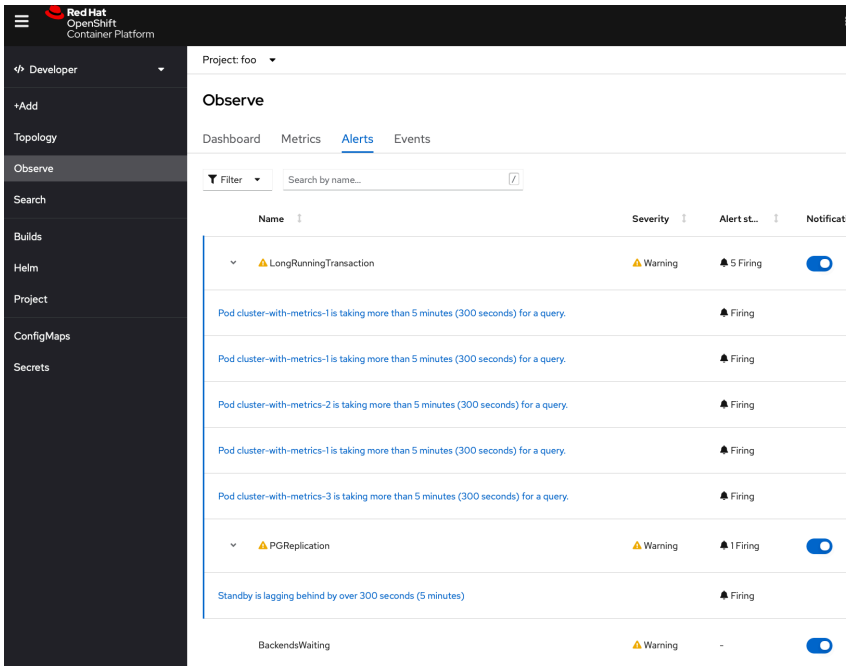


図 13: Prometheus alerts

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。

5.4 Transparent Data Encryption (TDE)

重要

TDE は、それをサポートするオペランドでのみ使用できます。EPAS および PG Extended、バージョン 15 以降。

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、EDB Postgres ® AI for CloudNativePG™ Cluster オペレーターによってサポートされています。

重要

続行する前に、時間をかけて *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

5.4.1 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。

参考

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。

5.4.2 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。

5.4.3 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`

5.4.4 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
```

5.5 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。

5.5.1 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。

> **重要**
>
> 外部バックアップアダプターは、バックアップを実行するツールではありません。これは、Kubernetes スペースのサードパーティバックアップツールが使用できる汎用インターフェイスを提供するだけです。このようなツールは、PVC および/またはコンテンツを安全に保存し、復旧時に Kubernetes クラスターに必要なすべてのリソース定義とともに利用できるようにします。

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" のいずれか

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 バックアップ前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
```

(次のページに続く)

(前のページからの続き)

```
data:
  # ...
  EXTERNAL_BACKUP_ADDON_CONFIGURATION: |-
    electedResourcesDecorators:
      - 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"
```

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:
```

(次のページに続く)

(前のページからの続き)

```
  name: cluster-example
  annotations:
    "k8s.enterprisedb.io/addons": ["external-backup-adapter-cluster"]
    "k8s.enterprisedb.io/externalBackupAdapterClusterConfig": |-
      electedResourcesDecorators:
        - 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

> **ヒント**
>
> 以下のコンテンツをコピーし、 *Cluster* で演算子またはアノテーションを構成するために使用する `ConfigMap` または `Secret` の中に貼り付けます。必ず、 YAML reserves for literals の `|` 文字と適切なインデントを使用します。コメントを使用して、ツールのオプションをカスタマイズします。

```yaml
#  An array of labels and/or annotations that will be placed

#  on the elected PVC group

electedResourcesDecorators:
  - key: "backup.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"

#  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"
    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
```

(次のページに続く)

(前のページからの続き)

```
#  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.
  # 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:
```

(次のページに続く)

(前のページからの続き)

```
    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"
```

5.5.2 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 および Kasten by Veeam ImplementationGuide 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

```
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。

重要

クラスターに `k8s.enterprisedb.io/snapshotAllowColdBackupOnPrimary: enabled` のアノテーションが付けられていない限り、オペレーターはプライマリインスタンスをシャットダウンしてコールドバックアップを取ることを拒否します

Kasten の構成と使用方法の詳細については、実装ガイドの *Instance pod configuration* および *Iron Bank* イメージを使用した*{{name.abbr}}*オペレーターのインストール セクション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 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。

5.5.3 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。

重要

クラスターに `k8s.enterprisedb.io/snapshotAllowColdBackupOnPrimary: enabled` のアノテーションが付けられていない限り、オペレーターはプライマリインスタンスをシャットダウンしてコールドバックアップを取得することを拒否します。

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。

重要

デフォルトでは、Velero アドオンはバックアップオペレーションからいくつかのリソース、つまり選択されていないインスタンスのポッドと PVC のみを除外します。ご存知のとおり、オペレーターは最初のレプリカの PVC をバックアップしようとします。ただし、 *velero backup* コマンドのオプションを使用して、バックアップの一部にするリソースを微調整できます。

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6章　Reference

6.1 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。

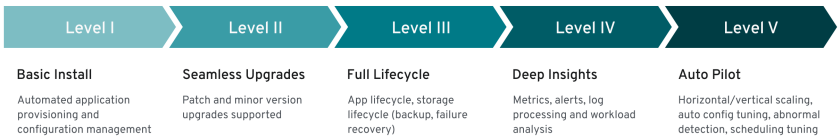


図 1: Operator Capability Levels

重要

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

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。

6.1.1 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。

重要

情報セキュリティはこのレベルの一部であると考えられます。

宣言的構成を介した Operator の展開

オペレーターは、4 つの主要な `CustomResourceDefinition` オブジェクト `Cluster` 、`Pooler` 、`Backup` 、および `ScheduledBackup` を定義する Kubernetes マニフェストを使用して宣言的方法でインストールされます。

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。オペランドを展開するときに、オペレーターは次のリソースも作成します。`Pod` 、`Service` 、`Secret` 、`ConfigMap` 、`PersistentVolumeClaim` 、`PodDisruptionBudget` 、`ServiceAccount` 、`RoleBinding` 、および `Role` 。

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、内部に PostgreSQL を使用したオペランドコンテナイメージをサポートするように設計されています。デフォルトでは、オペレーターは、PostgreSQL コミュニティでサポートされ、EDB によって quay.io で公開された最新の安定したメジャーバージョンの最新の利用可能なマイナーバージョンを使用します。CR で `imageName` 属性を設定することにより、プライマリ/スタンバイアーキテクチャをサポートする PostgreSQL の互換イメージを直接使用できます。また、オペレーターは、プライベートコンテナレジストリにアクセスするための `imagePullSecrets` をサポートし、コンテナイメージの不変性をより詳細に制御するためのダイジェストとタグをサポートしています。イメージ名を指定したくない場合は、PostgreSQL メジャーバージョンを参照するだけで *image catalogs* を活用できます。さらに、イメージカタログを使用すると、カスタムカタログを簡単に作成し、特定の要件に基づいて画像を指定できます。

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 の組織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 **デフォルトサービスの無効化** 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。

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。また、拡張機能とスキーマ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 *management of PostgreSQL roles, users, and groups through declarative configuration* をサポート

.spec.managed.roles スタンザを使用します。

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が目的です。

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。

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* コンテナイメージ を使用したクラスターのインストールをサポート

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。

6.1.2 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

CR、特に `imageName` パラメーターの変更の一部として、宣言的構成アプローチを使用してオペランドをアップグレードできます。これは通常、セキュリティ更新または Postgres のマイナーバージョンの更新によって開始されます。スタンバイサーバーが存在する場合、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。これは、既存のポッドを削除し、基になるストレージを再利用する新しい要求されたオペランドイメージで新しいポッドを作成することにより実行されます。`primaryUpdateStrategy` の値に応じて、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続行します。または、ユーザーが kubectl の `cnp` プラグインを介してスイッチオーバープロシージャー `supervised` を手動で発行するのを待機します。操作はアプリケーションにダウンタイムを生成する可能性があるため、使用する設定はビジネス要件によって異なります。このダウンタイムは、実際のデータベースワークロードに基づいて、数秒から数分の範囲です。

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クラスターに適用される場合、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。アップグレードは、`.spec.imageName` オプションを介してイメージタグを更新することにより、またはイメージカタログを使用してバージョンの変更を管理することによりトリガーできます。アップグレード中に、すべてのクラスターポッドがシャットダウンして、データの一貫性を保証します。次に、アップグレード条件を検証し、`pg_upgrade` を実行し、必要に応じて `PGDATA` の新しいディレクトリ、WAL ファイル、およびテーブルスペースを作成するための新しいジョブが作成されます。アップグレードが完了すると、レプリカが再作成されます。失敗したアップグレードはロールバックできます。

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。

6.1.3 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 CNP-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。

CNP-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 CNP-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

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のサポートを使用して、CloudNativePG™クラスタークラスターのローカル高可用性 EDB Postgres ® AI を達成します。オペレーターは、いつでも使用可能な予想同期スタンバイレプリカの数を定義する柔軟な方法を提供し、必要に応じて synchronous_standby_names オプションをカスタマイズできます。

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目標復旧時間 **バックアップ頻度と** *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、多様な環境にわたって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、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向上させ、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。

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。ハイバネーションでは、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。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。

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。

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、オンラインデータの移行パブリック DBaaS ソリューションからでもおよびメジャー PostgreSQL のアップグレードの場合のインポート機能と特に役立ちます。

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。

6.1.4 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に受け入れ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モニタし、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
- pg_failover_slots 物理フェールオーバーにわたって論理レプリケーションスロットを使用できるようにし、PostgreSQL のネイティブ論理レプリケーションに基づいて変更データキャプチャ CDC コンテキストの復元力を保証します

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。

6.1.5 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。

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。

6.2 Frequently Asked Questions (FAQ)

6.2.1 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™クラスターオペレーターの 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、完全な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 13 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 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。

6.2.2 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?

CloudNativePG™クラスターオペレーターの EDB Postgres ® AI は、特に、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ステータスを保持することにしました。Postgres 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。

6.2.3 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* コンテナイメージ 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:
```

(次のページに続く)

(前のページからの続き)

```
  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 ]---------
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?

アーキテクチャ 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 :ref:``cnp` プラグインの宣言的ハイバネーション <*cnp* プラグインの宣言的ハイバネーション>` 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 **スーパーユーザー**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、*Declarative role management* 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。

6.3 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提供します。

> **ヒント**
>
> Kubernetes 管理者は、 kubectl ページをブックマークする必要があります。

6.3.1 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 :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>`

- 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。

6.3.2 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。

6.3.3 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。

6.3.4 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。

> **重要**
>
> 以下にある指示は、緊急の場合、および一時的なバックアップファイルは、組織で有効なデータ保護ポリシーの下で保存される場合にのみ実行する必要があります。ダンプファイルは実際に `kubectl` コマンドを実行するクライアントマシンに保存されているため、すべての保護が設定されており、バックアップファイルを保存するための十分な領域があることを確認してください。

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
```

重要

このセクションの例では、推奨に従って、ダンプとリストアの他のグローバルオブジェクトデータベースとロールがないことを前提としています。複数のロールがある場合は、*pg_dumpall -g* を使用してバックアップをとり、新しいクラスターに手動で復元していることを確認します。複数のデータベースがある場合、一度に 1 つのデータベースごとに上記の操作を繰り返し、適切な所有権を割り当てていることを確認する必要があります。 PostgreSQL に慣れていない場合は、専門のサポート会社の指導の下でこれらの重要な操作を実行することをお勧めします。

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。

6.3.5 ログ

CloudNativePG™クラスターログの EDB Postgres によって作成および管理されてい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 には *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。

6.3.6 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デプロイメントの詳細 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の EDB Postgres ® AI のすべてのポッドからログを取得しますマルチオペレーター展開がある場合。

```
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:   quay.io/enterprisedb/postgresql:17.5-3
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 ␣
↪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
```

6.3.7 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 **準備** 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:    quay.io/enterprisedb/postgresql:17.5-3
```

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。

6.3.8 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>
```

(次のページに続く)

(前のページからの続き)

```
#  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
```

(次のページに続く)

(前のページからの続き)

```
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
```

6.3.9 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>
```

6.3.10 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。

6.3.11 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。

6.3.12 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
```

(次のページに続く)

(前のページからの続き)

```
    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
```

6.3.13 ネットワーキング

CloudNativePG™クラスターの EDB Postgres ® AI を使用するには、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との間の必要な接続に影響を与える可能性があります。/またはポッドの間。

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
allow-prometheus       k8s.enterprisedb.io/cluster=cluster-example   47m
default-deny-ingress   <none>                                57m
```

6.3.14 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内容を制御できます。

> **注釈**
>
> ラベルとアノテーション を参照してください。

EDB Postgres ® AI for CloudNativePG™クラスター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、

> **重要**
>
> この設定はポッドの起動中にのみ有効になり、アノテーションを変更してもインスタンスの自動ロールアウトはトリガーされないことに注意してください。

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。

6.3.15 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 で指摘されているように、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。

6.4 API Reference

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

6.4.1 Resource Types

- *Appendix C - Common object stores for backups*

6.4.2 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
apiVersion **[Required]** string	*postgresql.k8s.enterprisedb.io/v1*
kind **[Required]** string	*Backup*
metadata **[Required]** _meta/v1.ObjectMeta_	説明はありません。`metadata`フィールド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 **[Required]** *_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
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

6.4.3 Cluster

クラスターは PostgreSQL API のスキーマです

Field	Description
apiVersion **[Required]** string	*postgresql.k8s.enterprisedb.io/v1*
kind **[Required]** string	*Cluster*
metadata **[Required]** _meta/v1.ObjectMeta_	説明はありません。`metadata`フィールド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 2 – 前のページからの続き

Field	Description
spec **[Required]** *_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
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

6.4.4 ClusterImageCatalog

ClusterImageCatalog は、 clusterimagecatalogs API のスキーマです

Field	Description
apiVersion **[Required]** string	*postgresql.k8s.enterprisedb.io/v1*
kind **[Required]** string	*ClusterImageCatalog*
metadata **[Required]** *_meta/v1.ObjectMeta_*	説明はありません。`metadata`フィールド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 **[Required]** *_ImageCatalogSpec_*	ClusterImageCatalog 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

6.4.5 Database

データベースはデータベース API のスキーマです

Field	Description
apiVersion **[Required]** string	*postgresql.k8s.enterprisedb.io/v1*
kind **[Required]** string	*Database*
metadata **[Required]** *_meta/v1.ObjectMeta_*	説明はありません。`metadata`フィールド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 **[Required]** *_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
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

6.4.6 FailoverQuorum

利用箇所

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
apiVersion **[Required]** string	*postgresql.k8s.enterprisedb.io/v1*
kind **[Required]** string	*FailoverQuorum*
metadata **[Required]** _meta/v1.ObjectMeta_	説明はありません。`metadata`フィールド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。
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。

6.4.7 ImageCatalog

ImageCatalog は、imagecatalogs API のスキーマです

Field	Description
apiVersion **[Required]** string	*postgresql.k8s.enterprisedb.io/v1*
kind **[Required]** string	*ImageCatalog*
metadata **[Required]** _meta/v1.ObjectMeta_	説明はありません。`metadata`フィールド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 **[Required]** *_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

6.4.8 Pooler

Pooler は、プーラー API のスキーマです

Field	Description
apiVersion **[Required]** string	*postgresql.k8s.enterprisedb.io/v1*
kind **[Required]** string	*Pooler*
metadata **[Required]** _meta/v1.ObjectMeta_	説明はありません。`metadata`フィールド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 **[Required]** *_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

次のページに続く

表 7 – 前のページからの続き

Field	Description
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

6.4.9 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
apiVersion **[Required]** string	*postgresql.k8s.enterprisedb.io/v1*
kind **[Required]** string	*Publication*
metadata **[Required]** *_meta/v1.ObjectMeta_*	説明はありません。`metadata`フィールド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 **[Required]** *_PublicationSpec_*	説明はありません。
status **[Required]** *_PublicationStatus_*	説明はありません。

6.4.10 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
apiVersion **[Required]** string	*postgresql.k8s.enterprisedb.io/v1*
kind **[Required]** string	*ScheduledBackup*
metadata **[Required]** *_meta/v1.ObjectMeta_*	説明はありません。`metadata`フィールド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 **[Required]** *_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
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

6.4.11 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
apiVersion **[Required]** string	*postgresql.k8s.enterprisedb.io/v1*
kind **[Required]** string	*Subscription*
metadata **[Required]** _meta/v1.ObjectMeta_	説明はありません。`metadata`フィールド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 **[Required]** *_SubscriptionSpec_*	説明はありません。
status **[Required]** *_SubscriptionStatus_*	説明はありません。

6.4.12 AffinityConfiguration

利用箇所

- *ClusterSpec*

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

Field	Description
enablePodAntiAffinity _bool_	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の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。その詳細については、k8s ドキュメントを参照してください
nodeSelector _map[string]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ペアのマップです。詳細 https://kubernetes.io/docs/concepts/configuration/assign-pod-node/
nodeAffinity _core/v1.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を説明します。詳細 https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#node-affinity
tolerations [_[]core/**v1.Toleration**_](https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#toleration-v1-core)	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必要がある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、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、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。詳細 https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#inter-pod-affinity-and-anti-affinity

次のページに続く

表 11 – 前のページからの続き

Field	Description
additionalPodAntiAffinity _core/v1.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が生成した用語に追加するポッドアンチアフィニティ用語、false に設定されている場合に排他的に使用されるポッドアンチアフィニティ用語を指定できます。
additionalPodAffinity _core/v1.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。

6.4.13 AvailableArchitecture

利用箇所

- *ClusterStatus*

AvailableArchitecture は、クラスターのアーキテクチャの状態を表します

Field	Description
goArch **[Required]** _string_	GoArch は実行可能アーキテクチャの名前です
hash **[Required]** _string_	ハッシュは実行可能ファイルのハッシュです

6.4.14 BackupConfiguration

利用箇所

- *ClusterSpec*

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。

Field	Description
volumeSnapshot *_VolumeSnapshotConfiguration_*	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 _github.com/cloudnative-pg/barman-cloud/pkg/api.BarmanObjectStoreConfig	barman-cloud ツールスイートの構成
retentionPolicy _string_	RetentionPolicy は、バックアップと 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`の形式で表されます。ここで、 *XX* `は正の整数、`*u*`は `*[dwm]*-日、週、月です。現在、BarmanObjectStore メソッドを使用する場合にのみ適用されます。

次のページに続く

表 13 – 前のページからの続き

Field	Description
*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、バックアップを常にプライマリインスタンスで実行する `primary`、可能な場合は最も更新されたスタンバイでバックアップを実行する `prefer-standby`です。

6.4.15 BackupMethod

`string` のエイリアス

利用箇所

- *BackupSpec*

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

6.4.16 BackupPhase

`string` のエイリアス

利用箇所

- *BackupStatus*

BackupPhase はバックアップのフェーズです

6.4.17 BackupPluginConfiguration

利用箇所

- *BackupSpec*

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

Field	Description
name **[Required]** _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
parameters _map[string]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。

6.4.18 BackupSnapshotElementStatus

利用箇所

- *BackupSnapshotStatus*

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

Field	Description
name **[Required]** _string_	Name はスナップショットリソース名です
type **[Required]** _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
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

6.4.19 BackupSnapshotStatus

利用箇所

- *BackupStatus*

BackupSnapshotStatus volumeSnapshot 方法のバックアップ専用のフィールド

Field	Description
elements [_[]BackupSnapshotElementStatus_](BackupSnapshotElementStatus)	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

6.4.20 BackupSource

利用箇所

- *BootstrapRecovery*

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。

Field	Description
LocalObjectReference _github.com/cloudnative-pg/machinery/pkg/api.LocalObjectRefere	`LocalObjectReference`のメンバーはこのタイプに埋め込まれます。説明はありません。
endpointCA _github.com/cloudnative-pg/machinery/pkg/api.SecretKeySelector	EndpointCA は、バーマンエンドポイントの CA バンドルを保存します。自己署名証明書を使用して、証明書の発行者と barman-cloud-wal-archive でのエラーを回避する場合に役立ちます。

6.4.21 BackupSpec

利用箇所

- *Appendix C - Common object stores for backups*

BackupSpec は、バックアップの望ましい状態を定義します

Field	Description
cluster **[Required]** _github.com/cloudnative-pg/machinery/pkg/api.LocalObjectRefere	バックアップするクラスター
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`になります。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バックアップを実行します。`prefer-standby`は、可能な場合は最も更新されたスタンバイでバックアップを実行します。
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*
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 _bool_	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* かどうか クラスターフィールド '.spec.backup.volumeSnapshot.online' で指定されたデフォルト設定をオーバーライドします
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 クラスターで指定されたデフォルト設定をオーバーライドします '.backup.volumeSnapshot.onlineConfiguration' スタンザ

6.4.22 BackupStatus

利用箇所

- *Appendix C - Common object stores for backups*

BackupStatus は、バックアップの監視された状態を定義します

Field	Description
BarmanCredentials _github.com/cloudnative-pg/barman-cloud/pkg/api.BarmanCredentials_	`BarmanCredentials`のメンバーはこのタイプに埋め込まれます。各クラウドプロバイダーの潜在的な資格情報
endpointCA _github.com/cloudnative-pg/machinery/pkg/api.SecretKeySelector	EndpointCA は、バーマンエンドポイントの CA バンドルを保存します。自己署名証明書を使用して、証明書の発行者と 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する
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 と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 _meta/v1.Time_	バックアップが開始されたとき
stoppedAt _meta/v1.Time_	バックアップが終了したとき
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
backupLabelFile _[]byte_	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 _[]byte_	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 _bool_	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 _map[string]string_	プラグインのメタデータを含むマップ

6.4.23 BackupTarget

`string` のエイリアス

利用箇所

- *BackupConfiguration*

BackupTarget は、バックアップの優先ターゲットを説明します

6.4.24 BootstrapConfiguration

利用箇所

- *ClusterSpec*

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。

Field	Description
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_*	互換性のある別の 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

6.4.25 BootstrapInitDB

利用箇所

- *BootstrapConfiguration*

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。

Field	Description
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内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 _github.com/cloudnative-pg/machinery/pkg/api.LocalObjectRefere	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は最初から作成されます
redwood _bool_	Redwood の互換性を有効/無効にする必要がある場合。 EPAS が必要で、EPAS の場合はデフォルトで true
options _[]string_	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、一貫性のない構成を引き起こす可能性があります。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。

次のページに続く

表 21 – 前のページからの続き

Field	Description
dataChecksums _bool_	*-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_	追加の照合ルールを指定して、デフォルトの照合順序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。
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。
walSegmentSize _int_	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
postInitSQL _[]string_	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_	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
postInitTemplateSQL _[]string_	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 *_Import_*	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
postInitApplicationSQLRefs *_SQLRefs_*	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 ファイルを含む ConfigMap または Secret への参照のリスト。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。デフォルトでは空

次のページに続く

表 21 – 前のページからの続き

Field	Description
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 ConfigMap または Secret への参照のリスト。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。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 ConfigMap または Secret への参照のリスト。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。デフォルトでは空

6.4.26 BootstrapPgBaseBackup

利用箇所

- *BootstrapConfiguration*

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

Field	Description
source **[Required]** _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
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内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 _github.com/cloudnative-pg/machinery/pkg/api.LocalObjectRefere	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は最初から作成されます

6.4.27 BootstrapRecovery

利用箇所

- *BootstrapConfiguration*

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。

Field	Description
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。現在、既存の PVC グループをマッピングする `VolumeSnapshot`および `PersistentVolumeClaim`リソースをサポートし、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撮影されます将来的にオンラインバックアップが行われるときに削除される制限実装されました)。 `backup`と相互排他的。
recoveryTarget _RecoveryTarget_ by specifying a *RecoveryTarget* object, as expected by PostgreSQL (i.e., timestamp, transaction Id, LSN, ...). More info: https://www.postgresql.org/docs/current/runtime-config-wal.html#RUNTIME-CONFIG-WAL-RECOVERY	NA
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内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 _github.com/cloudnative-pg/machinery/pkg/api.LocalObjectRefere	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は最初から作成されます

6.4.28 CatalogImage

利用箇所

- *ImageCatalogSpec*

CatalogImage はイメージとメジャーバージョンを定義します

Field	Description
image **[Required]** _string_	画像参照
major **[Required]** _int_	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。

6.4.29 CertificatesConfiguration

利用箇所

- *CertificatesStatus*

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。

Field	Description
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は自己署名 CA で作成され、TLS 証明書 ServerTLSSecret の生成に使用されます。以下が含まれます *`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`* **として使用されます。** *ca.key* サーバー SSL 証明書を生成するために使用されるキー。ServerTLSSecret が提供されている場合、これは省略できます。
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`として設定されます。定義されていない場合、ServerCASecret は`ca.key`も提供する必要があり、提供された CA を使用して新しいシークレットが作成されます。
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は、提供された CA を使用して作成されます。
clientCASecret _string_	クライアント CA 証明書を含むシークレット。定義しない場合、自己署名 CA で新しいシークレットが作成され、すべてのクライアント証明書を生成するために使用されます。以下が含まれます *`ca.crt`* **クライアントを検証するために使用する** *CA* **証明書。すべてのインスタンスの** *`ssl_ca_file`* **として使用されます。** *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、これは省略できます。
serverAltDNSNames _[]string_	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。

6.4.30 CertificatesStatus

利用箇所

- *ClusterStatus*

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。

Field	Description
CertificatesConfiguration _*CertificatesConfiguration*_	`CertificatesConfiguration`のメンバーはこのタイプに埋め込まれています。サーバー証明書を処理するために必要な構成、必要に応じてデフォルト値で初期化されます。
expirations _map[string]string_	すべての証明書の有効期限。

6.4.31 ClusterMonitoringTLSConfiguration

利用箇所

- *MonitoringConfiguration*

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

Field	Description
enabled _bool_	監視エンドポイントの 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のロールアウトが強制されます。

6.4.32 ClusterSpec

利用箇所

- レプリカクラスター

ClusterSpec は、クラスターの望ましい状態を定義します

Field	Description
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_	コンテナイメージの名前、タグ *<image>:<tag>* と決定的で反復可能な展開用のダイジェスト *<image>:<tag>@sha256:<digestValue>* の両方をサポート

次のページに続く

表 28 – 前のページからの続き

Field	Description
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 _core/v1.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`が使用されます。更新できません。詳細 https://kubernetes.io/docs/concepts/containers/images#updating-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 https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/
postgresUID _int64_	画像内の `postgres`ユーザーの UID、デフォルトは `26`
postgresGID _int64_	画像内の `postgres`ユーザーの GID、デフォルトは `26`
instances **[Required]** _int_	クラスターで必要なインスタンスの数
minSyncReplicas _int_	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。未定義または 0 では、使用可能なスタンバイがない場合に書き込みを完了できます。
maxSyncReplicas _int_	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も少ない場合、削減できます。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。
postgresql *_PostgresConfiguration_*	PostgreSQL サーバーの構成
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 _github.com/cloudnative-pg/machinery/pkg/api.LocalObjectRefere	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 _bool_	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 [_[]github.com/cloudnative-pg/machinery/pkg/**api.LocalObjectRefe** //pkg.go.dev/github.com/cloudnative-pg/machinery/pkg/api/#LocalObjectReference)	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。ライセンスキーに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

次のページに続く

表 28 – 前のページからの続き

Field	Description
serviceAccountTemplate *_ServiceAccountTemplate_*	サービスアカウントの世代を構成する
walStorage _StorageConfiguration_	NA
ephemeralVolumeSource _core/v1.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 _int32_	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。起動プローブ障害しきい値は、次の式 ceiling(startDelay / 10) を使用してこの値から導き出されます。
stopDelay _int32_	PostgreSQL インスタンスが正常にシャットダウンできるまでの秒単位の時間デフォルト 1800
smartStopDelay _int32_	非推奨: 代わりに SmartShutdownTimeout を使用してください
smartShutdownTimeout _int32_	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。オペレーターが 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確保してください *stopDelay- smartShutdownTimeout* 。
switchoverDelay _int32_	スイッチオーバー中にプライマリ 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。デフォルト値は 3600 秒 1 時間です。
failoverDelay _int32_	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待機する時間秒
livenessProbeTimeout _int32_	LivenessProbeTimeout は、PostgreSQL インスタンスが liveness プローブデフォルト 30 に正常に応答するまでに許可される時間秒単位です。 Liveness プローブの障害しきい値は、次の式を使用してこの値から導き出されます。ceiling(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 [_[]core/**v1.TopologySpreadConstraint** //kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#topologyspreadconstraint-v1-core)	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 https://kubernetes.io/docs/concepts/scheduling-eviction/topology-spread-constraints/
resources _core/v1.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、ユーザーは一時ボリュームの制限を設定できます
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/#prioritycla ss を参照してください。

次のページに続く

表 28 – 前のページからの続き

Field	Description
primaryUpdateStrategy _PrimaryUpdateStrategy_ or manual (*supervised*)	NA
primaryUpdateMethod _PrimaryUpdateMethod_	NA
backup *_BackupConfiguration_*	バックアップに使用される構成
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、クラスターは試用モードで動作し、有効期限デフォルト 30 日後、オペレーターは調整の試行を停止します。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 _core/v1.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_](ExternalCluster)	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
projectedVolumeTemplate _core/v1.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 [_]core/**v1.EnvVar_**](https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#envvar-v1-core)	Env は、 Env 形式に従って、クラスター内に作成されたポッドに環境変数を渡します
envFrom [_]core/**v1.EnvFromSource_**](https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#envfromsource-v1-core)	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
seccompProfile _core/v1.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`
tablespaces [_]TablespaceConfiguration_](TablespaceConfiguration)	テーブルスペース構成

次のページに続く

表 28 – 前のページからの続き

Field	Description
enablePDB _bool_	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安全に保護されます。逆に、`false`に設定すると、`PodDisruptionBudget`リソースが存在せず、PostgreSQL クラスターをホスト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。
plugins [_[]PluginConfiguration_](PluginConfiguration)	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 ポッドに挿入されるプローブの構成。

6.4.33 ClusterStatus

利用箇所

- レプリカクラスター

ClusterStatus は、クラスターの観察された状態を定義します

Field	Description
instances _int_	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 _int_	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。
instancesStatus _map[PodStatus][]string_	InstancesStatus は、インスタンスがどのステータスを示します
instancesReportedState [_map[PodName]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_](TablespaceState)	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
timelineID _int_	Postgres クラスターのタイムライン
topology *_Topology_*	インスタンスのトポロジ。
latestGeneratedNode _int_	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
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、スイッチオーバーまたはフェイルオーバー中の以前のインスタンスとは異なります。

次のページに続く

表 29 – 前のページからの続き

Field	Description
lastPromotionToken _string_	LastPromotionToken は、レプリカクラスターをプロモート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 _int32_	このクラスターによって作成された PVC の数
jobCount _int32_	このクラスターによって作成されたジョブの数
danglingPVC _[]string_	このクラスターによって作成され、ポッドに接続されていないまだ使用可能なすべての PVC のリスト
resizingPVC _[]string_	ResizingPVC 条件を持つすべての PVC のリスト。
initializingPVC _[]string_	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_	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_	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 _github.com/EnterpriseDB/cloud-native-postgres/pkg/**licensekey.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
secretsResourceVersion *_SecretsResourceVersion_*	オペレーターによって管理されているシークレットのリソースバージョンのリスト。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_*	オペレーターによって管理される、構成マップのリソースバージョンのリスト。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 [_map[BackupMethod]meta/**v1.Time**_](//kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#time-v1-meta)	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。

次のページに続く

表 29 – 前のページからの続き

Field	Description
lastSuccessfulBackupByMethod [_map[BackupMethod]meta/**v1.Time**_](https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#time-v1-meta)	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`が設定されたとき、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
availableArchitectures [_[]AvailableArchitecture_](AvailableArchitecture)	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 [_[]meta/**v1.Condition**_](https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#condition-v1-meta)	クラスターオブジェクトの条件
instanceNames _[]string_	クラスター内のインスタンス名のリスト
onlineUpdateEnabled _bool_	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。
pluginStatus [_[]PluginStatus_](PluginStatus)	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、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 TimeLineID、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などの pg_controldata からの情報を含む JSON トークン
systemID _string_	SystemID は、最後に検出された PostgreSQL のシステム ID です

6.4.34 ConfigMapResourceVersion

利用箇所

- *ClusterStatus*

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

Field	Description
metrics _map[string]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

6.4.35 DataDurabilityLevel

`string` のエイリアス

利用箇所

- *SynchronousReplicaConfiguration*

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。

6.4.36 DataSource

利用箇所

- *BootstrapRecovery*

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

Field	Description
storage **[Required]** _core/v1.TypedLocalObjectReference_	インスタンスのストレージの構成
walStorage _core/v1.TypedLocalObjectReference_	PostgreSQL WAL のストレージの構成書き込み先行ログ
tablespaceStorage [_map[string]core/**v1.TypedLocalObjec** //kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#typedlocalobjectreference-v1-core)	PostgreSQL テーブルスペースのストレージの構成

6.4.37 DatabaseObjectSpec

利用箇所

- *ExtensionSpec*

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

Field	Description
name **[Required]** _string_	拡張機能/スキーマの名前
ensure *EnsureOption*	拡張子/スキーマがデータベースに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、拡張機能/スキーマが存在しない場合は作成されます。 `absent`に設定している場合、拡張機能/スキーマが存在する場合は削除されます。

6.4.38 DatabaseObjectStatus

利用箇所

- *DatabaseStatus*

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

Field	Description
name **[Required]** _string_	オブジェクトの名前
applied **[Required]** _bool_	オブジェクトはデータベースに正常にインストールされました
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

6.4.39 DatabaseReclaimPolicy

`string` のエイリアス

利用箇所

- *DatabaseSpec*

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。

6.4.40 DatabaseRoleRef

利用箇所

- *TablespaceConfiguration*

DatabaseRoleRef は、PostgreSQL 内で使用可能なロールの参照です

Field	Description
name _string_	説明はありません。

6.4.41 DatabaseSpec

利用箇所

- *PostgreSQL Database Management*

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。

Field	Description
cluster **[Required]** _core/v1.LocalObjectReference_	データベースをホストする PostgreSQL クラスターの名前。
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。
name **[Required]** _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。
owner **[Required]** _string_	`CREATE DATABASE`の `OWNER`パラメーターにマッピングされます。 `ALTER DATABASE`の `OWNER TO`コマンドにマッピングされます。 PostgreSQL 内のデータベースを所有するユーザーのロール名。
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。
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 35 – 前のページからの続き

Field	Description
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。追加の照合ルールを指定して、デフォルトの照合順序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。
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。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。
isTemplate _bool_	`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`権限を持つすべてのユーザーがクローンを作成できます。
allowConnections _bool_	`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 _int_	`CREATE DATABASE`および `ALTER DATABASE`の `CONNECTION LIMIT`句にマップします。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ピングされます。 `ALTER DATABASE`の `SET TABLESPACE`コマンドにマッピングされます。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 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。
schemas [_[]SchemaSpec_](SchemaSpec)	データベースで管理するスキーマのリスト
extensions [_[]ExtensionSpec_](ExtensionSpec)	データベースで管理する拡張機能のリスト

6.4.42 DatabaseStatus

利用箇所

- *PostgreSQL Database Management*

DatabaseStatus は、データベースの観察された状態を定義します

Field	Description
observedGeneration _int64_	同期された最新の目的の状態を表すシーケンス番号
applied _bool_	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_](DatabaseObjectStatus)	Schemas は、管理対象スキーマのステータスです
extensions [_[]DatabaseObjectStatus_](DatabaseObjectStatus)	Extensions は、管理対象拡張機能のステータスです

6.4.43 EPASConfiguration

利用箇所

- *PostgresConfiguration*

EPASConfiguration には、EDB Postgres Advanced Server 固有の構成が含まれています

Field	Description
audit _bool_	true の場合 edb_audit ロギングを有効にします
tde *_TDEConfiguration_*	TDE 構成

6.4.44 EmbeddedObjectMetadata

利用箇所

- *ClusterSpec*

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

Field	Description
labels _map[string]string_	説明はありません。
annotations _map[string]string_	説明はありません。

6.4.45 EnsureOption

string のエイリアス

利用箇所

- *DatabaseObjectSpec*

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

6.4.46 EphemeralVolumesSizeLimitConfiguration

利用箇所

- *ClusterSpec*

Ephemeral VolumesSizeLimitConfiguration には、エフェメラルストレージの構成が含まれます

Field	Description
shm _k8s.io/apimachinery/pkg/api/resou	Shm は、共有メモリボリュームのサイズ制限です
temporaryData _k8s.io/apimachinery/pkg/api/resource.Q	TemporaryData は、一時データボリュームのサイズ制限です

6.4.47 ExtensionConfiguration

利用箇所

- *PostgresConfiguration*

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。

Field	Description
name **[Required]** _string_	拡張機能の名前、必須
image **[Required]** _core/v1.ImageVolumeSource_	拡張機能を含む画像必須
extension_control_path _[]string_	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」になります。
dynamic_library_path _[]string_	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_	ld_library_path に追加するイメージ内のディレクトリのリスト。

6.4.48 ExtensionSpec

利用箇所

- *DatabaseSpec*

ExtensionSpec はデータベースに拡張機能を構成します

Field	Description
DatabaseObjectSpec *_DatabaseObjectSpec_*	`DatabaseObjectSpec`のメンバーはこのタイプに埋め込まれます。共通フィールド
version **[Required]** _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
schema **[Required]** _string_	拡張機能がコンテンツを再配置できる場合、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。指定されない場合デフォルト、および拡張機能の制御ファイルも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。

6.4.49 ExternalCluster

利用箇所

- *ClusterSpec*

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

Field	Description
name **[Required]** _string_	サーバー名必須
connectionParameters _map[string]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 _core/v1.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 _core/v1.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 _core/v1.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
password _core/v1.SecretKeySelector_	サーバーへの接続に使用されるパスワードへの参照。パスワードが指定された場合、CloudNativePG™クラスターの EDB Postgres® AI は、*/controller/external/NAME/pass* で PostgreSQL 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です。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 _github.com/cloudnative-pg/barman-cloud/pkg/api.BarmanObjectStoreConfig	barman-cloud ツールスイートの構成
plugin **[Required]** *_PluginConfiguration_*	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

6.4.50 FailoverQuorumStatus

利用箇所

- *FailoverQuorum*

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。

Field	Description
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_	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 _int_	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。

6.4.51 ImageCatalogRef

利用箇所

- *ClusterSpec*

ImageCatalogRef は、ImageCatalog 内のメジャーバージョンへの参照を定義します

Field	Description
TypedLocalObjectReference _core/v1.TypedLocalObjectReference_	`TypedLocalObjectReference`のメンバーはこのタイプに埋め込まれます。説明はありません。
major **[Required]** _int_	ImageCatalog から使用する PostgreSQL のメジャーバージョン

6.4.52 ImageCatalogSpec

利用箇所

- *ClusterImageCatalog*

ImageCatalogSpec は、目的のイメージカタログを定義します

Field	Description
images **[Required]**[_[]CatalogImage_](CatalogIm	カタログで使用可能なカタログ画像のリスト

6.4.53 ImageInfo

利用箇所

- *ClusterStatus*

ImageInfo には、PostgreSQL イメージに関する情報が含まれています

Field	Description
image **[Required]** _string_	Image はイメージ名です
majorVersion **[Required]** _int_	MajorVersion はイメージのメジャーバージョンです

6.4.54 Import

利用箇所

- *BootstrapInitDB*

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

Field	Description
source **[Required]** *_ImportSource_*	輸入元
type **[Required]** *_SnapshotType_*	インポートタイプ。 `microservice`または `monolith`です。
databases **[Required]** _[]string_	インポートするデータベース
roles _[]string_	インポートするロール
postImportApplicationSQL _[]string_	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 クエリーのリスト。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 _bool_	true に設定されている場合、*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`*
pgDumpExtraOptions _[]string_	`pg_dump`コマンドに渡すカスタムオプションのリスト。重要オペレーターはコンテンツを検証していないため、これらのオプションは自分自身の責任で注意して使用します。特定のオプションは、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ることに注意してください。
pgRestoreExtraOptions _[]string_	`pg_restore`コマンドに渡すカスタムオプションのリスト。重要オペレーターはコンテンツを検証していないため、これらのオプションは自分自身の責任で注意して使用します。特定のオプションは、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ることに注意してください。

6.4.55 ImportSource

利用箇所

- *OpenShift の重要な概念*

ImportSource は、論理スナップショットのソースを説明します

Field	Description
externalCluster **[Required]** _string_	インポートに使用される外部クラスターの名前

6.4.56 InstanceID

利用箇所

- *BackupStatus*

InstanceID には、インスタンスを識別する情報が含まれます

Field	Description
podName _string_	ポッド名
ContainerID _string_	コンテナ ID

6.4.57 InstanceReportedState

利用箇所

- *ClusterStatus*

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

Field	Description
isPrimary **[Required]** _bool_	インスタンスがプライマリかどうかを示します
timeLineID _int_	インスタンスがどの TimelineId にあるかを示します
ip **[Required]** _string_	インスタンスの IP アドレス

6.4.58 IsolationCheckConfiguration

利用箇所

- *LivenessProbe*

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

Field	Description
enabled _bool_	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
requestTimeout _int_	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
connectionTimeout _int_	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

6.4.59 LDAPBindAsAuth

利用箇所

- *LDAPConfig*

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

Field	Description
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

6.4.60 LDAPBindSearchAuth

利用箇所

- *LDAPConfig*

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

Field	Description
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 _core/v1.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
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

6.4.61 LDAPConfig

利用箇所

- *PostgresConfiguration*

LDAPConfig には、LDAP 認証に必要なパラメーターが含まれています

Field	Description
server _string_	LDAP ホスト名または IP アドレス
port _int_	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
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 _bool_	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

6.4.62 LDAPScheme

`string` のエイリアス

利用箇所

- *LDAPConfig*

LDAPScheme は、 LDAP の使用可能なスキームを定義します

6.4.63 LivenessProbe

利用箇所

- *ProbesConfiguration*

LivenessProbe は liveness プローブの構成です

Field	Description
Probe *_Probe_*	`Probe`のメンバーはこのタイプに埋め込まれています。プローブは標準のプローブ構成です
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、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 の使用不可が検出された場合に安全にシャットダウンできることを保証します。デフォルトで有効になっています。

6.4.64 ManagedConfiguration

利用箇所

- *ClusterSpec*

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

Field	Description
roles [_[]RoleConfiguration_](RoleConfiguration)	`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

6.4.65 ManagedRoles

利用箇所

- *ClusterStatus*

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

Field	Description
byStatus _map[RoleStatus][]string_	ByStatus は、各状態のロールのリストを提供します
cannotReconcile _map[string][]string_	CannotReconcile は、PostgreSQL でリコンサイルできないロールと、その原因の説明をリストします
passwordStatus [_map[string]PasswordState_](Password	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

6.4.66 ManagedService

利用箇所

- *ManagedServices*

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。

Field	Description
selectorType **[Required]** *_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サービスを表します。
updateStrategy *_ServiceUpdateStrategy_*	UpdateStrategy は、サービスの違いを調整する方法を説明します
serviceTemplate **[Required]** *_ServiceTemplateSpec_*	ServiceTemplate は、サービスのテンプレート仕様です。

6.4.67 ManagedServices

利用箇所

- *ManagedConfiguration*

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。

Field	Description
disabledDefaultServices [_]ServiceSelectorType_](ServiceSelectorType)	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「r」および「ro」で、読み取りおよび読み取り専用サービスを表します。
additional [_]ManagedService_](ManagedService)	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。

6.4.68 Metadata

利用箇所

- *PodTemplateSpec*

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。

Field	Description
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 _map[string]string_	オブジェクトを整理および分類および選択するために使用できる文字列キーと値のマップ。レプリケーションコントローラと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 _map[string]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。これらは照会不可であり、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

6.4.69 MonitoringConfiguration

利用箇所

- *ClusterSpec*

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

Field	Description
disableDefaultQueries _bool_	デフォルトのクエリーを挿入するかどうか。クラスターにデフォルトのクエリーを挿入したくない場合は、`true`に設定します。デフォルト false。

次のページに続く

表 61 – 前のページからの続き

Field	Description
customQueriesConfigMap [_[]github.com/cloudnative-pg/machinery/pkg/**api.ConfigMapKeyS** //pkg.go.dev/github.com/cloudnative-pg/machinery/pkg/api/#ConfigMapKeySelector)	カスタムクエリを含む構成マップのリスト
customQueriesSecret [_[]github.com/cloudnative-pg/machinery/pkg/**api.SecretKeySelecto** //pkg.go.dev/github.com/cloudnative-pg/machinery/pkg/api/#SecretKeySelector)	カスタムクエリーを含むシークレットのリスト
enablePodMonitor _bool_	`PodMonitor`を有効または無効にします
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のロールアウトが強制されます。
podMonitorMetricRelabelings [_[]github.com/prometheus-operator/prometheus-operator/pkg/apis/monitoring/**v1.Relabel** //pkg.go.dev/github.com/prometheus-operator/prometheus-operator/pkg/apis/monitoring/v1#RelabelConfig)	`PodMonitor`のメトリックの再ラベル付けのリスト。摂取する前にサンプルに適用されます。
podMonitorRelabelings [_[]github.com/prometheus-operator/prometheus-operator/pkg/apis/monitoring/**v1.Relabel** //pkg.go.dev/github.com/prometheus-operator/prometheus-operator/pkg/apis/monitoring/v1#RelabelConfig)	`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。

6.4.70 NodeMaintenanceWindow

利用箇所

- *ClusterSpec*

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。

Field	Description
reusePVC _bool_	既存の PVC を再利用するノードが再び起動するまで待機するか、他の場所で再作成します- *instances* >1 の場合
inProgress _bool_	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?

6.4.71 OnlineConfiguration

利用箇所

- *BackupSpec*

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

Field	Description
waitForArchive _bool_	false の場合、ファンクションは、バックアップが完了すると、WAL がアーカイブされるのを待たずに、すぐに返されます。この動作は、WAL アーカイブを個別に監視するバックアップソフトウェアでのみ役立ちます。そうしないと、バックアップの一貫性を保つために必要な WAL が欠落し、バックアップが役に立たなくなる可能性があります。デフォルトでは、またはこのパラメータが true の場合、アーカイブが有効になっているときに pg_backup_stop は WAL がアーカイブされるまで待機します。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、即時セグメントスイッチをトリガーすると役立つ場合があります。
immediateCheckpoint _bool_	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します。デフォルトでは `false`。

6.4.72 PasswordState

利用箇所

- *ManagedRoles*

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

Field	Description
transactionID _int64_	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

次のページに続く

表 64 – 前のページからの続き

Field	Description
resourceVersion _string_	パスワードシークレットのリソースバージョン

6.4.73 PgBouncerIntegrationStatus

利用箇所

- *PoolerIntegrations*

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

Field	Description
secrets _[]string_	説明はありません。

6.4.74 PgBouncerPoolMode

`string` のエイリアス

利用箇所

- *PgBouncerSpec*

PgBouncerPoolMode は PgBouncer のモードです

6.4.75 PgBouncerSecrets

利用箇所

- *PoolerSecrets*

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

Field	Description
authQuery *_SecretVersion_*	認証クエリーのシークレットバージョン

6.4.76 PgBouncerSpec

利用箇所

- *PoolerSpec*

PgBouncerSpec は、PgBouncer の構成方法を定義します

Field	Description
*poolMode _PgBouncerPoolMode_*	プールモード。デフォルト `session`
authQuerySecret _github.com/cloudnative-pg/machinery/pkg/api.LocalObjectRefere	認証クエリー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" を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。
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 _map[string]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 ドキュメントを確認してください
pg_hba _[]string_	PostgreSQL ホストベースの認証ルール pg_hba.conf ファイルに追加される行
paused _bool_	`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。内部的には、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。

6.4.77 PluginConfiguration

利用箇所

- *ClusterSpec*

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

Field	Description
name **[Required]** _string_	Name はプラグイン名です
enabled _bool_	このプラグインを使用する場合、 Enabled は true
isWALArchiver _bool_	WALArchiver として宣言できるプラグインは 1 つだけです。".spec.backup.barmanObjectStore"構成が存在する場合、アクティブにできません。
parameters _map[string]string_	Parameters はプラグインの構成です

6.4.78 PluginStatus

利用箇所

- *ClusterStatus*

PluginStatus は、ロードされたプラグインのステータスです

Field	Description
name **[Required]** _string_	Name はプラグインの名前です
version **[Required]** _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
capabilities _[]string_	Capacity は、プラグインの機能のリストです。
operatorCapabilities _[]string_	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_	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_	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_	RestoreJobHookCapabilities は、RestoreJobHook 管理に関するプラグインの機能のリストです。
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

6.4.79 PodTemplateSpec

利用箇所

- *PoolerSpec*

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

Field	Description
metadata _Metadata_	標準オブジェクトのメタデータ。詳細情報 https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata
spec _core/v1.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

6.4.80 PodTopologyLabels

(`map[string]string` のエイリアス)

利用箇所

- レプリカクラスターから分散トポロジへ

PodTopologyLabels は、ポッドのトポロジを表します。map[labelName]labelValue

6.4.81 PoolerIntegrations

利用箇所

- *ClusterStatus*

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

Field	Description
pgBouncerIntegration _PgBouncerIntegrationStatus_	説明はありません。

6.4.82 PoolerMonitoringConfiguration

利用箇所

- *PoolerSpec*

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。

Field	Description
enablePodMonitor _bool_	`PodMonitor`を有効または無効にします
podMonitorMetricRelabelings [_[]github.com/prometheus-operator/prometheus-operator/pkg/apis/monitoring/**v1.Relabel**//pkg.go.dev/github.com/prometheus-operator/prometheus-operator/pkg/apis/monitoring/v1#RelabelConfig)	`PodMonitor`のメトリックの再ラベル付けのリスト。摂取する前にサンプルに適用されます。
podMonitorRelabelings [_[]github.com/prometheus-operator/prometheus-operator/pkg/apis/monitoring/**v1.Relabel**//pkg.go.dev/github.com/prometheus-operator/prometheus-operator/pkg/apis/monitoring/v1#RelabelConfig)	`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。

6.4.83 PoolerSecrets

利用箇所

- *PoolerStatus*

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

Field	Description
*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

6.4.84 PoolerSpec

利用箇所

- プーラー構成

PoolerSpec は、プーラーの目的の状態を定義します

Field	Description
cluster **[Required]** _github.com/cloudnative-pg/machinery/pkg/api.LocalObjectRefere	これは、プーラーが動作するクラスター参照です。プール名は、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。
*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`
instances _int32_	必要なレプリカの数。デフォルト 1
*template _PodTemplateSpec_*	作成する Pod のテンプレート
pgbouncer **[Required]** *_PgBouncerSpec_*	PgBouncer 構成
deploymentStrategy _apps/v1.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
*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

6.4.85 PoolerStatus

利用箇所

- プーラー構成

PoolerStatus は、プーラーの観察された状態を定義します

Field	Description
*secrets _PoolerSecrets_*	config オブジェクトのリソースバージョン
instances _int32_	スケジュールしようとしているポッドの数

6.4.86 PoolerType

string のエイリアス

利用箇所

- *PoolerSpec*

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。

6.4.87 PostgresConfiguration

利用箇所

- *ClusterSpec*

PostgresConfiguration は PostgreSQL 構成を定義します

Field	Description
parameters _map[string]string_	PostgreSQL 構成オプション postgresql.conf
synchronous *_SynchronousReplicaConfiguration_*	PostgreSQL 同期レプリケーション機能の構成
pg_hba _[]string_	PostgreSQL ホストベースの認証ルール pg_hba.conf ファイルに追加される行
pg_ident _[]string_	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 *_SynchronousReplicaElectionConstraints_*	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方法に影響します。
shared_preload_libraries _[]string_	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 _int32_	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 年より大きく、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です
enableAlterSystem _bool_	このパラメーターが true の場合、ユーザーはこの EDB Postgres AI for CloudNativePG™ クラスターで `ALTER SYSTEM`を呼び出すことができます。これは、デバッグとトラブルシューティングの場合にのみ使用してください。デフォルト false。
extensions [_[]ExtensionConfiguration_](ExtensionConfiguration)	追加する拡張機能の構成

6.4.88 PrimaryUpdateMethod

`string` のエイリアス

利用箇所

- *ClusterSpec*

PrimaryUpdateMethod 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使用するメソッドが含まれています

6.4.89 PrimaryUpdateStrategy

`string` のエイリアス

利用箇所

- *ClusterSpec*

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

6.4.90 Probe

利用箇所

- *LivenessProbe*

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。

Field	Description
initialDelaySeconds _int32_	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 https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes
timeoutSeconds _int32_	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 _int32_	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 _int32_	失敗した後に成功したとみなされるプローブの最小連続成功数。デフォルト 1。活性およびスタートアップの場合は 1 である必要があります。最小値は 1 です。
failureThreshold _int32_	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数。デフォルト 3。最小値は 1 です。

次のページに続く

表 77 – 前のページからの続き

Field	Description
terminationGracePeriodSeconds _int64_	プローブが失敗したときにポッドが正常に終了する必要がある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に終了信号が送信された後の秒単位の期間と、プロセスが kill 信号で強制的に停止される時間です。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も長く設定します。この値が nil の場合、ポッドの terminationGracePeriodSeconds 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 信号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、spec.terminationGracePeriodSeconds が使用されます。

6.4.91 ProbeStrategyType

`string` のエイリアス

利用箇所

- *ProbeWithStrategy*

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

6.4.92 ProbeWithStrategy

利用箇所

- *ProbesConfiguration*

ProbeWithStrategy は、スタートアッププローブの構成です

Field	Description
Probe *_Probe_*	`Probe`のメンバーはこのタイプに埋め込まれています。プローブは標準のプローブ構成です
type *_ProbeStrategyType_*	プローブ戦略
maximumLag _k8s.io/apimachinery/pkg/api/resource.Q	遅延制限。`streaming`ストラテジーにのみ使用されます

6.4.93 ProbesConfiguration

利用箇所

- *ClusterSpec*

ProbesConfiguration は、PostgreSQL ポッドに注入されるプローブの構成を表します

Field	Description
startup **[Required]** *_ProbeWithStrategy_*	起動プローブ構成
liveness **[Required]** *_LivenessProbe_*	liveness プローブ構成
readiness **[Required]** *_ProbeWithStrategy_*	Readiness Probe 構成

6.4.94 PublicationReclaimPolicy

`string` のエイリアス

利用箇所

- *PublicationSpec*

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。

6.4.95 PublicationSpec

利用箇所

- 出版物

PublicationSpec は、パブリケーションの望ましい状態を定義します

Field	Description
cluster **[Required]** _core/v1.LocalObjectReference_	「パブリッシャー」を識別する PostgreSQL クラスターの名前
name **[Required]** _string_	PostgreSQL 内のパブリケーションの名前
dbname **[Required]** _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
parameters _map[string]string_	PostgreSQL で想定される `WITH` 句のパブリケーションパラメーターの一部 `CREATE PUBLICATION` コマンド
target **[Required]** *_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

次のページに続く

表 80 – 前のページからの続き

Field	Description
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

6.4.96 PublicationStatus

利用箇所

- 出版物

PublicationStatus は、パブリケーションの観察された状態を定義します

Field	Description
observedGeneration _int64_	同期された最新の目的の状態を表すシーケンス番号
applied _bool_	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
message _string_	Message はリコンシリエーション出力メッセージです

6.4.97 PublicationTarget

利用箇所

- *PublicationSpec*

PublicationTarget は、このパブリケーションが公開するものです

Field	Description
allTables _bool_	データベース内のすべてのテーブル、今後作成されるテーブルを含む変更をレプリケートするパブリケーションとしてマークします。 PostgreSQL の `FOR ALL TABLES`に対応します。
objects [_[]PublicationTargetObject_](PublicationTargetObject)	次のスキーマオブジェクトのみ

6.4.98 PublicationTargetObject

利用箇所

- *PublicationTarget*

PublicationTargetObject は、パブリッシュするオブジェクトです

Field	Description
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、今後作成されるテーブルを含むすべてのテーブルの変更をレプリケートするパブリケーションとしてマークします。 PostgreSQL の `FOR TABLES IN SCHEMA`に対応します。
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`に対応します。

6.4.99 PublicationTargetTable

利用箇所

- *PublicationTargetObject*

PublicationTargetTable は、パブリッシュするテーブルです

Field	Description
only _bool_	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 **[Required]** _string_	テーブル名
schema _string_	スキーマ名
columns _[]string_	公開する列

6.4.100 RecoveryTarget

利用箇所

- *BootstrapRecovery*

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。

Field	Description
backupID _string_	リカバリプロセスを開始するバックアップの ID。空の場合デフォルト 、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
targetXID _string_	ターゲットトランザクション ID
targetName _string_	ターゲット名 `pg_create_restore_point`で事前に作成します
targetLSN _string_	ターゲット LSN ログシーケンス番号
targetTime _string_	RFC3339 標準のタイムスタンプとしてのターゲット時間
targetImmediate _bool_	一貫した状態に到達すると、リカバリーを終了します

次のページに続く

表 85 – 前のページからの続き

Field	Description
exclusive _bool_	ターゲットを排他的に設定します。省略した場合、デフォルトの false になるため、Postgres では `recovery_target_inclusive`が true になります

6.4.101 ReplicaClusterConfiguration

利用箇所

- *ClusterSpec*

ReplicaClusterConfiguration は、レプリカクラスターの構成をカプセル化します

Field	Description
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、これがプライマリクラスターであるかレプリカクラスターである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 **[Required]** _string_	レプリケーション元の外部クラスターの名前
enabled _bool_	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 _meta/v1.Duration_	レプリカモードが有効になっている場合、このパラメーターを使用すると、システム時間が少なくとも構成された時間コミット時間を経過している場合にのみトランザクションを再生できます。これにより、データ損失エラーを修正する機会が提供されます。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。

6.4.102 ReplicationSlotsConfiguration

利用箇所

- *ClusterSpec*

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

Field	Description
highAvailability _ReplicationSlotsHAConfiguration_	高可用性構成のレプリケーションスロット
updateInterval _int_	スタンバイは、`updateInterval`秒ごとに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デフォルト 30。
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

6.4.103 ReplicationSlotsHAConfiguration

利用箇所

- *ReplicationSlotsConfiguration*

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。

Field	Description
enabled _bool_	有効になっているデフォルト場合、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。無効にすると、オペレーターはレプリカとのストリーミング接続で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内のレプリケーションスロットも制御します。
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。
synchronizeLogicalDecoding _bool_	オペレーターは、有効にすると、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。 次の条件のいずれかが必要です。 *PostgreSQL* **バージョン** *17* **以降** pg_failover_slots 拡張機能が有効な PostgreSQL バージョン<17

6.4.104 RoleConfiguration

利用箇所

- *ManagedConfiguration*

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 参考 https://www.postgresql.org/docs/current/sql-createrole.html

Field	Description
name **[Required]** _string_	ロールの名前
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
passwordSecret _github.com/cloudnative-pg/machinery/pkg/api.LocalObjectRefere	ロールのパスワードを含むシークレット存在する場合 null 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
connectionLimit _int64_	ロールがログインできる場合、これはロールが確立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。
validUntil _meta/v1.Time_	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_	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。デフォルトは空。
inherit _bool_	ロールが、その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。
disablePassword _bool_	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 _bool_	ロールが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`superuser`であるかどうか。スーパーユーザーステータスは危険で、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 _bool_	`true`に設定すると、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 _bool_	ロールが他のロールのメンバーシップを作成、変更、ドロップ、コメント、セキュリティラベルの変更、および他のロールのメンバーシップを付与または取り消しできるかどうか。デフォルトは`false`です。
login _bool_	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 89 – 前のページからの続き

Field	Description
replication _bool_	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、レプリケーションスロットを作成またはドロップ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。 `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 _bool_	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。

6.4.105 SQLRefs

利用箇所

- *BootstrapInitDB*

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。

Field	Description
secretRefs [_]github.com/cloudnative-pg/machinery/pkg/**api.SecretKeySelecto** //pkg.go.dev/github.com/cloudnative-pg/machinery/pkg/api/#SecretKeySelector)	SecretRefs は、シークレットへの参照のリストを保持します
configMapRefs [_]github.com/cloudnative-pg/machinery/pkg/**api.ConfigMapKeyS** //pkg.go.dev/github.com/cloudnative-pg/machinery/pkg/api/#ConfigMapKeySelector)	ConfigMapRefs は、ConfigMaps への参照のリストを保持します

6.4.106 ScheduledBackupSpec

利用箇所

- *ScheduledBackup*

ScheduledBackupSpec は、 ScheduledBackup の目的の状態を定義します

Field	Description
suspend _bool_	このバックアップが一時停止されているかどうか
immediate _bool_	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 **[Required]** _string_	スケジュールは、追加の秒指定子が含まれるため、Kubernetes の Cron ジョブで使用される形式に従いません。参照してください。 https://pkg.go.dev/github.com/robfig/cron#hdr-CRON_Expression_Format
cluster **[Required]** _github.com/cloudnative-pg/machinery/pkg/api.LocalObjectRefere	バックアップするクラスター
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* **作成されたバックアップオブジェクトの所有者参照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バックアップの所有者として設定します * cluster クラスターをバックアップの所有者として設定します
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`になります。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バックアップを実行します。`prefer-standby`は、可能な場合は最も更新されたスタンバイでバックアップを実行します。
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*
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 _bool_	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* かどうか クラスターフィールド '.spec.backup.volumeSnapshot.online' で指定されたデフォルト設定をオーバーライドします
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 クラスターで指定されたデフォルト設定をオーバーライドします '.backup.volumeSnapshot.onlineConfiguration' スタンザ

6.4.107 ScheduledBackupStatus

利用箇所

- *ScheduledBackup*

ScheduledBackupStatus は、 ScheduledBackup の観察された状態を定義します

Field	Description
lastCheckTime _meta/v1.Time_	最新のスケジュール
lastScheduleTime _meta/v1.Time_	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 _meta/v1.Time_	次回バックアップを実行します

6.4.108 SchemaSpec

利用箇所

- *DatabaseSpec*

SchemaSpec はデータベースにスキーマを構成します

Field	Description
DatabaseObjectSpec *_DatabaseObjectSpec_*	`DatabaseObjectSpec`のメンバーはこのタイプに埋め込まれます。共通フィールド
owner **[Required]** _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。

6.4.109 SecretVersion

利用箇所

- *PgBouncerSecrets*

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

Field	Description
name _string_	シークレットの名前
version _string_	シークレットのリソースバージョン

6.4.110 SecretsResourceVersion

利用箇所

- *ClusterStatus*

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。

Field	Description
superuserSecretVersion _string_	"postgres"ユーザ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 95 – 前のページからの続き

Field	Description
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 _map[string]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 _map[string]string_	外部クラスターシークレットのリソースバージョン
metrics _map[string]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

6.4.111 ServiceAccountTemplate

利用箇所

- *ClusterSpec*

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

Field	Description
metadata **[Required]** *_Metadata_*	Metadata は、生成されたサービスアカウントに使用されるメタデータです

6.4.112 ServiceSelectorType

`string` のエイリアス

利用箇所

- *ManagedService*

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

6.4.113 ServiceTemplateSpec

利用箇所

- *ManagedService*

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。

Field	Description
metadata _Metadata_	標準オブジェクトのメタデータ。詳細情報 https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata
spec _core/v1.ServiceSpec_	サービスの目的な動作の仕様。詳細情報 https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status

6.4.114 ServiceUpdateStrategy

`string` のエイリアス

利用箇所

- *ManagedService*

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

6.4.115 SnapshotOwnerReference

`string` のエイリアス

利用箇所

- *VolumeSnapshotConfiguration*

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。

6.4.116 SnapshotType

`string` のエイリアス

利用箇所

- *OpenShift* の重要な概念

SnapshotType は許可されるインポートのタイプです

6.4.117 StorageConfiguration

利用箇所

- *ClusterSpec*

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

Field	Description
storageClass _string_	PVC に使用する StorageClass。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。このフィールドへの変更は、作成された PVC に自動的に再適用されます。サイズを小さくすることはできません。
resizeInUseVolumes _bool_	既存の PVC のサイズを変更します、デフォルトは true
pvcTemplate _core/v1.PersistentVolumeClaimSpec_	Persistent Volume Claim を生成するために使用されるテンプレート

6.4.118 SubscriptionReclaimPolicy

`string` のエイリアス

利用箇所

- *SubscriptionSpec*

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。

6.4.119 SubscriptionSpec

利用箇所

- サブスクリプション

SubscriptionSpec は、サブスクリプションの目的の状態を定義します

Field	Description
cluster **[Required]** _core/v1.LocalObjectReference_	「サブスクライバー」を識別する PostgreSQL クラスターの名前
name **[Required]** _string_	PostgreSQL 内のサブスクリプションの名前
dbname **[Required]** _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

次のページに続く

表 99 – 前のページからの続き

Field	Description
parameters _map[string]string_	PostgreSQL `CREATE SUBSCRIPTION`コマンドの`WITH`句に含まれるサブスクリプションパラメーター。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に変更できません。後で変更しても無視されます。次の文書に記載されている限定セットを除きます。 https://www.postgresql.org/docs/curren altersubscription.html# SQL-ALTERSUBSCRIPTION- PARAMS-SET
publicationName **[Required]** _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
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 **[Required]** _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
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

6.4.120 SubscriptionStatus

利用箇所

- サブスクリプション

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

Field	Description
observedGeneration _int64_	同期された最新の目的の状態を表すシーケンス番号
applied _bool_	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

6.4.121 SwitchReplicaClusterStatus

利用箇所

- *ClusterStatus*

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

Field	Description
inProgress _bool_	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。

6.4.122 SyncReplicaElectionConstraints

利用箇所

- *PostgresConfiguration*

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。

Field	Description
nodeLabelsAntiAffinity _[]string_	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 **[Required]** _bool_	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

6.4.123 SynchronizeReplicasConfiguration

利用箇所

- *ReplicationSlotsConfiguration*

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

Field	Description
enabled **[Required]** _bool_	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
excludePatterns _[]string_	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

6.4.124 SynchronousReplicaConfiguration

利用箇所

- *PostgresConfiguration*

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。

Field	Description
method **[Required]** *archive_timeout* as values.	NA
number **[Required]** _int_	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。
maxStandbyNamesFromCluster _int_	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。
standbyNamesPre _[]string_	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_	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。「preferred」に設定されている場合、正常なレプリカが利用可能な場合、データの耐久性は維持されますが、レプリカが利用できなくなると、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、操作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。

6.4.125 SynchronousReplicaConfigurationMethod

`string` のエイリアス

利用箇所

- *SynchronousReplicaConfiguration*

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

6.4.126 TDEConfiguration

利用箇所

- *EPASConfiguration*

TDEConfiguration には、透過的データ暗号化構成が含まれています

Field	Description
enabled _bool_	TDE を有効にする場合は True
secretKeyRef _core/v1.SecretKeySelector_	暗号化キーを含むシークレットへの参照
wrapCommand _core/v1.SecretKeySelector_	WrapCommand は、ユーザーが提供する encrypt コマンドです
unwrapCommand _core/v1.SecretKeySelector_	UnwrapCommand は、ユーザーが提供する復号コマンドです
passphraseCommand _core/v1.SecretKeySelector_	PassphraseCommand は、暗号化と復号化のために OpenSSL コマンドに渡されるパスフレーズを取得するために実行されるコマンドです

6.4.127 TablespaceConfiguration

利用箇所

- *ClusterSpec*

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

Field	Description
name **[Required]** _string_	テーブルスペースの名前
storage **[Required]** *_StorageConfiguration_*	テーブルスペースのストレージ構成
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 _bool_	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 ドキュメントを参照してください。

6.4.128 TablespaceState

利用箇所

- *ClusterStatus*

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

Field	Description
name **[Required]** _string_	Name はテーブルスペースの名前です

次のページに続く

表 107 – 前のページからの続き

Field	Description
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 **[Required]** *_TablespaceStatus_*	状態は最新の調整状態です
error _string_	Error はリコンシリエーションエラーである場合

6.4.129 TablespaceStatus

`string` のエイリアス

利用箇所

- *TablespaceState*

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

6.4.130 Topology

利用箇所

- *ClusterStatus*

トポロジにはクラスタートポロジが含まれます

Field	Description
instances [_map[PodName]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 _int32_	Nodes Used は、インスタンスを収容する個別のノードの数を表します。値「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る必要があり、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を意味します。
successfullyExtracted _bool_	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、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

6.4.131 VolumeSnapshotConfiguration

利用箇所

- *BackupConfiguration*

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。

Field	Description
labels _map[string]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 _map[string]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。特定のクラスが存在しない場合、それは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 _map[string]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
online _bool_	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*
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

6.5 CNCF Projects Integrations

6.5.1 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。

> **重要**
>
> 続行する前に、*cluster-example* Postgres クラスターが環境で稼働していることを確認してください。

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
```

(次のページに続く)

(前のページからの続き)

```
  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/v1beta1
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 }}"
        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™クラスター用に 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。

> **重要**
>
> この例では、 HashiCorp Vault が既にインストールされ、環境に適切に構成されており、チームがそれを操作するために必要な専門知識があることを前提としています。 Vault を展開するにはさまざまな方法があり、それらの詳細については、CloudNativePG™クラスターの EDB Postgres ® AI の範囲外です。Kubernetes 内で Vault を実行することは可能ですが、外部に展開されるのがより一般的です。詳細な手順については、 HashiCorp Vault documentation を参照してください。

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/v1beta1
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。

重要

この例では、基本的なトークンベースの認証を使用します。これは、API および CLI のユースケースをテストするのに適しています。これは Vault で有効になっているデフォルトの方法ですが、運用環境にはお勧めできません。実稼働環境では、より安全な認証方法の使用を検討します。 *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。

6.5.2 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許可されたポッドのみに制限することにより、CloudNativePG™クラスター PostgreSQL インスタンスの EDB Postgres ® AI を保護する方法を示します。

> **重要**
>
> 続行する前に、*cluster-example* Postgres クラスターが環境で稼働していることを確認してください。

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
```

(次のページに続く)

(前のページからの続き)

```
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 で 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
```

重要

postgresql-operator-system 名前空間は、YAML マニフェストを使用する場合のオペレーターのデフォルトの名前空間です。オペレーターが別のプロセス Helm、OLM などを使用してインストールされた場合、名前空間は異なる場合があります。名前空間を適切に調整してください。

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:
            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:
```

(次のページに続く)

(前のページからの続き)

```
      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
```

(次のページに続く)

(前のページからの続き)

```
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7章　Appendix

7.1 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 C - Common object stores for backups 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。

7.1.1 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:
      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ように、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 `SF03` の **Digital Ocean Spaces** の `bucket` を使用します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
[...]
spec:
  backup:
```

(次のページに続く)

(前のページからの続き)

```
    barmanObjectStore:
      destinationPath: "s3://[your-bucket-name]/[your-backup-folder]/"
      endpointURL: "https://sfo3.digitaloceanspaces.com"
      s3Credentials:
        [...]
```

プライベート CA でのオブジェクトストレージの使用

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。*CertificatesConfiguration* 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については、*CertificatesConfiguration* ドキュメントで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` のラベルを 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。

7.1.2 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™クラスター管理データベースの EDB Postgres ® AI のバックアップとリカバリーのためにストレージアカウントにアクセスするには、次の資格情報のいずれかの組み合わせが必要です。

- *Connection String*
- ストレージアカウント名と *Storage account access key*
- ストレージアカウント名と *Storage account SAS Token*
- ストレージアカウント名と Azure AD Workload Identity

適切に構成されました。

Azure AD Workload Identity 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
```

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:
        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:
```

(次のページに続く)

(前のページからの続き)

```
        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。

7.1.3 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:
  [...]
```

(次のページに続く)

(前のページからの続き)

```
  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。

重要

この認証方法は、Google Cloud Storage バケットにアクセスするために必要なすべての情報を含む JSON ファイルをコンテナ内に作成します。つまり、誰かがポッドへのアクセスを取得すると、バケットへの書き込み権限もあります。

7.2 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。

- **

7.2.1 主要なコンセプト

管理範囲

重要

CloudNativePG™クラスターの EDB Postgres ® AI は、データベース、ロール、テーブルスペースを含む PostgreSQL クラスターの**グローバルオブジェクト**を管理します。ただし、拡張機能とスキーマテーブルなどを超えてデータベースコンテンツは管理しません**。データベースのコンテンツを管理するには、専用のツールを使用するか、アプリケーション自分自身に依存します。

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。

```yaml
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* 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。

> **注釈**
>
> *metadata.name* と `spec.name` の違いにより、複数の `Database` リソースは、同じ Kubernetes 名前空間内の異なる EDB Postgres ® AI for CloudNativePG™ クラスターにわたって同じ名前のデータベースを参照できます。

7.2.2 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。

> **重要**
>
> *spec.name* を `postgres` 、*template0* 、または `template1` に設定して `Database` を作成することはできません。

7.2.3 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

```
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。

7.2.4 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。

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。

7.2.5 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。

7.2.6 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た EDB Postgres ® AI for CloudNativePG™ Cluster 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的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。

```
#  ...
```

(次のページに続く)

(前のページからの続き)

```
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 。

7.2.7 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。

7.3 Image Catalog

`ImageCatalog` および `ClusterImageCatalog` は、`Cluster` を作成するためのイメージを定義できるようにする重要なリソースです。

主な違いはスコープにあります。`ImageCatalog` は名前空間ですが、`ClusterImageCatalog` はクラスタースコープです。

どちらも、イメージのリストで構成される共通の構造を共有し、各イメージのメジャーバージョンを示す `major` フィールドを備えています。

> **警告**
>
> オペレーターはユーザー定義のメジャーバージョンを信頼し、PostgreSQL バージョンの検出の実行を控えます。カタログで宣言されたメジャーバージョンと PostgreSQL イメージ間のアライメントを保証するのはユーザーの責任です。

major フィールドの値は、カタログ内で一意を維持し、イメージ間の重複を防ぐ必要があります。ただし、別のカタログは、同じ major 値で異なるイメージを公開する場合があります。

名前空間 ``ImageCatalog`` の例

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: ImageCatalog
metadata:
  name: postgresql
  namespace: default
spec:
  images:
    - major: 15
      image: quay.io/enterprisedb/postgresql:15.6
    - major: 16
      image: quay.io/enterprisedb/postgresql:16.8
    - major: 17
      image: quay.io/enterprisedb/postgresql:17.5
```

``ClusterImageCatalog`` リソースを使用したクラスター全体のカタログの例

```yaml
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: ClusterImageCatalog
metadata:
  name: postgresql
spec:
  images:
    - major: 15
      image: quay.io/enterprisedb/postgresql:15.6
    - major: 16
      image: quay.io/enterprisedb/postgresql:16.8
    - major: 17
      image: quay.io/enterprisedb/postgresql:17.5
```

Cluster リソースは、 ImageCatalog または ClusterImageCatalog を参照して、目的の画像を正確に指定する柔軟性を備えています。

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

(次のページに続く)

(前のページからの続き)

```
    major: 16
  storage:
    size: 1Gi
```

これらのカタログを使用するクラスターは、継続的な監視を維持します。カタログ内の画像が変更されると、その特定のエントリを参照する**関連するすべてのクラスター** の自動更新がトリガーされます。

7.3.1 CloudNativePG™クラスターカタログ用の EDB Postgres ® AI

EDB Postgres ® AI for CloudNativePG™ Cluster プロジェクトは、提供する画像の `ClusterImageCatalogs` を維持します。これらのカタログは、各メジャーバージョンの最新のイメージで定期的に更新されます。EDB Postgres ® AI for CloudNativePG™クラスタープロジェクトの GitHub リポジトリから `ClusterImageCatalog.yaml` ファイルを適用することにより、クラスター管理者は、クラスターが指定されたメジャーリリース内の最新バージョンに自動的に更新されるようにできます。

PostgreSQL コンテナイメージ

latest version of the cluster catalog for the PostgreSQL Container Images をインストールできます

cloudnative-pg/postgres-containers リポジトリ)

```
kubectl apply \
  -f https://raw.githubusercontent.com/cloudnative-pg/postgres-containers/main/Debian/
↪ClusterImageCatalog-bookworm.yaml
```

PostGIS コンテナイメージ

latest version of the cluster catalog for the PostGIS Container Images をインストールできます

cloudnative-pg/postgis-containers リポジトリ)

```
kubectl apply \
  -f https://raw.githubusercontent.com/cloudnative-pg/postgis-containers/main/PostGIS/
↪ClusterImageCatalog.yaml
```

7.4 Iron Bank

CloudNativePG™クラスター {{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。

7.4.1 Iron Bank から EDB {{name.abbr}}とオペランドイメージをプルする

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。

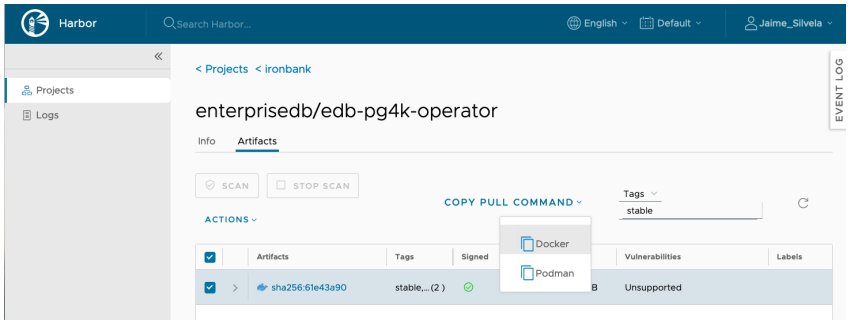


図 1: pulling-ironbank-images

たとえば、Ironbank から EPAS16 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-16:16
```

より具体的なタグを選択するか、特定の SHA を使用する場合は、Harbor page からそれを見つける必要があります。

7.4.2 Iron Bank イメージを使用した{{name.abbr}}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.abbr}}* 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` を使用して通常にマニフェストを適用できます。

7.4.3 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 16 オペランドを使用して{{name.abbr}}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-17:17
  imagePullSecrets:
    - name: my_ironbank_secret
```

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
```

7.5 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。

重要

CloudNativePG™クラスターの EDB Postgres ® AI でのパブリッシャークラスターのフェールオーバーの後に論理レプリケーションサブスクライバーを保護するには、論理デコードのレプリケーションスロット同期が有効になっていることを確認します。これがないと、論理レプリケーションクライアントはデータを失い、フェールオーバー後にシームレスに続行できない場合があります。構成の詳細については、 *Replication: Logical Decoding Slot Synchronization* を参照してください。

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。

EDB Postgres ® AI for CloudNativePG™ クラスター 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**

7.5.1 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**レプリケーションセット**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、**パブリッシャー** 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™ クラスター 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* 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
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。

重要

allTables は構成を簡素化しますが、PostgreSQL は特定のテーブルまたはターゲットを絞ったデータ変更をレプリケートするためのきめの細かい制御を提供します。高度な構成については、 *PostgreSQL documentation* を参照してください。さらに、:ref:` EDB Postgres ® AI for CloudNativePG™ Cluster API reference <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、

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。

調整とステータス

`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。

7.5.2 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、**サブスクライバー**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。

> **重要**
>
> スキーマ定義はレプリケートされないため、サブスクライバーは、データレプリケーションを開始する前に、対応するテーブルを既に定義している必要があります。

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* 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 データベース上の 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
```

(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
```

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。

調整とステータス

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
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。

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操作可能な状態を維持するには、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。

7.5.3 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、 EDB Postgres ® AI for CloudNativePG™ Cluster の機能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。: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>` 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。

7.5.4 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™クラスターの論理レプリケーションと EDB Postgres ® AI を組み合わせることで、Kubernetes 環境で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: Cluster
metadata:
  name: freddie
spec:
  instances: 1

  imageName: quay.io/enterprisedb/postgresql:16

  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

  managed:
    roles:
      - name: app
        login: true
        replication: true
- --
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

  storage:
    size: 1Gi

  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
```

(次のページに続く)

(前のページからの続き)

```
  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。

7.6 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 17.0 から 17.1
- メジャーバージョンのアップグレード 例、16.x から 17.0

7.6.1 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 17.1 では

- 17 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。たとえば、PostgreSQL 17.1 は、17.0 および 17.5 と互換性があります。

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。

7.6.2 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 **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。

> **重要**
>
> 運用アップグレードを続行する前に、制御された環境ですべてのメソッドをテストすることを強くお勧めします。

7.6.3 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。

> **重要**
>
> メジャーアップグレードは、同じオペレーティングシステムディストリビューションに基づくイメージ間でのみサポートされています。たとえば、以前のバージョンで `bullseye` イメージを使用している場合、*bookworm* イメージにアップグレードすることはできません。

> **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 18beta2 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。
- *Image Catalog* 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、 EDB Postgres ® AI for CloudNativePG™クラスター

- 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、完全なバックアップを取ることを強くお勧めします。既存のバックアップデータつまりベースバックアップと WAL ファイルは、以前のマイナー PostgreSQL リリースでのみ使用できます。

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。

CloudNativePG™クラスターの EDB Postgres ® AI はロールバックを自動的に決定できないため、アップグレードが失敗した場合は、クラスターの構成のメジャーバージョンの変更を手動で元に戻し、アップグレードジョブを削除する必要があります。

重要

このプロセスは、アップグレード中にデータが変更されないため、**既存のデータベースをデータ損失から保護します**。アップグレードが失敗した場合、通常はバックアップから完全なリカバリを実行せずにロールバックが可能です。プロセスを注意深く監視し、必要に応じて修正アクションを実行します。

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: quay.io/enterprisedb/postgresql:16-minimal-bookworm
  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 17 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 17 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: quay.io/enterprisedb/postgresql:17-minimal-bookworm
  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-4` および `cluster-example-5` 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 17.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
```

7.7 Preview Versions

CloudNativePG™クラスター用の EDB Postgres ® AI 候補リリースは、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。

> **重要**
>
> EDB Postgres ® AI for CloudNativePG™ Cluster リリース候補は、実稼働システムでの使用を想定していません。

7.7.1 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じにな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。

7.7.2 コミュニティへの参加

CloudNativePG™クラスター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。

7.7.3 使用アドバイザリー

CloudNativePG™クラスターコミュニティの 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™クラスターの 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。

7.7.4 現在のプレビューバージョン

現在のプレビューはありません。次の準備ができたらこのページを更新します。

7.8 EDB private container registries

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。

重要

プライベートレジストリへのアクセスには、EDB のアカウントが必要で、有効な subscription plan を持つ EDB 顧客に予約されています。資格情報は、EDB アカウントを介して収集されます。

重要

レジストリごとに 10GB/月の帯域幅クォータがあります。

注釈

プライベートレジストリからオペレーターとオペランドをインストールする場合、 *License and License keys* は必要ありません。

7.8.1 どのリポジトリを選択しますか？

CloudNativePG™ Cluster の EDB Postgres ® AI は、「EDB Enterprise プラン」または「EDB Standard プラン」のいずれかで利用できます。

サブスクリプションプランに応じて、CloudNativePG™クラスターの EDB Postgres ® AI は、以下の表で説明しているように、次のリポジトリのいずれかにあります。

Plan	Repository
EDB Standard Plan	*k8s_standard*
EDB EnterpriseDB Plan	*k8s_enterprise*

リポジトリの名前は、たとえば `docker login` または kubernetes.io/dockerconfigjson を介して EDB コンテナレジストリにログインしようとするときに**ユーザー名** として使用されます。

重要

各リポジトリには、プランでアクセスできるすべてのイメージが含まれています。演算子やオペランドイメージなどのさまざまなイメージにアクセスするために別のリポジトリに接続する必要はありません。

7.8.2 トークンの取得方法

repositories page in EDB には、 `Repo Token` が隠れて表示されている *EDB Repos 2.0* セクションがあります。同じトークンは、 `Repos 2.0 token` というラベルの付いた Account profile でも使用できます。

トークンの横には、トークンをコピーするボタンと、トークンのコンテンツをクリアテキストとして表示する場合の目のアイコンがあります。トークンは、EDB コンテナレジストリにアクセスしようとするときに**パスワード** として使用されます。

`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るはずです。 `Copy Token` ボタンを使用してリポトークンをコピーすることをお勧めします。次の `docker` コマンドは、ユーザー名とパスワードの入力を求めます。

上記で説明したように、ユーザー名はアクセスしようとしているリポジトリであり、パスワードはコピーしたばかりのトークンである必要があります。

```
$ docker login docker.enterprisedb.com
Username: k8s_enterprise
Password:
Login Succeeded
```

7.8.3 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含むさまざまな PostgreSQL ディストリビューションをサポートしています。

- EDB Postgres Advanced EPAS
- EDB Postgres Extended PGE

> **注釈**
>
> PostgreSQL イメージはプライベートレジストリでは利用できませんが、 *quay.io/enterprisedb/postgresql* または `ghcr.io/enterprisedb/postgresql` ですぐに利用できます。

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™ Cluster 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	Repositories
EDB Postgres Extended (PGE)	*edb-postgres-extended*	*k8s_standard*, *k8s_enterprise*
EDB Postgres Advanced (EPAS)	*edb-postgres-advanced*	*k8s_enterprise*

7.8.4 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 EDB Postgres ® AI for CloudNativePG™クラスターオペレーター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=<REGISTRY-NAME> \
  --docker-password=<TOKEN>
```

前述のように、docker-username はレジストリの名前です。つまり、k8s_standard または k8s_enterprise です。 docker-password は、**トークンの取得方法** 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、k8s_enterprise リポジトリから 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_enterprise/edb-postgres-extended:16.2
  imagePullSecrets:
  - name: registry-pullsecret

  storage:
    storageClass: standard
    size: 1Gi
```

7.8.5 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 **ユーザー名** はリポジトリの名前であり、トークンは**パスワード** 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_enterprise/edb-postgres-for-
↪kubernetes \
  --set image.imageCredentials.username=k8s_enterprise \
  --set image.imageCredentials.password=<ENTITLEMENT_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
```

7.9 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。

> **重要**
>
> デフォルトのサービス名は、CloudNativePG™クラスターを使用するために EDB Postgres ® AI 用に予約されています。

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。

7.9.1 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。

重要

CloudNativePG™クラスターの EDB Postgres ® AI は PostgreSQL レプリケーションを保証するためにそれに依存しているため、 *rw* サービスは必須であり、無効にすることはできません。

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"]
```

7.9.2 独自のサービスの追加

重要

独自のサービスを定義する場合、規約 `<CLUSTER_NAME>-<SERVICE_NAME>` に従っているデフォルトの予約サービス名は使用できません。 Kubernetes 名前空間でサービスの一意の名前を選択するのはあなたの責任です。

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。

```
#  <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。

7.10 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。

7.10.1 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 **宣言テーブルスペース**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 EDB Postgres ® AI for CloudNativePG™ Cluster の宣言的テーブルスペースは、永続ボリュームクレーム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応される予定です。

7.10.2 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 クラスター内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
```

7.10.3 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;
```

重要

この例では、 PostgreSQL declarative partitioning に精通していることを前提としています。

7.10.4 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
```

重要

表領域の所有権を変更する場合は、既存のロールを使用していることを確認します。それ以外の場合、クラスターのステータスが問題を報告し、修正されるまでテーブルスペースのリコンサイルを停止します。ステータスとログを監視し、問題を修正して迅速に介入するのはあなたの責任です。

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
```

(次のページに続く)

(前のページからの続き)

```
# ...

tablespaces:
  - name: tbs1
    storage:
      size: 1Gi
  - name: tbs2
    storage:
      size: 2Gi
  - name: tbs3
    owner:
      name: badhombre
    storage:
      size: 2Gi
      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
```

7.10.5 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。

7.10.6 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
```

7.10.7 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。

7.10.8 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
[...]
```

7.10.9 制限事項

現在、既存の EDB Postgres ® AI for CloudNativePG™ Cluster クラスターからテーブルスペースを削除することはできません。