The CloudNativePG Contributors.

リリース *1.22.1*

The CloudNativePG Contributors.

2024 年 02 月 07 日

目 次

第 1 章　はじめに **3**
1.1　サポートされている Kubernetes ディストリビューション . . . 3
1.2　コンテナイメージ . . . 3
1.3　主な機能 . . . 3
1.4　このガイドについて . . . 5

第 2 章　始める前に **7**
2.1　Kubernetes の用語 . . . 7
2.2　PostgreSQL の用語 . . . 8
2.3　クラウド用語 . . . 9
2.4　次に何をするか . . . 9

第 3 章　ユースケース **11**
3.1　ケース 1Kubernetes 内のアプリケーション . . . 11
3.2　ケース 2Kubernetes 外部のアプリケーション . . . 11

第 4 章　アーキテクチャ **15**
4.1　状態の同期 . . . 15
4.2　Kubernetes アーキテクチャ . . . 16
4.2.1　マルチアベイラビリティーゾーン Kubernetes クラスター . . . 16
4.2.2　単一のアベイラビリティーゾーン Kubernetes クラスター . . . 17
4.3　PostgreSQL アーキテクチャ . . . 18
4.3.1　読み取り/書き込みワークロード . . . 19
4.3.2　読み取り専用ワークロード . . . 19
4.4　Kubernetes クラスター全体のデプロイメント . . . 19

第 5 章　インストールとアップグレード **23**
5.1　Kubernetes へのインストール . . . 23
5.1.1　オペレーターマニフェストを直接使用する . . . 23
5.1.2　`kubectl` の `cnpg` プラグインを使用する . . . 23
5.1.3　最新の開発スナップショットのテスト . . . 24
5.1.4　Helm チャートを使用する . . . 24
5.1.5　OLM を使用する . . . 24
5.2　デプロイメントの詳細 . . . 25
5.3　アップグレード . . . 25
5.3.1　インスタンスマネージャーのインプレース更新 . . . 26
5.3.2　バージョン間の互換性 . . . 27
5.3.3　1.22.0、1.21.2 または 1.20.5 へのアップグレード . . . 27
5.3.4　以前のマイナーバージョンから 1.21 へのアップグレード . . . 28
5.3.5　スーパーユーザーアクセスは無効になっています . . . 29
5.3.6　HA のレプリケーションスロット . . . 29

5.3.7 PostgreSQL のシャットダウンの遅延 . . . 29
5.3.8 PostgreSQL の起動の遅延 . . . 30
5.3.9 PostgreSQL スイッチオーバーの遅延 . . . 31
5.3.10 ラベル . . . 31
5.3.11 既存の動作を維持するためのショートカット . . . 31
5.3.12 以前のマイナーバージョンから 1.20 へのアップグレード . . . 32
5.3.13 スタンバイからのバックアップ . . . 32
5.3.14 ローリング更新後のプライマリの再起動 . . . 32
5.3.15 高可用性のためのレプリケーションスロット . . . 33

第 6 章 クイックスタート 35
6.1 パート 1: ローカル Kubernetes プレイグラウンドをセットアップする . . . 35
6.1.1 Minikube . . . 36
6.1.2 種類 . . . 36
6.2 パート 2 CloudNativePG のインストール . . . 36
6.3 パート 3 PostgreSQL クラスターを展開する . . . 37
6.4 パート 4 Prometheus と Grafana を使用してクラスターを監視する . . . 38
6.4.1 インストール . . . 39
6.4.2 Prometheus での表示 . . . 40
6.4.3 Grafana ダッシュボード . . . 41

第 7 章 Bootstrap 43
7.1 `bootstrap` セクション . . . 44
7.2 `externalClusters` セクション . . . 44
7.2.1 パスワードファイル . . . 45
7.3 空のクラスターをブートストラップする `initdb` . . . 45
7.3.1 `initdb` にオプションを渡す . . . 47
7.3.2 初期化後のクエリの実行 . . . 49
7.4 別のクラスターからのブートストラップ . . . 50
7.4.1 バックアップからのブートストラップ `recovery` . . . 50
7.4.2 ライブクラスターからのブートストラップ `pg_basebackup` . . . 50
7.4.3 要件 . . . 51
7.4.4 レプリケーションユーザーについて . . . 52
7.4.5 ユーザー名/パスワードの認証 . . . 52
7.4.6 TLS 証明書認証 . . . 53
7.4.7 現在の制限 . . . 55

第 8 章 Postgres データベースのインポート 57
8.1 その仕組み . . . 58
8.2 `microservice` タイプ . . . 58
8.3 `monolith` タイプ . . . 60
8.4 インポートの最適化 . . . 62

第 9 章 セキュリティ 65
9.1 コード . . . 65
9.2 コンテナ . . . 65
9.3 クラスター . . . 66
9.3.1 ロールベースのアクセス制御 RBAC . . . 66

9.3.2 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 . . . 67
9.3.3 ポッドセキュリティポリシー . . . 69
9.3.4 AppArmor を使用してポッドアクセスを制限する . . . 69
9.3.5 ネットワークポリシー . . . 70
9.3.6 公開ポート . . . 70
9.3.7 PostgreSQL . . . 70
9.3.8 ストレージ . . . 72

第 10 章 Postgres インスタンスマネージャー 73
10.1 スタートアップ、活性、および準備状況プローブ . . . 73
10.2 シャットダウン制御 . . . 74
10.2.1 スイッチオーバー中のプライマリのシャットダウン . . . 74
10.3 フェイルオーバー . . . 75

第 11 章 スケジューリング 77
11.1 ポッドのアフィニティとアンチアフィニティ . . . 77
11.2 `nodeSelector` によるノード選択 . . . 79
11.3 許容範囲 . . . 79

第 12 章 リソース管理 81

第 13 章 障害モード 85
13.1 ストレージ領域の使用量 . . . 85
13.2 障害モード . . . 86
13.2.1 ユーザーによって削除されたポッド . . . 86
13.2.2 Readiness プローブの失敗 . . . 87
13.2.3 Liveness プローブの障害 . . . 87
13.2.4 ワーカーノードがドレインされました . . . 87
13.2.5 ワーカーノードの障害 . . . 88
13.3 自己修復 . . . 88
13.4 手動介入 . . . 88

第 14 章 ローリングアップデート 91
14.1 自動更新 `unsupervised` . . . 91
14.2 手動更新 `supervised` . . . 92

第 15 章 レプリケーション 93
15.1 アプリケーションレベルのレプリケーション . . . 93
15.2 非常に成熟したテクノロジー . . . 93
15.3 ストリーミングレプリケーションのサポート . . . 94
15.3.1 継続的なバックアップ統合 . . . 94
15.4 同期レプリケーション . . . 95
15.4.1 同期レプリケーションのノードを選択します . . . 96
15.5 高可用性のためのレプリケーションスロット . . . 97
15.5.1 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 . . . 99

第 16 章 バックアップ リカバリー 101

第 17 章 バックアップ 103

17.1 WAL アーカイブ . . . 104
17.2 コールドおよびホットバックアップ . . . 105
17.3 オブジェクトストアまたはボリュームスナップショットどちらを使用しますか? . . . 105
17.4 スケジュールされたバックアップ . . . 107
17.5 オンデマンド バックアップ . . . 108
17.6 スタンバイからのバックアップ . . . 110

第 18 章 オブジェクトストアのバックアップ 113
18.1 共通オブジェクトストア . . . 113
18.2 リテンション ポリシー . . . 114
18.3 圧縮アルゴリズム . . . 114
18.4 バックアップオブジェクトのタグ付け . . . 115

第 19 章 WAL アーカイブ 117

第 20 章 ボリュームスナップショットのバックアップ 119
20.1 標準のボリュームスナップショットについて . . . 119
20.2 要件 . . . 120
20.3 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 . . . 120
20.4 ホットおよびコールドバックアップ . . . 121
20.4.1 デフォルト動作のオーバーライド . . . 122
20.5 ボリュームスナップショットオブジェクトの永続化 . . . 123
20.6 例 . . . 124

第 21 章 リカバリー 127
21.1 オブジェクトストアからのリカバリー . . . 128
21.2 `VolumeSnapshot` オブジェクトからのリカバリー . . . 129
21.3 `Backup` オブジェクトからのリカバリー . . . 130
21.4 追加の考慮事項 . . . 131
21.5 ポイント インタイム リカバリー PITR . . . 131
21.5.1 オブジェクトストアからの PITR . . . 132
21.5.2 `VolumeSnapshot` オブジェクトからの PITR . . . 133
21.5.3 リカバリーターゲット . . . 134
21.6 アプリケーションデータベースの構成 . . . 136
21.7 リカバリーの内部的な仕組み . . . 137
21.8 バックアップセクションを使用したクラスターへのリストア . . . 138

第 22 章 PostgreSQL の構成 139
22.1 `postgresql` セクション . . . 140
22.1.1 レプリケーション設定 . . . 142
22.1.2 ログ制御設定 . . . 142
22.1.3 共有プリロードライブラリ . . . 142
22.1.4 マネージド拡張機能 . . . 143
22.1.5 `auto_explain` の有効化 . . . 144
22.1.6 `pg_stat_statements` の有効化 . . . 144
22.1.7 `pgaudit` の有効化 . . . 145
22.1.8 `pg_failover_slots` の有効化 . . . 145
22.2 `pg_hba` セクション . . . 146

22.2.1 LDAP 構成 . . . 147
22.3 pg_ident セクション . . . 147
22.4 構成の変更 . . . 148
22.5 ALTER SYSTEM の有効化 . . . 148
22.6 動的共有メモリ設定 . . . 149
22.6.1 POSIX 共有メモリ . . . 149
22.6.2 System V 共有メモリ . . . 150
22.7 固定パラメーター . . . 150

第 23 章 データベースロール管理 153
23.1 パスワード管理 . . . 155
23.1.1 パスワードの有効期限 VALID UNTIL . . . 156
23.1.2 パスワードがハッシュ化されました . . . 157
23.2 実現不可能なロール構成 . . . 157
23.3 管理対象ロールのステータス . . . 158

第 24 章 テーブルスペース 161
24.1 宣言的テーブルスペース . . . 161
24.2 宣言的テーブルスペースの使用 . . . 162
24.3 ストレージクラスとテーブルスペース . . . 164
24.4 テーブルスペースの所有権 . . . 165
24.5 バックアップ リカバリー . . . 166
24.6 レプリカクラスター . . . 166
24.7 一時テーブルスペース . . . 167
24.8 kubectl プラグインのサポート . . . 168
24.9 制限事項 . . . 168

第 25 章 オペレーター設定 169
25.1 使用可能なオプション . . . 169
25.2 オペレーター構成マップの定義 . . . 170
25.3 オペレーターシークレットの定義 . . . 171
25.4 オペレーターを再起動して構成をリロードする . . . 171
25.5 pprof HTTP Server . . . 172

第 26 章 インスタンスポッド構成 173
26.1 予測ボリューム . . . 173
26.2 エフェメラルボリューム . . . 175

第 27 章 一時ストレージのボリューム要求テンプレート 177
27.1 環境変数 . . . 178

第 28 章 ストレージ 181
28.1 バックアップ リカバリー . . . 182
28.2 CloudNativePG のベンチマーク . . . 182
28.3 保存時の暗号化 . . . 183
28.4 Persistent Volume Claim Claim PVC . . . 183
28.5 ストレージクラスを介した構成 . . . 183
28.6 PVC テンプレートを介した構成 . . . 184
28.7 WAL のボリューム . . . 184

28.8 テーブルスペースのボリューム . . . 186
28.9 ボリュームの拡張 . . . 186
28.9.1 ボリューム拡張 Kubernetes 機能の使用 . . . 186
28.9.2 AKS での PVC ボリュームの拡張 . . . 186
28.9.3 AKS のボリューム拡張の回避策 . . . 187
28.9.4 ストレージの再作成 . . . 188
28.10 永続ボリュームの静的プロビジョニング . . . 189
28.11 ブロックストレージの考慮事項 Ceph/Longhorn . . . 190

第 29 章 ラベルとアノテーション **191**
29.1 事前定義されたラベル . . . 191
29.2 事前定義されたアノテーション . . . 192
29.3 前提条件 . . . 193
29.4 クラスターのメタデータの定義 . . . 194
29.5 現在の制限 . . . 194

第 30 章 モニタリング **195**
30.1 モニタリングインスタンス . . . 195
30.1.1 Prometheus Operator の例 . . . 196
30.1.2 事前定義されたメトリックセット . . . 197
30.1.3 ユーザー定義メトリック . . . 206
30.1.4 ユーザー定義メトリックの例 . . . 207
30.1.5 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 . . . 208
30.1.6 ユーザー定義メトリックの構造 . . . 209
30.1.7 ユーザー定義メトリックの出力 . . . 210
30.1.8 メトリックのデフォルトセット . . . 211
30.1.9 Prometheus Postgres エクスポーターとの違い . . . 212
30.2 オペレーターのモニタリング . . . 212
30.2.1 Prometheus Operator の例 . . . 212
30.3 エクスポートされたメトリックを検査する方法 . . . 213
30.4 補助リソース . . . 214

第 31 章 ロギング **215**
31.1 オペレーターログ . . . 215
31.2 PostgreSQL ログ . . . 216
31.3 PGAudit ログ . . . 217
31.4 その他のログ . . . 218

第 32 章 証明書 **221**
32.1 オペレーター管理モード . . . 221
32.1.1 サーバー証明書 . . . 222
32.1.2 サーバー CA シークレット . . . 222
32.1.3 サーバー TLS シークレット . . . 222
32.1.4 サーバー代替 DNS 名 . . . 222
32.1.5 クライアント証明書 . . . 222
32.1.6 クライアント CA シークレット . . . 222
32.1.7 クライアント `streaming_replica`証明書 . . . 222
32.2 ユーザー提供の証明書モード . . . 223

32.2.1 サーバー証明書 . . . 223
32.2.2 例 . . . 223
32.2.3 Cert-manager の例 . . . 224
32.2.4 クライアント証明書 . . . 228
32.2.5 Cert-manager の例 . . . 229

第 33 章 クライアント TLS/SSL 接続 233
33.1 新しい証明書の発行 . . . 233
33.2 TLS 証明書を介した接続のテスト . . . 234
33.3 TLS プロトコルバージョンについて . . . 237

第 34 章 アプリケーションから接続する 239
34.1 DNS 解決 . . . 239
34.2 環境変数 . . . 240
34.3 シークレット . . . 240

第 35 章 接続プーリング 241
35.1 アーキテクチャ . . . 241
35.2 クイックスタート . . . 242
35.3 リソースのライフサイクルのプール化 . . . 243
35.4 セキュリティ . . . 243
35.4.1 証明書 . . . 243
35.5 認証 . . . 244
35.6 ポッドテンプレート . . . 245
35.7 高可用性 HA . . . 246
35.8 PgBouncer 構成オプション . . . 247
35.9 モニタリング . . . 248
35.10 ロギング . . . 255
35.11 接続の一時停止 . . . 256
35.12 制限事項 . . . 256
35.12.1 単一の PostgreSQL クラスター . . . 256
35.12.2 制御された構成可能性 . . . 256

第 36 章 レプリカクラスター 257
36.1 基本的な概念 . . . 257
36.2 レプリカクラスターのセットアップ . . . 258
36.2.1 pg_basebackup を使用した例 . . . 259
36.2.2 オブジェクトストアからのバックアップを使用した例 . . . 260
36.2.3 ボリュームスナップショットを使用した例 . . . 261
36.3 レプリカクラスターでの指定されたプライマリの昇格 . . . 261

第 37 章 Kubernetes のアップグレード 263
37.1 `reusePVC` が `false` に設定された単一インスタンスクラスター . . . 264

第 38 章 Postgres サービスの公開 267
38.1 Minikube でのテスト . . . 270

第 39 章 Kubectl プラグイン 271
39.1 インストール . . . 271

39.1.1 インストールスクリプトを介して . . . 271
39.1.2 Debian または RedHat パッケージを使用する . . . 271
39.1.3 Debian パッケージ . . . 271
39.1.4 RPM パッケージ . . . 272
39.1.5 Krew を使用する . . . 273
39.1.6 サポートされているアーキテクチャ . . . 273
39.2 使用する . . . 273
39.2.1 インストールマニフェストの生成 . . . 273
39.2.2 ステータス . . . 274
39.2.3 プロモート . . . 278
39.2.4 証明書 . . . 278
39.2.5 再起動 . . . 279
39.2.6 リロード . . . 279
39.2.7 メンテナンス . . . 280
39.2.8 レポート . . . 280
39.2.9 レポート演算子 . . . 280
39.2.10 レポートクラスター . . . 283
39.2.11 ログ . . . 285
39.2.12 クラスターログ . . . 285
39.2.13 デストロイ . . . 286
39.2.14 クラスターハイバーネーション . . . 287
39.2.15 pgbench を使用したデータベースのベンチマーク . . . 288
39.2.16 fio を使用したストレージのベンチマーク . . . 288
39.2.17 新しい物理バックアップの要求 . . . 288
39.2.18 psql の起動 . . . 289
39.2.19 Postgres クラスターのスナップショット . . . 290
39.2.20 評価/デモ目的でのみ pgAdmin4 を使用する . . . 290
39.3 K9s との統合 . . . 291

第 40 章 自動フェイルオーバー **293**
40.1 RTO と RPO のインパクト . . . 294
40.2 遅延フェイルオーバー . . . 294

第 41 章 障害対応 **295**
41.1 始める前に . . . 295
41.1.1 Kubernetes 環境 . . . 295
41.1.2 便利なユーティリティ . . . 295
41.2 最初の一歩 . . . 296
41.3 バックアップはありますか？ . . . 296
41.4 緊急バックアップ . . . 297
41.5 ログ . . . 298
41.6 オペレーター情報 . . . 298
41.6.1 オペレーターに関する詳細情報を収集する . . . 299
41.7 クラスター情報 . . . 300
41.8 ポッド情報 . . . 301
41.8.1 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 . . . 302
41.9 バックアップ情報 . . . 303

41.10 ストレージ情報 . . . 303
41.11 ノード情報 . . . 304
41.12 条件 . . . 304
41.12.1 特定の条件を待機する方法 . . . 305
41.13 ネットワーキング . . . 305
41.14 PostgreSQL コアダンプ . . . 306
41.15 よくある問題 . . . 307
41.15.1 ストレージがいっぱいです . . . 307
41.15.2 ポッドが Pending 状態でスタックしている . . . 307
41.15.3 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 . . . 308
41.15.4 クラスターが Creating new replica でスタックしている . . . 308
41.15.5 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 308
41.15.6 データディレクトリのブートストラップ中のエラー . . . 309

第 42 章 フェンシング 311
42.1 インスタンスをフェンスする方法 . . . 311
42.2 フェンシングのリフティング方法 . . . 312
42.3 フェンシングの仕組み . . . 312

第 43 章 宣言ハイバーネーション 315
43.1 ハイバネーション . . . 315
43.2 水分補給 . . . 316

第 44 章 PostGIS 317
44.1 PostGIS クラスターに関する基本的な概念 . . . 317
44.2 PostGIS で新しい PostgreSQL クラスターを作成する . . . 318

第 45 章 エンドツーエンドテスト 321

第 46 章 コンテナイメージの要件 323
46.1 イメージタグの要件 . . . 324

第 47 章 オペレーター能力レベル 325
47.1 レベル 1 基本インストール . . . 325
47.1.1 宣言的構成を介した Operator の展開 . . . 326
47.1.2 宣言的構成を介した PostgreSQL クラスターの展開 . . . 326
47.1.3 CRD を介したオペランドイメージのオーバーライド . . . 326
47.1.4 ラベルとアノテーション . . . 326
47.1.5 自己完結型インスタンスマネージャー . . . 326
47.1.6 ストレージ構成 . . . 327
47.1.7 レプリカ構成 . . . 327
47.1.8 データベース構成 . . . 327
47.1.9 Postgres のロール、ユーザー、およびグループの構成 . . . 327
47.1.10 ポッドセキュリティポリシー . . . 328
47.1.11 アフィニティ . . . 328
47.1.12 トポロジスプレッドコンストレイン . . . 328
47.1.13 コマンドラインインターフェイス . . . 328
47.1.14 クラスターの現在のステータス . . . 328
47.1.15 オペレーターの認証局 . . . 329

47.1.16 クラスターの認証局 . . . 329
47.1.17 TLS 接続 . . . 329
47.1.18 ストリーミングレプリケーションの証明書認証 . . . 329
47.1.19 継続的な構成管理 . . . 329
47.1.20 既存の PostgreSQL データベースのインポート . . . 329
47.1.21 PostGIS クラスター . . . 330
47.1.22 PostgreSQL の基本的な LDAP 認証 . . . 330
47.1.23 複数のインストール方法 . . . 330
47.1.24 構成より規約 . . . 330
47.2 レベル 2 シームレスなアップグレード . . . 330
47.2.1 オペレーターのアップグレード . . . 331
47.2.2 管理対象ワークロードのアップグレード . . . 331
47.2.3 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 . . . 331
47.3 レベル 3 完全なライフサイクル . . . 331
47.3.1 PostgreSQL WAL アーカイブ . . . 332
47.3.2 PostgreSQL のバックアップ . . . 332
47.3.3 スタンバイからのバックアップ . . . 333
47.3.4 バックアップからの完全な復元 . . . 333
47.3.5 バックアップからのポイント インタイム リカバリー PITR . . . 333
47.3.6 同期レプリケーションを介したデータ損失ゼロのクラスター . . . 333
47.3.7 レプリカクラスター . . . 334
47.3.8 テーブルスペースのサポート . . . 334
47.3.9 Liveness および readness プローブ . . . 334
47.3.10 ローリング展開 . . . 335
47.3.11 レプリカのスケールアップとスケールダウン . . . 335
47.3.12 Kubernetes ノードのメンテナンスウィンドウと PodDisruptionBudget . . . 335
47.3.13 フェンシング . . . 335
47.3.14 ハイバネーション宣言的 . . . 336
47.3.15 ハイバネーション (必須) . . . 336
47.3.16 ポッドの永続ボリュームストレージの再利用 . . . 336
47.3.17 CPU とメモリの要求と制限 . . . 336
47.3.18 PgBouncer を使用した接続プーリング . . . 336
47.4 レベル 4 深い洞察 . . . 337
47.4.1 構成可能なクエリを備えた Prometheus エクスポータ . . . 337
47.4.2 Grafana ダッシュボード . . . 337
47.4.3 JSON 形式の PostgreSQL エラーメッセージの標準出力ログ . . . 337
47.4.4 リアルタイムクエリモニタリング . . . 337
47.4.5 監査 . . . 338
47.4.6 Kubernetes イベント . . . 338
47.5 レベル 5 自動操縦 . . . 338
47.5.1 自己修復のための自動フェイルオーバー . . . 338
47.5.2 スタンバイの自動再作成 . . . 338

第 48 章 カスタムポッドコントローラ **339**
48.1 PVC のサイズ変更 . . . 339
48.2 プライマリインスタンスとレプリカ . . . 339
48.3 PVC のコヒーレンス . . . 340

48.4 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 340

第 49 章 例 **343**
49.1 基本 343
49.2 バックアップ 351
49.3 レプリカクラスター 353
49.4 PostGIS 359
49.5 管理対象ロール 359
49.6 宣言的テーブルスペース 360

第 50 章 ネットワーキング **363**
50.1 オペレーターのクロス名前空間ネットワークポリシー 363
50.2 クロスクラスターネットワーキング 364

第 51 章 ベンチマーク **365**
51.1 pgbench 365
51.2 fio 367

第 52 章 コマーシャルサポート **369**

第 53 章 よくある質問 FAQ **371**
53.1 Kubernetes での PostgreSQL の実行 371
53.2 高可用性 374

第 54 章 リソースタイプ **375**

第 55 章 Backup **377**

第 56 章 クラスター **379**

第 57 章 プーラー **381**

第 58 章 ScheduledBackup **383**

第 59 章 AffinityConfiguration **385**

第 60 章 AzureCredentials **387**

第 61 章 BackupConfiguration **389**

第 62 章 BackupMethod **391**

第 63 章 BackupPhase **393**

第 64 章 BackupSnapshotElementStatus **395**

第 65 章 BackupSnapshotStatus **397**

第 66 章 BackupSource **399**

第 67 章 BackupSpec **401**

第 68 章 BackupStatus **403**

第 69 章 BackupTarget 405

第 70 章 BarmanCredentials 407

第 71 章 BarmanObjectStoreConfiguration 409

第 72 章 BootstrapConfiguration 411

第 73 章 BootstrapInitDB 413

第 74 章 BootstrapPgBaseBackup 415

第 75 章 BootstrapRecovery 417

第 76 章 CertificatesConfiguration 419

第 77 章 CertificatesStatus 421

第 78 章 ClusterSpec 423

第 79 章 ClusterStatus 427

第 80 章 CompressionType 429

第 81 章 ConfigMapKeySelector 431

第 82 章 ConfigMapResourceVersion 433

第 83 章 DataBackupConfiguration 435

第 84 章 DataSource 437

第 85 章 DatabaseRoleRef 439

第 86 章 EmbeddedObjectMetadata 441

第 87 章 EncryptionType 443

第 88 章 EnsureOption 445

第 89 章 Ephemeral VolumesSizeLimitConfiguration {#postgresql-cnpg-io-v1-Ephemeral VolumesSize-LimitConfiguration} 447

第 90 章 外部クラスター 449

第 91 章 GoogleCredentials 451

第 92 章 Import 453

第 93 章 ImportSource 455

第 94 章 インスタンス ID 457

第 95 章 InstanceReportedState 459

第 96 章 LDAPBindAsAuth 461

第 97 章 LDAPBindSearchAuth 463

第 98 章 LDAPConfig 465

第 99 章 LDAPScheme 467

第 100 章LocalObjectReference 469

第 101 章ManchedConfiguration 471

第 102 章ManchedRoles 473

第 103 章メタデータ 475

第 104 章MonitoringConfiguration 477

第 105 章NodeMaintenanceWindow 479

第 106 章OnlineConfiguration 481

第 107 章PasswordState 483

第 108 章PgBouncerIntegrationStatus 485

第 109 章PgBouncerPoolMode 487

第 110 章PgBouncerSecrets 489

第 111 章PgBouncerSpec 491

第 112 章PodTemplateSpec 493

第 113 章PodTopologyLabels 495

第 114 章PoolerIntegrations 497

第 115 章PoolerMonitoringConfiguration 499

第 116 章PoolerSecrets 501

第 117 章PoolerSpec 503

第 118 章PoolerStatus 505

第 119 章PoolerType 507

第 120 章PostInitApplicationSQLRefs 509

第 121 章PostgresConfiguration 511

第 122 章PrimaryUpdateMethod 513

第 123 章PrimaryUpdateStrategy 515

第 124 章RecoveryTarget 517

第 125 章ReplicaClusterConfiguration **519**

第 126 章ReplicationSlotsConfiguration **521**

第 127 章ReplicationSlotsHAConfiguration **523**

第 128 章RoleConfiguration **525**

第 129 章S3Credentials **527**

第 130 章ScheduledBackupSpec **529**

第 131 章ScheduledBackupStatus **531**

第 132 章SecretKeySelector **533**

第 133 章SecretVersion **535**

第 134 章SecretsResourceVersion **537**

第 135 章ServiceAccountTemplate **539**

第 136 章SnapshotOwnerReference **541**

第 137 章SnapshotType **543**

第 138 章StorageConfiguration **545**

第 139 章SyncReplicaElectionConstraints **547**

第 140 章TablespaceConfiguration **549**

第 141 章TablespaceState **551**

第 142 章TablespaceStatus **553**

第 143 章トポロジ **555**

第 144 章VolumeSnapshotConfiguration {#postgresql-cnpg-io-v1- VolumeSnapshotConfiguration} **557**

第 145 章WalBackupConfiguration **559**

第 146 章サポートされているリリース **561**

146.1 サポートポリシー 561
146.2 命名スキーム 562
146.3 CloudNativePG リリースのサポートステータス 562
146.3.1 サポートされている PostgreSQL バージョン 563
146.4 今後のリリース 563
146.5 古いリリース 564
146.6 サポートとは 564

第 147 章Release notes **567**

147.1 CloudNativePG 1.22 のリリースノート 567
147.1.1 バージョン 1.22.1 567

147.1.2 バージョン 1.22.0 . . . 568
147.2 CloudNativePG 1.21 のリリースノート . . . 570
147.2.1 バージョン 1.21.3 . . . 570
147.2.2 バージョン 1.21.2 . . . 571
147.2.3 バージョン 1.21.1 . . . 572
147.2.4 バージョン 1.21.0 . . . 573
147.3 CloudNativePG 1.20 のリリースノート . . . 576
147.3.1 バージョン 1.20.6 . . . 576
147.3.2 バージョン 1.20.5 . . . 577
147.3.3 バージョン 1.20.4 . . . 578
147.3.4 バージョン 1.20.3 . . . 578
147.3.5 バージョン 1.20.2 . . . 580
147.3.6 バージョン 1.20.1 . . . 581
147.3.7 バージョン 1.20.0 . . . 582
147.4 1.14.0 以前のリリースノート . . . 583
147.4.1 バージョン 1.14.0 . . . 583
147.4.2 バージョン 1.13.0 . . . 584
147.4.3 バージョン 1.12.0 . . . 586
147.4.4 バージョン 1.11.0 . . . 587
147.4.5 バージョン 1.10.0 . . . 588
147.4.6 バージョン 1.9.2 . . . 590
147.4.7 バージョン 1.9.1 . . . 590
147.4.8 バージョン 1.9.0 . . . 590
147.4.9 バージョン 1.8.0 . . . 591
147.4.10バージョン 1.7.1 . . . 592
147.4.11バージョン 1.7.0 . . . 592
147.4.12バージョン 1.6.0 . . . 593
147.4.13バージョン 1.5.1 . . . 594
147.4.14バージョン 1.5.0 . . . 594
147.4.15バージョン 1.4.0 . . . 595
147.4.16バージョン 1.3.0 . . . 596
147.4.17バージョン 1.2.1 . . . 597
147.4.18バージョン 1.2.0 . . . 597
147.4.19バージョン 1.1.0 . . . 597
147.4.20バージョン 1.0.0 . . . 598
147.5 Release notes old version . . . 598
147.5.1 CloudNativePG 1.19 のリリースノート . . . 598
147.5.2 CloudNativePG 1.18 のリリースノート . . . 606
147.5.3 CloudNativePG 1.17 のリリースノート . . . 612
147.5.4 CloudNativePG 1.16 のリリースノート . . . 617
147.5.5 CloudNativePG 1.15 のリリースノート . . . 622

第 148 章Appendix A **629**
148.1 付録 A-バックアップ用の共通オブジェクトストア . . . 629
148.1.1 AWS S3 . . . 629
148.1.2 Azure Blob Storage . . . 632
148.1.3 Google Cloud Storage . . . 634

148.1.4 MinIO ゲートウェイ 635

CloudNativePG は、オープンソース operator

プライベート、パブリック、ハイブリッド、またはマルチクラウド環境で実行されているサポートされている Kubernetes クラスターで PostgreSQL ワークロードを管理するように設計されています。CloudNativePG は、宣言的構成や不変のインフラストラクチャなどの DevOps の原則と概念を遵守しています。

`Cluster` と呼ばれる新しい Kubernetes リソースを定義します。高可用性と読み取り専用クエリのオフロードのために、選択した Kubernetes 名前空間に共存する単一のプライマリとオプションの数のレプリカで構成される PostgreSQL クラスターを表します。

同じ Kubernetes クラスターにあるアプリケーションは、オペレーターによってのみ管理されるサービスを使用して PostgreSQL データベースにアクセスでき、フェールオーバーまたはスイッチオーバーに続くプライマリロールの変更を心配する必要はありません。Kubernetes クラスターの外部にあるアプリケーションは、TCP 経由で Postgres を公開するように Service または Ingress オブジェクトを構成する必要があります。Web アプリケーションは、PgBouncer に基づいたネイティブ接続プーラーを利用できます。

CloudNativePG は、もともと EDB によってビルドされ、その後、Apache License 2.0 の下でオープンソースをリリースし、2022 年 4 月に CNCF サンドボックスに提出されました。

> source code repository is in Github 。

注釈: *Operator Capability Levels model* に基づいて、ユーザーは CloudNativePG オペレーターからの**「レベル V - 自動パイロット」**機能セットを期待できます。

第1章　はじめに

1.1 サポートされている Kubernetes ディストリビューション

CloudNativePG の各マイナーリリースは、さまざまな Kubernetes バージョン、通常、マイナーバージョンが最初にリリースされたときに CNCF によってサポートされているバージョンで動作するように設計されています。

詳細は **サポートされているリリース** ページを参照してください。

1.2 コンテナイメージ

> CloudNativePG community は、演算子とオペランドの両方、つまり PostgreSQL のコンテナイメージを維持します。

CloudNativePG オペレーターコンテナイメージは distroless です

> cloudnative-pg で利用できます。

PostgreSQL オペランドコンテナイメージは、複数のアーキテクチャでのすべての PGDG supported versions of PostgreSQL で
> postgres-containers から直接利用できます。

さらに、コミュニティは *PostGIS extension* のイメージを提供しています。

> **警告:** CloudNativePG 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ことを必要とします。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。

1.3 主な機能

*外部ツールを必要とせず、高可用性を実現する Kubernetes API サーバーとの直接統合

以下を介した自己修復機能最も調整されたレプリカをプロモーションす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*レプリカの自動再作成

- 選択したレプリカの昇格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

*スケールアップ/ダウン機能

- 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

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

ワークロードを読み取るためのインスタンスにアプリケーションを接続するための読み取り専用*サービスの定義

- クラスターを介した特定の人気の Postgres 拡張機能を含む PostgreSQL 構成の宣言的管理 `spec` `pgaudit`、`auto_explain`、`pg_stat_statements`、および `pg_failover_slots`
- Postgres のロール、ユーザー、グループの宣言的管理
- 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

*ポッドの永続ボリュームストレージの再利用

- WAL ファイルとテーブルスペースの個別のボリューム
- 一時テーブルスペースを含む Postgres テーブルスペースの宣言的管理
- PostgreSQL マイナーバージョンのローリングアップデート

*オペレーターアップグレードのインプレースまたはローリング更新

- TLS 接続とクライアント証明書認証

*カスタム TLS 証明書のサポート cert-manager との統合を含む

*オブジェクトストアへの継続的な WAL アーカイブ AWS S3 および S3 互換、Azure Blob Storage、および Google Cloud Storage

*ボリュームスナップショットのバックアップ基になるストレージクラスでサポートされている場合

*オブジェクトストアのバックアップ AWS S3 および S3 互換、Azure Blob Storage、および Google Cloud Storage

- 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の既存のバックアップからのフルリカバリーおよびポイントインタイムリカバリー
- PostgreSQL のメジャーアップグレードを含む、既存の PostgreSQL データベースのオフラインインポート

*宣言的な方法での PostgreSQL クラスター全体、またはインスタンスのサブセットのフェンシング

*宣言的な方法での PostgreSQL クラスターの休止状態

*同期レプリカのサポート

- クラスターレベルでの HA 物理レプリケーションスロットのサポート
- スタンバイからのバックアップ

*バックアップ保持ポリシーリカバリウィンドウに基づいて、オブジェクトストアのみ

*並列 WAL アーカイブとリストアにより、データベースが大量の書き込みシステムでの WAL 生成に追いつくことができます

*オブジェクトストアにアップロードされたバックアップファイルのタグ付けをサポートして、オブジェクトストアレイヤーでオプションの保持管理を有効にする

- 複数の Kubernetes クラスターにわたる PostgreSQL の展開、プライベート、パブリック、ハイブリッド、マルチクラウドアーキテクチャを有効にします
- PgBouncer を使用した接続プーリング
- `nodeSelector` を介したノードアフィニティのサポート
- `metrics` ポート 9187 を介したプロメテウスのユーザー定義メトリックのネイティブなカスタマイズ可能なエクスポーター
- JSON 形式の PostgreSQL エラーメッセージの標準出力ログ

*ポッドの `readOnlyRootFilesystem` セキュリティコンテキストを自動的に設定します

- `kubectl` 用の `cnpg` プラグイン
- 単純なバインドおよび検索+バインドの LDAP クライアント認証
- マルチアーチフォーマットのコンテナイメージ
- OLM のインストール

注釈: CloudNativePG は、データの永続性の管理に `StatefulSet` s を使用しません。代わりに、永続ボリューム要求 PVC を直接管理します。興味がある場合は、 **カスタムポッドコントローラ** を読んで詳細をご覧ください。

1.4 このガイドについて

クイックスタート 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 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、
始める前に を参照してください。

- 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。*

第2章　始める前に

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。

2.1 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。

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 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 サポートされているリリース ページを参照してください。

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 *Cluster* 。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 は、ローカルの Unix ドメインソケットを介して *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**
: CloudNativePG の用語における 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 のボリューム *walStorage* を含むメインボリュームと WAL のボリューム *walStorage* に属する関連 PVC のグループです。

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** 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。

2.4 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 CloudNativePG on your laptop using a local cluster* を決定できます。

第3章　ユースケース

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

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

3.1 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。

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 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。

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 リソースに対応する IP アドレスまたはホスト名と TCP ポートで表されます。

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

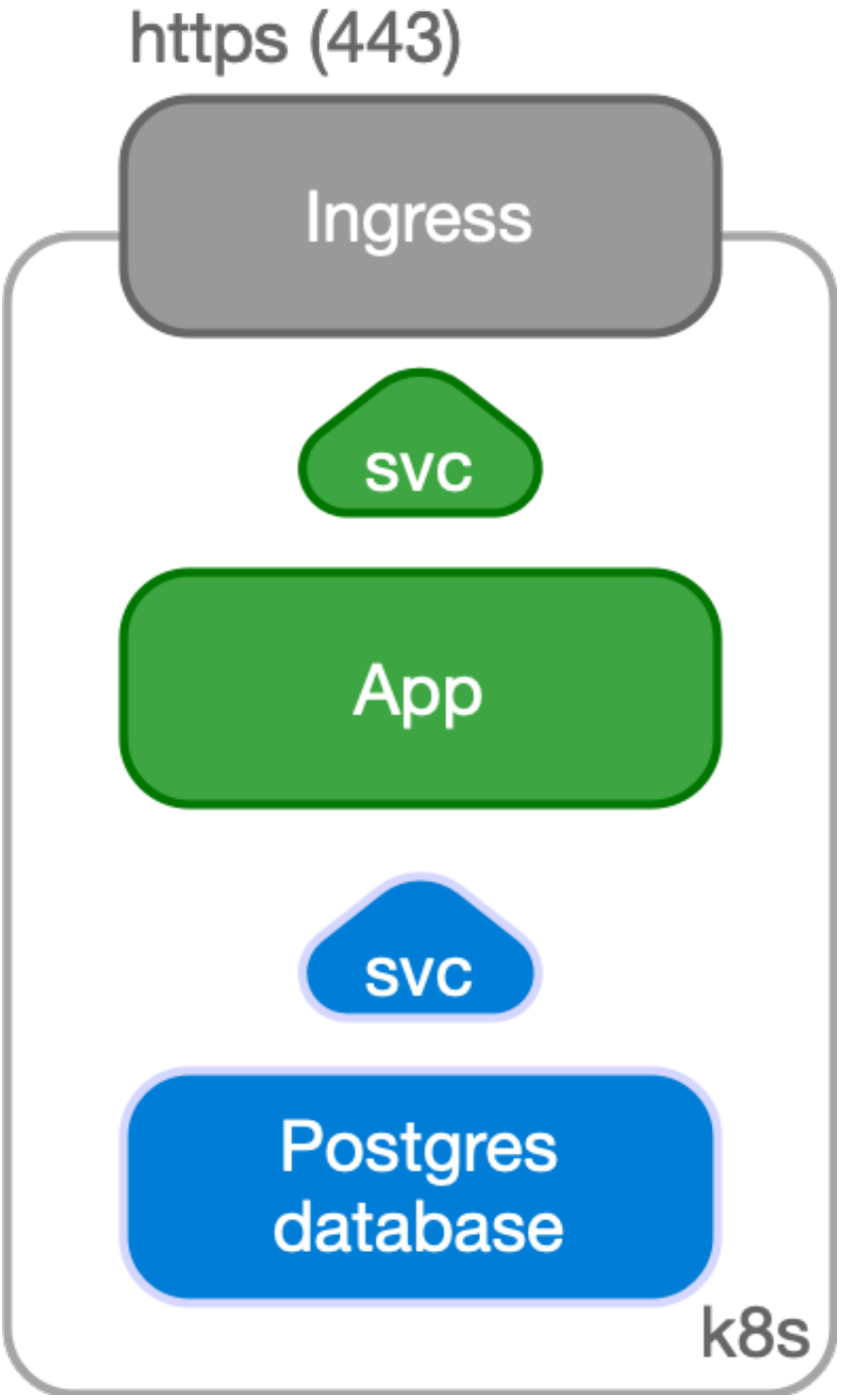


図 1: Application and Database inside Kubernetes

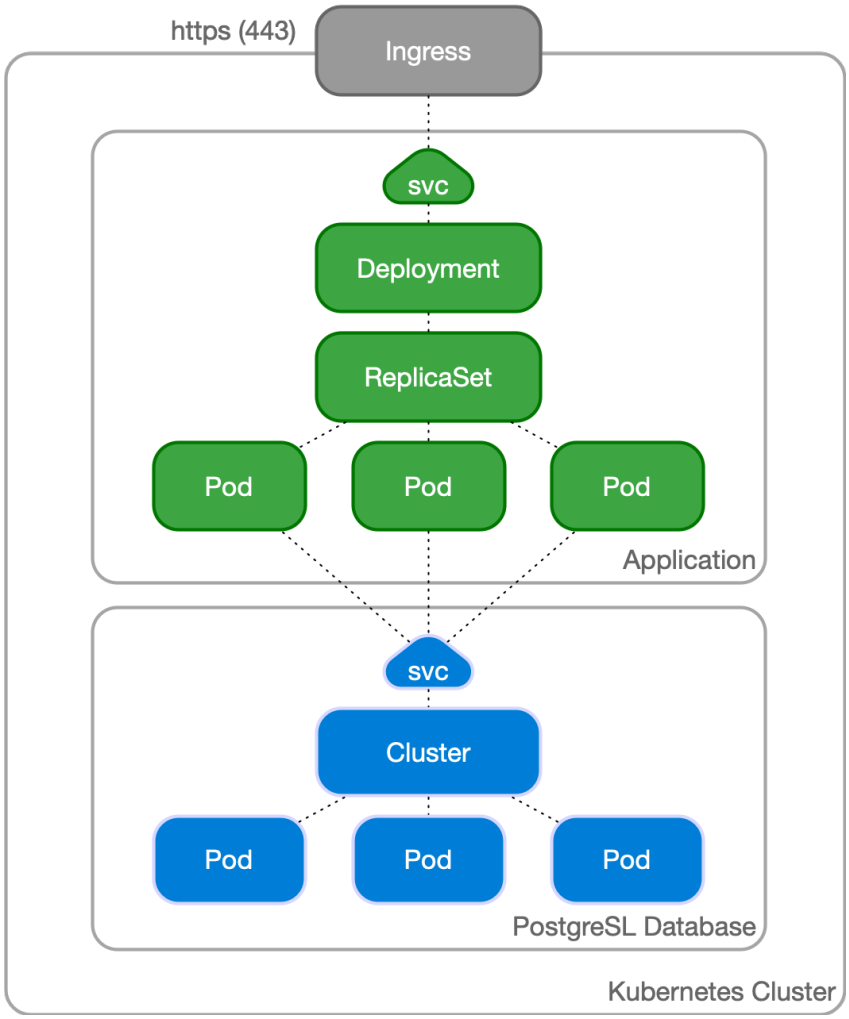


図 2: Close-up view of application and database inside Kubernetes

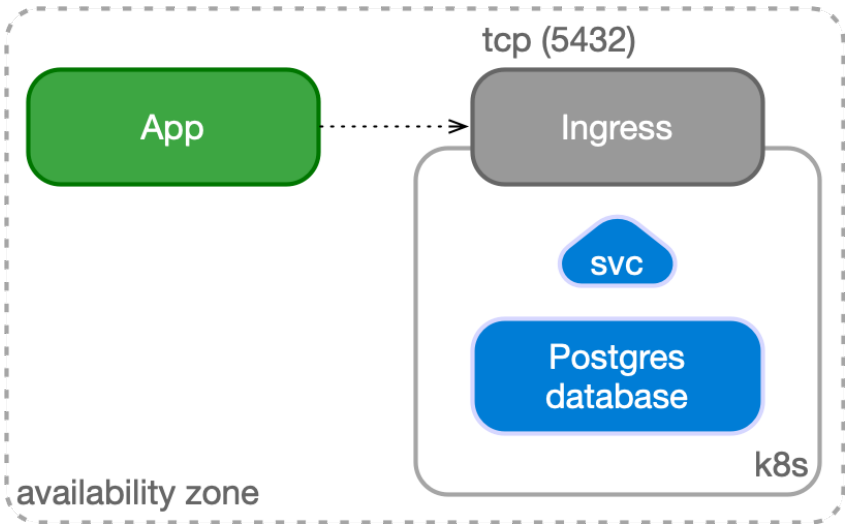


図 3: Application outside Kubernetes

第4章　アーキテクチャ

このセクションでは、Kubernetes に PostgreSQL を展開するときに考慮する必要がある主なアーキテクチャの側面について説明します。

重要: CNCF ブログ用に書いたタイトル *Recommended Architectures for PostgreSQL in Kubernetes* の記事を読むことをお勧めします。

重要: 自己管理の Kubernetes 環境に PostgreSQL を展開する場合、 *Kubernetes* **アーキテクチャ**

クラウドネイティブの世界への旅行を計画し始めるときは、以下を参照してください。

4.1 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 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、**ログの先行書き込み WAL 配送** に基づいた堅牢で信頼性の高いビルトイン**物理レプリケーション** 機能が付属しています。10 年以上、世界中の何百万ものユーザーが実稼働環境で使用していました。

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はお勧めしません**が、CloudNativePG 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。

4.2 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。

4.2.1 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。

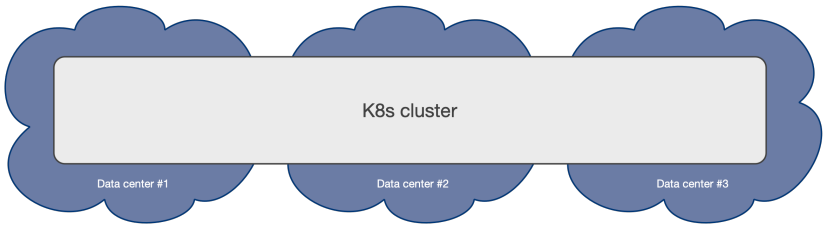


図 1: Kubernetes cluster spanning over 3 independent data centers

このようなアーキテクチャにより、CloudNativePG オペレーターは、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` リソースのライフサイクル全体を制御できます。これには、他の操作の中で、宣言的方法でワークロードの **スケジューリング** が含まれますアフィニティルール、トレランス、ノードセレクターなど)、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、追加の **マルチアベイラビリティーゾーン** *Kubernetes* **クラスター**を使用して「パッシブ」PostgreSQL レプリカクラスターをホストすることにより、それを利用できます。この場合のフェイルオーバーとプロ

モーションは手動で行う必要がありますが、これは主に DR、読み取り専用操作、またはクロスリージョンの可用性のために使用する必要があります。

4.2.2 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 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、可能な限り単一障害点 SPoF をゾーンのレベルにプッシュします。つまり、CloudNativePG 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 は、残念ながら、低遅延接続の届く範囲内で **2 つのデータセンター**のみが利用できる唯一の実行可能なオプションです。通常は同じ大都市圏。ゾーンが 2 つだけであると、ユーザーはマルチプルを作成できません。-可用性ゾーンの Kubernetes クラスター最小数の 3 ゾーンに到達しないため、アクティブ/パッシブ構成で 2 つの異なる Kubernetes クラスターを作成することを強制します。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ます。

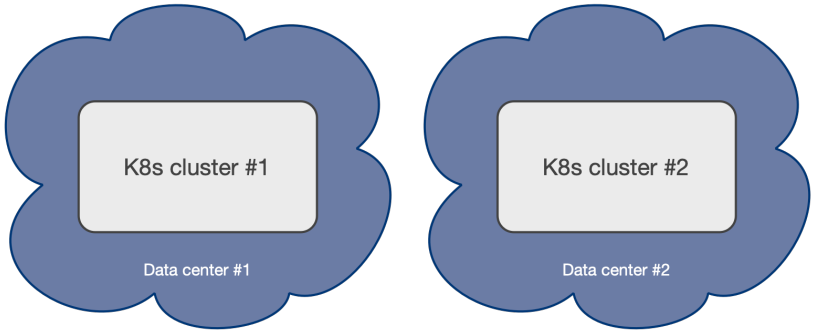


図 2: Example of a Kubernetes architecture with only 2 data centers

---

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 **マルチアベイラビリティーゾーン *Kubernetes* クラスター** を使用して「パッシブ」PostgreSQL レプリカクラスターをホストすることにより、SPoF を単一ゾーンより上にプッシュできます。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。以下で説明するように、オペレーターは単一の Kubernetes クラスター内でのみ作業できるため、現時点では CloudNativePG を使用する PostgreSQL では利用できません。

4.3 PostgreSQL アーキテクチャ

CloudNativePG 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 ``-ro`` **アプリケーションは読み取り専用ワークロード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ワークロードのインスタンスに接続します

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。で実行されます* 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

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

CloudNativePG 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 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、 **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。

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 に接続する方法については、 **アプリケーションから接続する** 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、 **接続プーリング** を参照してください。

4.3.1 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**現在のプライマリ**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。

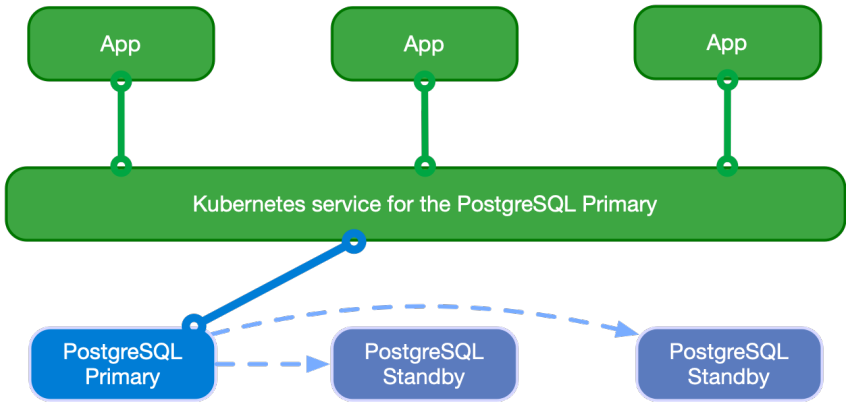


図 3: Applications writing to the single primary

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、高可用性の目的で、CloudNativePG 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。

4.3.2 読み取り専用ワークロード

重要: アプリケーションは、次の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。

4.4 Kubernetes クラスター全体のデプロイメント

注釈: CloudNativePG は、このセクションで説明する**レプリカクラスター**と呼ばれる機能を介して、複数の Kubernetes クラスターへの PostgreSQL の展開をサポートしています。

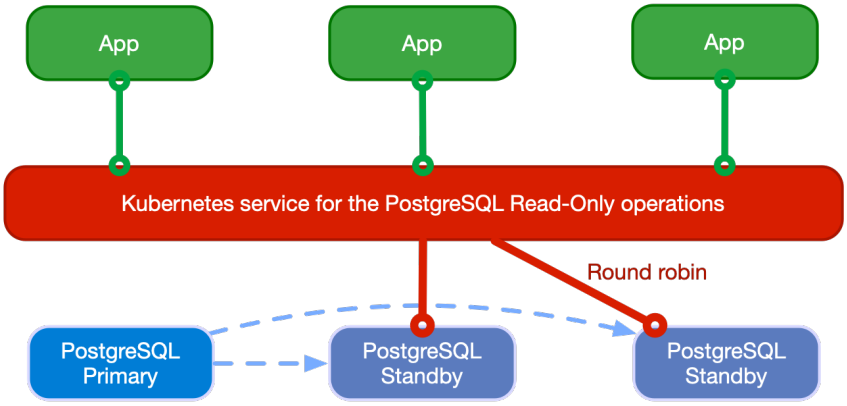


図 4: Applications reading from hot standby replicas in round robin

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、グローバルな**リカバリポイント目標** RPO 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、グローバルな**目標復旧時間** RTO を削減します。

上記の懸念事項に対処するために、CloudNativePG は *PostgreSQL* **レプリカクラスター**の概念を導入しています。レプリカクラスターは、プライベート、パブリック、ハイブリッド、マルチクラウドのコンテキストでマルチクラスター展開を有効にする CloudNativePG 方法です。

レプリカクラスターは別個の `Cluster` リソースです。

1. 定義された外部ソースクラスターからの `bootstrap` オプションとして `pg_basebackup` または完全な `recovery` のいずれかを持つ
2. `replica.enabled` オプションを `true` に設定する
3. `replica.source` で指定される定義された外部クラスターからの複製。通常は Kubernetes クラスターの外部にあります
4. リカバリオブジェクトストア PostgreSQL の `restore_command` パラメーターを使用して、またはストリーミングレプリケーションを介して PostgreSQL の `primary_conninfo` パラメーターを使用、または 2 つのいずれかから受け取った WAL 情報の再生 `barmanObjectStore` と `connectionParameters` の両方が外部クラスターで定義されている場合
5. PostgreSQL のホットスタンバイでサポートされている読み取り接続のみの受け入れ

参考:

別のクラスター `externalClusters` セクションで定義からの PostgreSQL クラスターのクローン作成の詳細については、 *Bootstrap* を参照してください。

次の図は、2 つの異なる Kubernetes クラスターにまたがる PostgreSQL クラスターを示しています。プライ

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

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

指定されたプライマリは、いつでも昇格でき、レプリカクラスターを、書き込み接続を受け入れることができるプライマリクラスターにすることができます。

> **警告:** CloudNativePG は、現時点ではクラスター間のスイッチオーバーまたはフェイルオーバーを実行しません。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当局に委任する必要があります。各 PostgreSQL クラスターは、他から独立しています。

上記の例で指定されたプライマリは、WAL ストリーミング `primary_conninfo` を介してフィードされ、`restore_command` および `barman-cloud-wal-restore` を介したファイルベースの WAL 配送のフォールバックオプションを使用します。

CloudNativePG では、複数のレプリカクラスターを定義できます。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の動作と、さまざまな Kubernetes クラスターで読み取り専用クラスターを構成して、グローバルディザスターリカバリーと HA 戦略を向上させ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。

第5章　インストールとアップグレード

5.1 Kubernetes へのインストール

5.1.1 オペレーターマニフェストを直接使用する

オペレーターは、Kubernetes の他のリソースと同様に、kubectl を介して適用される YAML マニフェストを介してインストールできます。

latest operator manifest をインストールできます

このマイナーリリースでは次のようになります。

```
kubectl apply -f \
  https://raw.githubusercontent.com/cloudnative-pg/cloudnative-pg/release-1.22/
↪releases/cnpg-1.22.1.yaml
```

これは次の方法で確認できます。

```
kubectl get deployment -n cnpg-system cnpg-controller-manager
```

5.1.2 kubectl の cnpg プラグインを使用する

cnpg 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 cnpg install generate \
  --watch-namespace "specific-namespace" \
  > cnpg_for_specific_namespace.yaml
```

より包括的な例については、 Backup {postgresql-cnpg-io-v1-Backup} ドキュメントを参照してください。

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

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。

5.1.3 最新の開発スナップショットのテスト

次の公式パッチリリースの前に CloudNativePG の最新の開発スナップショットをテストまたは評価する場合は、 cloudnative-pg/artifacts からマニフェストをダウンロードできます。

これにより、現在のトランクメインおよびサポートされている各リリースに簡単にアクセスできます。

たとえば、次の使用でオペレーターの最新のスナップショットをインストールできます。

```
curl -sSfL \
  https://raw.githubusercontent.com/cloudnative-pg/artifacts/main/manifests/operator-
↪manifest.yaml | \
  kubectl apply -f -
```

代わりに、この特定のマイナーリリースのオペレーターの最新のスナップショットを探している場合は、次を実行できます。

```
curl -sSfL \
  https://raw.githubusercontent.com/cloudnative-pg/artifacts/release-1.22/manifests/
↪operator-manifest.yaml | \
  kubectl apply -f -
```

重要: スナップショットは CloudNativePG ではサポートされておらず、本番使用を目的としていません。

5.1.4 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。

5.1.5 OLM を使用する

CloudNativePG は、 Operator Lifecycle Manager (OLM) を使用してインストールすることもできます

OperatorHub.io から直接。

5.2 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 cnpg-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 cnpg 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 cnpg-controller-manager と呼ばれます。Helm を介してインストールされた場合、デフォルト名は cnpg-cloudnative-pg です。

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 cnpg-system
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
<deployment-name>   1/1     1            1           18m
```

```
kubectl describe deploy \
  -n cnpg-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 オペレーター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、 **オペレーター設定** セクションを参照してください。

5.3 アップグレード

重要: リリースノート 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。

> **警告:** バージョン 1.20 にアップグレードする場合は、 dedicated section below を注意深くお読みください。

CloudNativePG オペレーター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、コントローラを更新した後に自動的に実行され、デフォルトでは、新しいインスタンスマネージャーを使用するように展開されたすべての PostgreSQL インスタンスのローリング更新をトリガーします。ローリング更新手順は、スイッチオーバーで最高潮に達します。これは、`primaryUpdateStrategy` オプションで制御されます。デフォルトでは `unsupervised` に設定されます。 `supervised` に設定している場合、ユーザーは `kubectl` の `cnpg` プラグインを介して新しいインスタンスを手動で昇格させて、ローリング更新を完了する必要があります。

参考:

このプロセスは、 ローリングアップデート 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。

バージョン 1.10.0 以降、ローリング更新動作を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。後者は、PostgreSQL インスタンスの再起動、その結果、クラスター内のスイッチオーバーを必要としません。デフォルトで無効になっているこの動作について以下に説明します。

5.3.1 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 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 のバージョン 1.10 からのインスタンスマネージャーは、完全性検証フェーズが完了すると、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と、すべての内部プロセスの正常な終了をサポートしています。新しいインスタンスマネージャーが新しいバイナリを使用して再起動すると、既に実行されている *postmaster* が採用されます。

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。

> オペレーター設定 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。

重要: この機能では、すべてのポッド演算子とオペランドが同じプラットフォーム/アーキテクチャたとえば、すべて `linux/amd64` で実行される必要があります。

5.3.2 バージョン間の互換性

CloudNativePG 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.x.y 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 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。

> リリースノート ページには、CloudNativePG 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。

5.3.3 1.22.0、1.21.2 または 1.20.5 へのアップグレード

重要: CloudNativePG の既存のすべてのユーザーは、バージョン 1.22.0 または少なくとも現在使用しているマイナーリリースの最新の安定バージョン 1.21.2 または 1.20.5 にアップグレードすることをお勧めします。

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、1.20.x から 1.22 に移行する場合は、1.21 および 1.22 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。

CloudNativePG は、デフォルトでのセキュリティのアプローチを遵守し続けます。バージョン 1.22 以降、`ALTER SYSTEM` コマンドの使用はデフォルトで無効になりました。

この選択の背後にある理由は、デフォルトで、CloudNativePG によって制御されるデータベースクラスターの PostgreSQL 構成の変更が、Kubernetes API を介してのみ許可されるようにするためです。

同時に、バージョン 1.22.0、1.21.2、および 1.20.5 から一時的にでも使用する必要がある場合に、次の抜粋のように`.spec.postgresql.enableAlterSystem` を `true` に設定することにより、`ALTER SYSTEM` を有効にするオプションを提供しています。

```
...
  postgresql:
    enableAlterSystem: true
...
```

1. 22 のクラスターでは、デフォルトで `enableAlterSystem` が `false` に設定されます。1.22 で既存の動作を保持したい場合は、上記のように `enableAlterSystem` を `true` に明示的に設定する必要があります。

バージョン 1.21.2 および 1.20.5、および 1.20 および 1.21 ブランチの以降のパッチリリースでは、既存の動作を維持しながら、`enableAlterSystem` はデフォルトで `true` に設定されます。 `ALTER SYSTEM` を使用する必要がない場合は、 `enableAlterSystem` を明示的に `false` に設定することをお勧めします。

重要: 上記の例に示すように、バージョン 1.22.0、1.21.2、または 1.20.5 へのアップグレードの直後に `enableAlterSystem` の値を設定できます。

5.3.4 以前のマイナーバージョンから 1.21 へのアップグレード

オペレーターの **構成よりコンベンション**動作をすぐに改善し続けることを目的に、CloudNativePG は次の領域でいくつかのノブのデフォルト値を変更します。

- PostgreSQL インスタンスの起動およびシャットダウン制御
- 自己修復
- セキュリティ
- ラベル

警告: 以下の変更のリスト、および既存のワークロードを中断したくない場合は、既存の動作を保持するように `Cluster` マニフェストを変更する方法を注意深くお読みください。または、確認するまでアップグレードを延期します。一般に、正当な理由がない限り、これらのデフォルト値を採用することをお勧めします。

5.3.5 スーパーユーザーアクセスは無効になっています

デフォルトのセキュリティに向けて、CloudNativePG は、明示的に有効になっていない限り、すべての新しいクラスターのネットワークを介した `postgres` スーパーユーザーアクセスを無効にします。

実行している CloudNativePG のバージョンに関係なく、PostgreSQL クラスターへのスーパーユーザーアクセスを保証したい場合は、次の設定で明示的に宣言することをお勧めします。

```
spec:
  ...
  enableSuperuserAccess: true
```

5.3.6 HA のレプリケーションスロット

高可用性のレプリケーションスロットはデフォルトで有効になっています。

実行している CloudNativePG のバージョンに関係なく、レプリケーションスロットを無効にしたい場合は、次の設定で明示的に宣言することをお勧めします。

```
spec:
  ...
  replicationSlots:
    highAvailability:
      enabled: false
```

5.3.7 PostgreSQL のシャットダウンの遅延

1. 20.2、 the `stopDelay parameter <Postgres インスタンスマネージャー>` まで

30 秒に設定されました。この値を変更して調整するという推奨事項にもかかわらず、サポートインシデントまたはコミュニティの問題中に調べたほぼすべてのケースは、この値が変更されないままであることを示しています。

> new default value is 1800 seconds 、30 分に相当。

新しい `smartShutdownTimeout` パラメーターは、PostgreSQL の `smart` シャットダウンプロシージャを完了するために予約されている `stopDelay` 値内の最大時間ウィンドウを定義するために導入されました。この間、Postgres サーバーは、すべての通常のセッションが終了するのを待機している間、新しい接続を拒否します。

経過すると、 `stopDelay` までの残り時間は PostgreSQL がアーカイブとストリーミングレプリカの両方との WAL コミットメントに関する義務を完了し、クラスターがデータを失わないようにするために予約されます。

古い動作を保持したい場合は、明示的に設定する必要があります。

```
spec:
  ...
  stopDelay: 30
```

そして、 **アップグレードが完了した後** 、 `smartShutdownTimeout` を指定します。

```
spec:
  ...
  stopDelay: 30
  smartShutdownTimeout: 15
```

5.3.8 PostgreSQL の起動の遅延

1. 20.2、 the `startDelay parameter <Postgres インスタンスマネージャー>` まで

は 30 秒に設定され、CloudNativePG はこのパラメーターを Kubernetes liveness プローブの `initialDelaySeconds` として使用しました。サポートされているすべての Kubernetes リリースが startup probes を提供していることを考慮すると、`startDelay` はそれぞれ 10 秒の期間に自動的に分割されます。

重要: *startupProbe* を追加するには、各ポッドを再起動する必要があります。その結果、オペレーターをアップグレードすると、オンライン更新の場合でも、クラスターのワンタイムローリング更新が実行されます。

この値を変更して調整するという推奨事項にもかかわらず、サポートインシデントまたはコミュニティの問題中に調べたほぼすべてのケースは、この値が変更されないままであることを示しています。このパラメーターが PostgreSQL インスタンスの起動に影響を与えることを考えると、`startDelay` の値が低いと、Postgres が一貫した復旧状態に到達せず、無期限に再起動されます。

このため、 `startDelay` は 1 時間に相当する raised by default to 3600 seconds になりました。

新しい実装を使用して既存の動作を保持したい場合は、次のことを明示的に設定することによりそれを行うことができます。

```
spec:
  ...
  startDelay: 30
```

5.3.9 PostgreSQL スイッチオーバーの遅延

1. 20.2、 the `switchoverDelay parameter <Postgres インスタンスマネージャー>` まで

非常に長い間隔をシミュレートするために、デフォルトで 40000000 秒 15 か月以上に設定されました。

> default value has been lowered to 3600 seconds 、1 時間に相当します。

古い動作を保持したい場合は、明示的に設定する必要があります。

```
spec:
   ...
   switchoverDelay: 40000000
```

5.3.10 ラベル

バージョン 1.18 では、クラスターの名前を識別するためにポッドの `postgresql` ラベルを非推奨にし、より正規な `cnpg.io/cluster` ラベルに置き換えました。 `postgresql` ラベルは維持されなくなりました。

同様に、このバージョンから、`role` ラベルは非推奨です。新しいラベル `cnpg.io/instanceRole` が使用されるようになり、将来のリリースでは `role` ラベルを完全に置き換えます。

5.3.11 既存の動作を維持するためのショートカット

CloudNativePG の動作を明示的にバージョン 1.20.2 までに保ちたい場合は、新しいバージョンにアップグレードする前に**、すべての `Cluster` 定義でこれらの値を設定する必要があります。

```
spec:
   ...
   # Changed in 1.21.0, 1.20.3 and 1.19.5
   startDelay: 30
   stopDelay: 30
   switchoverDelay: 40000000
   # Changed in 1.21.0 only
   enableSuperuserAccess: true
   replicationSlots:
     highAvailability:
       enabled: false
```

アップグレードが完了したら、以下も追加します。

```
spec:
   ...
   smartShutdownTimeout: 15
```

5.3.12 以前のマイナーバージョンから 1.20 へのアップグレード

CloudNativePG 1.20 では、構成より規約を通じて、すぐに Postgres クラスターの復元力と使いやすさを向上させることを目的に、いくつかの機能のデフォルト動作にオペレーターの以前のバージョンからの変更を導入します。

重要: これらの変更はすべて、少なくとも 1 つのレプリカが存在する場合が関係し、**新しい `Cluster` リソースにのみ影響します**。

5.3.13 スタンバイからのバックアップ

スタンバイからのバックアップ

CloudNativePG 1.19 で導入されましたが、デフォルトでは無効になっています。つまり、ターゲットがスタンバイを優先するように明示的に設定されていない限り、ベースバックアップはプライマリから取得されます。

バージョン 1.20 から、1 つ以上のレプリカが利用可能な場合、オペレーターは最も調整されたスタンバイを優先して完全なベースバックアップを取得します。

CloudNativePG 展開を 1.20 にアップグレードしており、この機能が、作成する新しい `Cluster` リソースの実稼働環境に影響を与える可能性が懸念される場合は、すべての `Cluster` リソースに次の行を追加することにより、ターゲットを明示的にプライマリに設定できます。

```
spec:
  ...
  backup:
    target: "primary"
```

5.3.14 ローリング更新後のプライマリの再起動

Automated rolling updates

CloudNativePG では常に利用でき、デフォルトでは、最も調整されたレプリカへのスイッチオーバーを実行した後にプライマリを更新します。

バージョン 1.20 から、プライマリのデフォルトの更新方法をスイッチオーバーから再起動に変更します。ほとんどの場合、これが最も速くて安全な方法です。

CloudNativePG 展開を 1.20 にアップグレードしており、この機能が、作成する新しい `Cluster` リソースの実稼働環境に影響を与える可能性が懸念される場合は、すべての `Cluster` ファイルに次の行を追加することにより、プライマリの更新方法をスイッチオーバーに明示的に設定できますリソース

```
spec:
  ...
  primaryUpdateMethod: switchover
```

5.3.15 高可用性のための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

バージョン 1.18 の CloudNativePG に導入されましたが、デフォルトでは無効になっています。

バージョン 1.20 は、レプリケーションスロットが高可用性クラスターの復元力と堅牢性を強化するため、将来のリリースでこの機能をデフォルトで有効にするための基礎を準備します。

将来の互換性のために、環境がレプリケーションスロットを必要としないことがすでにわかっている場合は、`Cluster` リソースに次の行を追加して、管理を明示的に無効にすることをお勧めします。

```
spec:
  ...
  replicationSlots:
    highAvailability:
      enabled: false
```

第6章　クイックスタート

このセクションでは、

Kind または Minikube のローカル Kubernetes クラスターで CloudNativePG を使用して、ラップトップ/コンピューターで PostgreSQL クラスターをテストする方法について説明します。

> **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 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。

このページの手順に従うと、ローカルの Kubernetes 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を確認してください。 *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。

6.1 パート 1: ローカル Kubernetes プレイグラウンドをセットアップする

最初の部分は、Minikube または Kind 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

6.1.1 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 to install 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。

6.1.2 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
```

6.2 パート 2 CloudNativePG のインストール

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 のインストールに進むことができます。

> インストールとアップグレード セクションを参照して、PostgreSQL クラスターの展開を続行してください。

6.3 パート 3 PostgreSQL クラスターを展開する

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。

は

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

  storage:
    size: 1Gi
```

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.cnpg.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3

  storage:
    size: 1Gi
```

3 ノード 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 cnpg.io/cluster ラベルを適用します。例

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

重要: ラベルとして `cnpg.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。

cngp.io/cluster を使用してください。

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 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。例、PostgreSQL 13.6 をインストールするには

```
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  # [...]
spec:
  # [...]
  imageName: ghcr.io/cloudnative-pg/postgresql:13.6
  #[...]
```

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。

6.4 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 クラスターを監視するための Grafana ダッシュボードを作成する方法、およびアラート条件を定義する Prometheus ルールセットを示します。

> Kube-Prometheus stack , Helm チャートを利用します。これは、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 チャートの展開のための構成ファイルが含まれており、CloudNativePG クラスターのオブザーバビリティに役立つ初期設定を提供します。

6.4.1 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* の指示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、提供するサンプル構成を使用して *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/cloudnative-pg/cloudnative-pg/main/docs/src/
↪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` ファイルの値でインストールされます。

- Prometheus インストールから、 **any** `PodMonitor` **monitoring** を参照してください。
- Grafana インストールは、Grafana ダッシュボード `ConfigMap` を監視します。

参考:

上記のコマンドの詳細については、 helm install ドキュメントを参照してください。

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
```

6.4.2 Prometheus での表示

この時点で、モニタリングをアクティブ化して展開された CloudNativePG クラスターは、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.cnpg.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/

モニタリングスタックが正常に展開され、enablePodMonitor: true を使用したクラスターがあると仮定すると、CloudNativePG クラスターに関連する一連のメトリックを見つける必要があります。詳細については、

> *monitoring section* を参照してください。

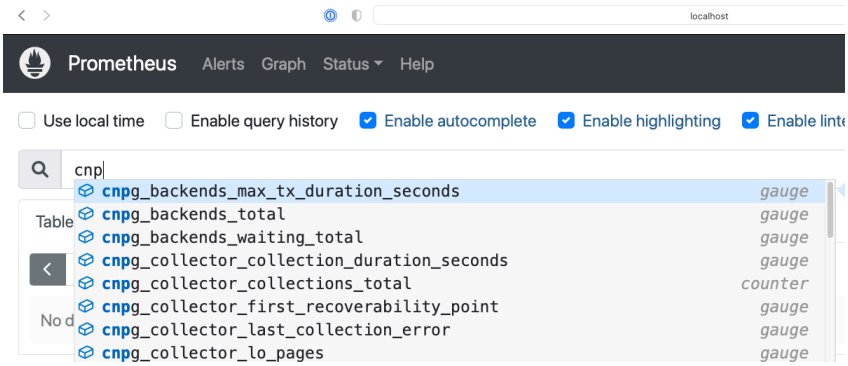


図 1: local prometheus

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るようになりました。

```
kubectl apply -f \
  https://raw.githubusercontent.com/cloudnative-pg/cloudnative-pg/main/docs/src/
↪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
cnpg-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。

6.4.3 Grafana ダッシュボード

「プレーン」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。

ポートフォワードができます。

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

`http://localhost:3000/ <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 は、公式

Helm chart の一部として Grafana のデフォルトのダッシュボードを提供します。*grafana-dashboard.json* をダウンロードできます。

ファイルを作成し、GUI を介して手動でインポートします。

> **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。

この設定例では、Prometheus と Grafana の両方が、モニタリングがアクティブ化された状態で展開された他の CloudNativePG クラスターを選択することに注意してください。

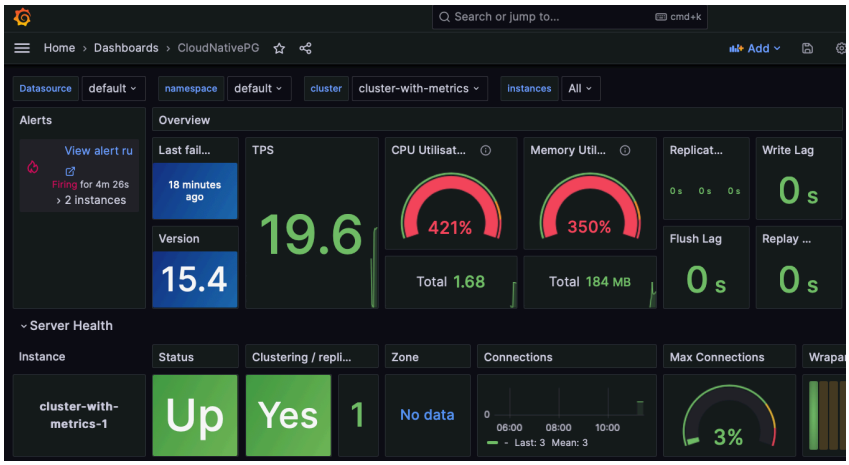


図 2: local grafana

第7章　Bootstrap

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必要なオプションと、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 の外部で、Postgres の異なるメジャーバージョンを使用している場合でも、既存の Postgres 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可能性も提供します。この機能の詳細については、

> *Postgres* **データベースのインポート** 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は、**レプリカクラスター**を作成する可能性を開きます。これは、継続的リカバリーでソースと同期し、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。

警告: CloudNativePG 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 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。

7.1 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 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。Kubernetes の外部からでもデータベースを CloudNativePG に移行する場合に役立ちます。

`initdb` 方法とは異なり、 `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。

CloudNativePG オペレーターが提供するバックアップストレージの組み合わせがいくつかあることを考慮して、各方法のガイダンスについては 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 <リソースタイプ>` を参照してください。

詳細については、

7.2 externalClusters セクション

`externalClusters` セクションは、現在の構成に関連付けられている 1 つ以上の PostgreSQL クラスターを指定するメカニズムを提供します。その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。

- オリジン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されます

- 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。

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 <リソースタイプ>` を参照してください。

詳細については、

7.2.1 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 は自律的に 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 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。

7.3 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.cnpg.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-initdb
spec:
  instances: 3

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

  storage:
    size: 1Gi
```

上記のブートストラップの例では、次のことを行います。

1. PostgreSQL のネイティブ `initdb` コマンドを使用して、新しい `PGDATA` フォルダーを作成します
2. `app` という名前の **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。

7.3.1 initdb にオプションを渡す

実際の PostgreSQL データディレクトリは、 initdb PostgreSQL コマンドの呼び出しを介して作成されます。そのコマンドにカスタムオプションを追加する必要がある場合つまり、テンプレートデータベースに使用される locale を変更するか、データチェックサムを追加する場合、次のパラメーターを使用できます。

dataChecksums dataChecksums が true に設定されている場合、CNPG 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 。

enabled encoding が値に設定されている場合、CNPG 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。

localeCollate localeCollate が値に設定されている場合、CNPG 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、CNPG 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。

walSegmentSize walSegmentSize が値に設定されている場合、CNPG 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 によって定義されています。

注釈: *initdb* ブートストラップ中に CloudNativePG 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.cnpg.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 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。

データベース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することもできます。これらのクエリーは、`postgres` データベースに接続する **スーパーユーザー** `postgres` として実行されます。

```
apiVersion: postgresql.cnpg.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
```

(次のページに続く)

(前のページからの続き)

```
    postInitSQL:
      - CREATE DATABASE angus
  storage:
    size: 1Gi
```

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、*postInitSQL* 、*postInitApplicationSQL* 、および `postInitTemplateSQL` オプションを細心の注意して使用してください。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、クラスターは不完全なままになります。

7.3.2 初期化後のクエリの実行

さらに、シークレットおよび/またはデータベースが作成および構成された後に実行される SQL スクリプトを含む ConfigMaps のリストを指定できます。これらの SQL スクリプトは、**superuser** ロール `postgres` を使用して実行され、 `initdb` セクションで指定されたデータベースに接続されます。

```
apiVersion: postgresql.cnpg.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: *secretRefs* で参照される SQL スクリプトは、 *configMapRefs* で参照される SQL スクリプトよりも前に実行されます。両方のセクションで、SQL スクリプトはリスト内の順序を尊重して実行されます。 SQL スクリプト内では、各 SQL ステートメントは PostgreSQL semantics に従ってサーバー上の単一の exec で実行されます。コメントを含めることができますが、 *psql* のような内部コマンドはできません。

警告: *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てください。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 ファイルのエラーは、ブートストラップフェーズを正常に完了できません。

7.4 別のクラスターからのブートストラップ

CloudNativePG は、同じメジャーバージョンの別のクラスターから開始するクラスターのブートストラップを有効にします。この操作は、ストリーミングレプリケーション `pg_basebackup` を介してソースクラスターに直接接続することにより、または既存の物理的な**ベースバックアップ** `recovery` を介して間接的に接続することにより発生します。

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。 *barmanObjectStore.serverName* プロパティデフォルトで、外部クラスター定義の `name` の値に割り当てられますを使用して、別のプロパティを指定できます。

7.4.1 バックアップからのブートストラップ `recovery`

バックアップとリカバリーの観点から CloudNativePG オペレーターが提供するいくつかの可能性、方法、および組み合わせを考慮して、
バックアップ リカバリー を参照してください。

7.4.2 ライブクラスターからのブートストラップ `pg_basebackup`

`pg_basebackup` ブートストラップモードでは、有効な**ストリーミングレプリケーション**接続を介して、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。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。

この方法の主なユースケースは、Kubernetes の外部または Kubernetes 内から別のオペレーターなどから CloudNativePG への**移行** によって表されます。

警告: 現在の実装では、クローンプロセスが終了し、作成されたクラスターがすぐに起動されるときに、元の PostgreSQL インスタンスの*スナップショット*を作成します。詳細については、以下の 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の場合と同様、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所有権を取得します。これには、CloudNativePG で必要ないくつか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。生成されるクラスターは、ソースインスタンスから完全に独立しています。

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の構成は、実際のコンテキストと環境に依存するため、CloudNativePG ドキュメントの範囲を超えています。

---

pg_basebackup によって透過的に管理されるターゲットインスタンス上のストリーミングレプリケーションクライアントは、次のいずれかの方法でソースインスタンスで自分自身を認証できます。

1. **ユーザー名/パスワードの認証** 経由
2. TLS client certificate 経由

後者は、CloudNativePG によって管理されているソース、または TLS 認証用に構成されているソースに接続する場合に推奨されるものです。ただし、最初のオプションは、一般に PostgreSQL サーバーへの最も一般的な認証形式であり、ソースインスタンスが Kubernetes 外部の従来の環境にある場合、最も簡単な方法になる可能性があります。両方の場合について以下に説明します。

7.4.3 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。

7.4.4 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 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 証明書認証を利用できます。デフォルトでは、streaming_replica に許可される唯一の接続方法です。

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。

7.4.5 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 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 キーに保存されます。

```yaml
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: target-db
spec:
  instances: 3
  imageName: ghcr.io/cloudnative-pg/postgresql:16.1

  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 16.1 を含むすべての要件を満たす必要があります。

7.4.6 TLS 証明書認証

pg_basebackup ブートストラップを使用して CloudNativePG 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 16.1 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。

```
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: cluster-clone-tls
spec:
  instances: 3
  imageName: ghcr.io/cloudnative-pg/postgresql:16.1

  bootstrap:
    pg_basebackup:
      source: cluster-example

  storage:
    size: 1Gi

  externalClusters:
  - name: cluster-example
    connectionParameters:
      host: cluster-example-rw.default.svc
      user: streaming_replica
      sslmode: verify-full
    sslKey:
      name: cluster-example-replication
      key: tls.key
    sslCert:
      name: cluster-example-replication
      key: tls.crt
    sslRootCert:
      name: cluster-example-ca
      key: ca.crt
``` #### アプリケーションデータベースの構成

また、`initdb` および `recovery` ブートストラップ方法の場合と同様に、ライブクラスターからブートストラップするクラスターのアプリケーションデータベースの構成もサポートしています。新しいクラスターがレプリカクラスターとしてレプリカモードが有効化されて作成される場合、アプリケーションデータベースの構成はスキップされます。

次の例では、ライブクラスターからブートストラップした後、指定されたシークレット `app-secret` のパスワードを使用してアプリケーションデータベース `app` を構成します。

```yaml
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
[...]
spec:
```

(次のページに続く)

(前のページからの続き)

```
bootstrap:
  pg_basebackup:
    database: app
    owner: app
    secret:
      name: app-secret
    source: cluster-example
```

上記の構成では、復旧が完了すると次が発生します。

1. データベース app が存在しない場合、新しいデータベース app が作成されます。
2. ユーザー app が存在しない場合、新しいユーザー app が作成されます。
3. ユーザー app がデータベースの所有者でない場合、ユーザー app はデータベース app の所有者として付与されます。
4. username の値がシークレットの owner の値と一致する場合、アプリケーションデータベースのパスワードはシークレットの password の値に変更されます。

重要: レプリカモードが有効になっているレプリカクラスターの場合、オペレーターは PostgreSQL インスタンスにデータベースまたはユーザーを作成しません。これらは元のクラスターからリカバリされるためです。

7.4.7 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、Kubernetes の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。

重要: 移行を試行する前に、プロシージャ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。

第8章　Postgresデータベースのインポート

このセクションでは、新しい CloudNativePG クラスター内に 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 11.x からバージョン 15.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。

8.1 その仕組み

概念的には、インポートには、 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*>` を使用して新しいクラスター**宛先クラスター** 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 **ソースクラスター**からオブジェクトをインポートする必要があります。PostgreSQL の推奨事項に従って、 **宛先クラスター**の PostgreSQL メジャーバージョンが **ソースクラスター**のバージョン以上であることをお勧めします。

CloudNativePG は、ソースクラスターから宛先クラスターにオブジェクトをインポートする 2 つの主な方法を提供します。

- **マイクロサービスアプローチ**宛先クラスターは、CloudNativePG プロジェクトで推奨されているように、指定されたアプリケーションユーザーが所有する単一のアプリケーションデータベースをホストするように設計されています
- **モノリスアプローチ**宛先クラスターは、ソースクラスターからインポートされた複数のデータベースとさまざまなユーザーをホストするように設計されています。

最初のインポート方法は `microservice` タイプを介して使用でき、後者は `monolith` タイプで使用できます。

> **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 "SQL Dump" を参照してください。

詳細については、

8.2 `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 -Fc` 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

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

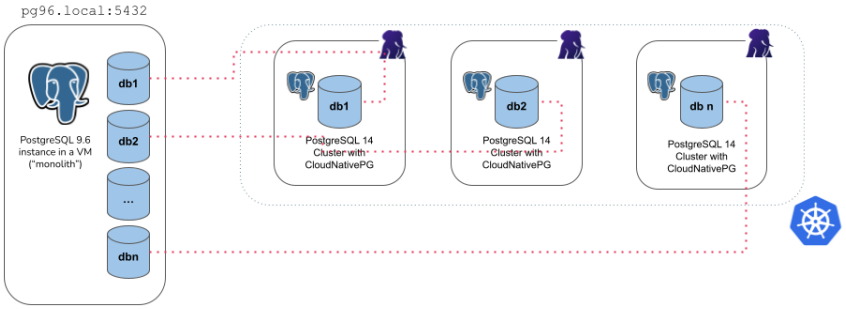


図 1: Example of microservice import type

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。

```yaml
apiVersion: postgresql.cnpg.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、したがって CloudNativePG でサポートされていないバージョンの PostgreSQL を実行しているソースデータベースを意図的に使用しています。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**スーパーユーザー** は OK です
- 現在、`pg_dump -Fc` 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

8.3 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 -Fc` 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`

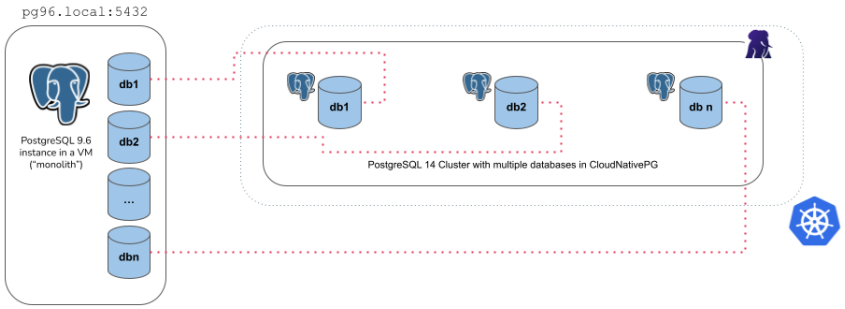


図 2: Example of monolith import type

ユーザーを使用して postgres データベースに接続し、cluster-pg96-superuser シークレットに保存されているパスワードを介して postgres データベースに接続します。

```yaml
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: cluster-monolith
spec:
  instances: 3
  bootstrap:
    initdb:
      import:
        type: monolith
        databases:
          - accounting
          - banking
          - resort
        roles:
          - accountant
          - bank_user
        source:
          externalCluster: cluster-pg96
  storage:
    size: 1Gi
  externalClusters:
    - name: cluster-pg96
      connectionParameters:
        # Use the correct IP or host name for the source database
        host: pg96.local
        user: postgres
        dbname: postgres
        sslmode: require
      password:
        name: cluster-pg96-superuser
        key: password
```

monolith タイプを使用するときは、いくつか注意する必要があります。

- インポートするデータを含む既存の PostgreSQL インスタンスをポイントする externalCluster が必要です。詳細については、 :ref:``externalClusters` セクション <*externalClusters* セクション>` を参照してください。
- オペレーション中に Kubernetes クラスターと externalCluster の間でトラフィックを許可する必要がある
- ソースデータベースへの接続は、pg_dump を実行し、ロール情報を取得する必要がある指定されたユーザーで付与される必要があります**スーパーユーザー** は OK です
- 現在、pg_dump -Fc 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

8.4 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 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 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プロセスが完了した後に 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 の範囲を超えていますが、`Cluster` 構成で `shared_buffers`、`max_wal_size`、`checkpoint_timeout` などの Postgres GUC を直接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。

第9章　セキュリティ

このセクションには、CloudNativePG 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

9.1 コード

CloudNativePG のソースコードは、CI / CD パイプラインで直接 GolangCI-Lint と呼ばれる Go の人気のオープンソースリンターを使用して、**セキュリティ問題** を含む静的分析のために **体系的にスキャンされます。** *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。

これらの 1 つは Golang Security Checker または単に `gosec` で、ハードコードされた資格情報などのコードに隠された既知の脆弱性、脅威、弱点の発見を目的とした一連のルールに対してソースの抽象構文ツリーをスキャンするリンター、整数オーバーフロー、SQL インジェクション-いくつか例を示します。

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 の配信全体のブロッカーです。つまり、各コミットは GolangCI-Lint で定義されたすべてのリンターに対して検証されます。

9.2 コンテナ

CloudNativePG の一部であるすべてのコンテナイメージは、各コミットに続いて CI / CD パイプラインを介して自動的にビルドされます。このようなイメージには、演算子だけでなくオペランド、特にサポートされているすべての PostgreSQL バージョンが含まれます。パイプライン内では、画像は次の方法でスキャンされます。

- Dockle コンテナビルドプロセスに関するベストプラクティス用

重要: 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発生した場合、すべてのオペランドイメージはパイプラインによって 1 日 1 回自動的に再ビルドされ、コミュニティが配布するコンテナイメージの**パッチレベルの更新**を提供します。

コンテナレベルのセキュリティでは、次のガイドラインとフレームワークが考慮されています。

- Container Image Creation and Deployment Guide 、米国国防総省 DoD の防衛情報システム庁 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:

Security and Containers in CloudNativePG を参照してください。

EDB が CloudNativePG のコンテナレベルでのセキュリティに対して採用したアプローチの詳細については、ブログ記事を参照してください。

9.3 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。

9.3.1 ロールベースのアクセス制御 RBAC

オペレーターは、`cnpg-manager` と呼ばれる専用の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。Kubernetes では、これはデフォルトで `cnpg-system` 名前空間にインストールされ、このサービスアカウントと、オペレーターに付与されるルール/リソース/動詞のセットを定義する `cnpg-manager` クラスターロール間のクラスターロールバインディングを使用します。

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。

以下に、いくつかの例と、最も重要な理由を提供します CloudNativePG が標準の Kubernetes 名前空間リソースの完全または部分的な管理を必要とする理由。

`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する必要があるため、どのノードでポッドをスケジュールできるかを決定して、レプリカが同じノードにならないようにします、特にノードが別のアベイラビリティーゾーンにある場合。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するためにも使用され、まったく作成できないポッドの作成を回避します。

オペレーターに必要なすべての権限を表示するには、`kubectl describe clusterrole cnpg-manager` を実行できます。

9.3.2 インスタンスマネージャーによる API サーバーへの呼び出し

オペランドコンテナのエントリポイントであるインスタンスマネージャーは、Kubernetes API サーバーへのいくつかの呼び出しを行って、一部のリソースのステータスが正しく更新されることを確認し、その Postgres クラスターに関連付けられている構成マップとシークレットにアクセスする必要があります。。このような呼び出しは、同じ PostgreSQL Cluster リソース名を共有するオペレーターによって作成された専用の ServiceAccount を介して実行されます。

重要: このオペランドは、API サーバーを介してリソースの特定の限定されたサブセットにのみアクセスできます。サービスアカウントは recommended way to access the API server from within a Pod です。

透明性を維持するために、サービスアカウントに関連付けられている権限は roles.go

ファイル。たとえば、 myns 名前空間の汎用 mypg クラスターの権限を取得するには、次のコマンドを入力します。

```
kubectl get role -n myns mypg -o yaml
```

次に、ロールがサービスアカウントにバインドされていることを確認します。

```
kubectl get rolebinding -n myns mypg -o yaml
```

重要: **ロールは特定の名前空間**に制限されていることに注意してください。

以下に、汎用の Kubernetes リソースのサービスアカウントに関連付けられた権限の簡単な概要を提供します。

configmaps インスタンスマネージャーは、カスタムモニタリングクエリなど、同じクラスターに関連する構成マップのみを読み取ることができます

secrets インスタンスマネージャーは、同じクラスターに関連するシークレットのみを読み取ることができます。ストリーミングレプリケーションユーザー、アプリケーションユーザー、スーパーユーザー、LDAP 認証ユーザー、クライアント CA、サーバー CA、サーバー証明書、バックアップ資格情報、カスタムモニタリングクエリー

events インスタンスマネージャーは、クラスターのイベントを作成し、PostgreSQL インスタンスのライフサイクルの特定の側面について API サーバーに通知できます

ここでは、代わりに、CloudNativePG 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

9.3.3 ポッドセキュリティポリシー

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 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。

9.3.4 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。

参考:

edb_notranlate_2

> **警告:** この種類のアノテーションを使用すると、クラスターが動作を停止する可能性があります。この場合、アノテーションは `Cluster` から安全に削除できます。

AppArmor 構成は Kubernetes ノードレベルである必要があります。つまり、基になるオペレーティングシステムでこのオプションを有効にし、適切に構成する必要があります。

これは状況ではなく、アノテーションが `Cluster` 作成時に追加された場合、ポッドは作成されません。一方、`Cluster` が作成された後にアノテーションを追加すると、クラスター内のポッドは起動できず、次のようなエラーが発生します。

```
metadata.annotations[container.apparmor.security.beta.kubernetes.io/postgres]:␣
↪Forbidden: may not add AppArmor annotations]
```

このような場合、Kubernetes 管理者に連絡し、使用する適切な AppArmor プロファイルを聞いてください。

9.3.5 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 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。

9.3.6 公開ポート

CloudNativePG 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。

システム |ポート番号 |露出 |名前 |証明書 |認証 :————— |:——— – |:————— – |:————— – |:——— – |:———— – 演算子 | 9443 | Webhook サーバー| `webhook-server` | TLS | Yes 演算子 | 8080 |メトリック | `metrics` | TLS なし |インスタンスマネージャーなし | 9187 |メトリック | `metrics` | TLS なし |インスタンスマネージャーなし | 8000 |ステータス | `status` | TLS なし |オペランドなし | 5432 | PostgreSQL インスタンス | `postgresql` |オプショナル TLS |はい

9.3.7 PostgreSQL

CloudNativePG 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:** CloudNativePG 1.21 より前では、*enableSuperuserAccess* はデフォルトで `true` に設定されていました。この変更は、オペレーターのデフォルトセキュリティの姿勢を向上させるために実装され、PostgreSQL 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、開発者にデータベース内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。

パスワードの暗号化に関する限り、CloudNativePG 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、 **シークレット** を参照してください。

これらのファイルを使用して、データベースへのアプリケーションのアクセスを構成できます。

デフォルトでは、すべてのレプリカは、`streaming_replica` と呼ばれる特別なユーザーを使用して、現在のプライマリインスタンスに**物理非同期ストリーミングレプリケーション**で接続するように自動的に構成されます。ノード間の接続は **暗号化**され、認証は **TLS クライアント証明書**を介して行われます。詳細については、

クライアント *TLS/SSL* **接続** ページを参照してください。デフォルトでは、オペレーターは TLS

v1.3 接続を必要とします。

現在、オペレーターは、管理者が `postgresql` 構成の `pg_hba` セクションの一部としてマニフェストに `pg_hba.conf` 行を直接追加できます。マニフェストで定義された行は、デフォルトの `pg_hba.conf` に追加されます。

オペレーターによる `pg_hba.conf` の管理方法の詳細については、ドキュメントの *PostgreSQL* **の構成** 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* **の構成** 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。

9.3.8 ストレージ

CloudNativePG 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。

第10章　Postgresインスタンスマネージャー

CloudNativePG は、フェールオーバー管理に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 主要プロセス `postmaster` としても知られるライフサイクル全体を管理します。

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。

10.1 スタートアップ、活性、および準備状況プローブ

スタートアップとライブネスプローブは `pg_isready` に依存しますが、準備状況プローブは、データベースが稼働していて、スーパーユーザーの資格情報を使用して接続を受け入れることができるかどうかを確認します。

ポッドがトラフィックを受け入れる準備ができている場合、準備状況プローブはポジティブになります。liveness プローブは、起動プローブ間隔が経過した後、コンテナを再起動するタイミングを制御します。

重要: liveness プローブと readness プローブは、プローブコマンドが各チェックの間に 10 秒の間隔で 3 回失敗した場合に失敗を報告します。

---

liveness プローブは、PostgreSQL インスタンスが破損した状態にあり、再起動が必要かどうかを検出します。`startDelay` の値は、プローブの実行を遅延するために使用され、起動時間の長いインスタンスが再起動されないようにします。

ポッドが起動してから liveness プローブが動作を開始するまでの間隔秒単位は、`.spec.startDelay` パラメーターで表されます。デフォルトは 3600 秒です。クラスターの正しい値は、PostgreSQL の起動に必要な時間に関連しています。

> **警告:** *.spec.startDelay* が低すぎる場合、PostgreSQL の起動が完了する前に liveness プローブは動作を開始し、Pod が途中で再起動される可能性があります。

10.2 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 RPO 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ために、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。

10.2.1 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、PostgreSQL データベースの RPO および RTO に影響します。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。

10.3 フェイルオーバー

プライマリポッドに障害が発生した場合、クラスターはフェールオーバーモードになります。詳細は **自動フェイルオーバー** を参照してください。

第11章　スケジューリング

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。

> AffinityConfiguration {postgresql-cnpg-io-v1-AffinityConfiguration} を介して CloudNativePG 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

注釈: CloudNativePG は、ワークロードのスケジュールをより詳細に制御するためのポッドテンプレートをサポートしていません。これらは初期コンセプトの一部でしたが、開発チームは、API の新しいバージョンおそらく CNPG の v2 での導入を延期することにしました。

11.1 ポッドのアフィニティとアンチアフィニティ

Kubernetes では、これらのノードで既に実行されている実際のワークロードに基づいて、ポッドを**アフィニティ**スケジュールするか、**アンチアフィニティ**しないノードを制御できます。これは、技術的には** 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。

デフォルトでの CloudNativePG は、クラスターのインスタンスをできれば別のノードで構成し、次の `affinity` 定義になります。

```
affinity:
  podAntiAffinity:
    preferredDuringSchedulingIgnoredDuringExecution:
      - podAffinityTerm:
          labelSelector:
```

(次のページに続く)

(前のページからの続き)

```
          matchExpressions:
            - key: postgresql
              operator: In
              values:
                - cluster-example
        topologyKey: kubernetes.io/hostname
      weight: 100
```

次のクラスター仕様の結果。

```
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3
  imageName: ghcr.io/cloudnative-pg/postgresql:16.1

  affinity:
    enablePodAntiAffinity: true #default value
    topologyKey: kubernetes.io/hostname #defaul value
    podAntiAffinityType: preferred #default value

  storage:
    size: 1Gi
```

したがって、Kubernetes は、リソースが許可する 3 つの異なるノードを介して 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。

前述のデフォルトの動作は、上記の設定を調整することにより変更できます。

`podAntiAffinityType` は `required` に設定できます。その結果、`preferredDuringSchedulingIgnoredDuringExecution` の代わりに `requiredDuringSchedulingIgnoredDuringExecution` が使用されます。このような強い要件は、リソースが利用できない場合に保留中のインスタンスが発生する可能性があることに注意してください。これは、Kubernetes クラスターの自動ホリゾンタルスケーリングに Cluster Autoscaler を使用する場合に予想される条件です。 。

参考:

このトピックの詳細については、 *Kubernetes documentation* を参照してください。

クラウド環境での `topologyKey` の別の値は `topology.kubernetes.io/zone` にすることができ、ポッドがノードだけでなくアベイラビリティーゾーンに分散されるようにします。

> Well-Known Labels, Annotations and Taints を参照してください。

さらに多くのオプションについては。

`enablePodAntiAffinity` を false に設定することにより、オペレーターが生成したアンチアフィニティポリシーを無効にできます。

さらに、より詳細な制御が必要な場合、 additionalPodAffinity および additionalPodAntiAffinity 構成属性を介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のリストを指定できます。これらのルールは、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、それ以外の場合は透過的に渡されます。

注釈: ポッド仕様で想定されている `podAntiAffinity` または `podAffinity` のコンテンツ全体を `additionalPodAntiAffinity` または `additionalPodAffinity` に渡す必要があります。どの PostgreSQL に関係なく、すべてのワーカーノードで PostgreSQL の 1 つのインスタンスのみを実行する例として、次の YAML を参照してください。それらが属しているクラスター)。

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

11.2 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 は、 affinity セクションで nodeSelector を定義することに同意して、これらのラベルを持つノードでのみ実行する PostgreSQL クラスターを要求できます。

11.3 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。

第12章　リソース管理

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 を使用すると、管理者は、マニフェストの `resources` セクションを介して、クラスターのポッドによるリソース使用を制御および管理できます。

- `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。

```yaml
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。

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` OOM はメモリ不足の略です `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.cnpg.io/v1
kind: Cluster
metadata:
  name: postgresql-resources
spec:

  instances: 3

  postgresql:
    parameters:
      shared_buffers: "256MB"

  resources:
    requests:
      memory: "1024Mi"
      cpu: 1
    limits:
      memory: "1024Mi"
      cpu: 1

  storage:
    size: 1Gi
```

上記の例では、 `256MB` の値で `shared_buffers` パラメーターを指定しました。つまり、データをキャッシュするために PostgreSQL サーバーに専用のメモリの量です。このパラメーターのデフォルト値は定義されていない場合の `128MB` です。

`shared_buffers` の適切な開始値は、システム内のメモリの 25%です。例えば `shared_buffers` が 256 MB の場合、コンテナメモリサイズの推奨値は 1 GB です。これは、ポッド内ですべてのコンテナが合計 1 GB のメモリを持つことを意味し、Kubernetes が常に保持し、コンテナができることを意味します予想通りに動作します。詳細については、

Resource Consumption を参照してください。

PostgreSQL ドキュメントのセクション。

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。

第13章　障害モード

このセクションでは、PostgreSQL がその有効期間中に Kubernetes クラスターで直面する可能性のある主要な障害シナリオの概要を提供します。

重要: 発生している障害シナリオがこのセクションでカバーされていない場合、すぐに professional support を探してください。

参考:

Postgres **インスタンスマネージャー** を参照してください。

CloudNativePG によって実装される liveness プローブと readness プローブの詳細については、を参照してください。

13.1 ストレージ領域の使用量

オペレーターは、PostgreSQL インスタンスごとに 1 つの PVC をインスタンス化して、`PGDATA` コンテンツを保存します。クラスターの初期化中に`.spec.walStorage` が指定された場合、WAL ストレージ専用の 2 番目の PVC がプロビジョニングされます。

このような記憶領域は、2 つの場合に再利用のために設定されます。

- 対応するポッドがユーザーによって削除され、新しいポッドが再作成されるとき
- 対応するポッドが削除され、別のノードでスケジュールされたとき

オペレーターが特定の PVC を再利用したくない場合は、ポッドを削除する前に PVC を削除する必要があります。このために、次のコマンドを使用できます。

```
kubectl delete -n [namespace] pvc/[cluster-name]-[serial] pod/[cluster-name]-[serial]
```

注釈: 専用の WAL ボリュームを指定した場合、このプロセス中に削除する必要があります。

```
kubectl delete -n [namespace] pvc/[cluster-name]-[serial] pvc/[cluster-name]-[serial]-
↪wal pod/[cluster-name]-[serial]
```

例

```
$ kubectl delete -n default pvc/cluster-example-1 pvc/cluster-example-1-wal pod/
↪cluster-example-1
persistentvolumeclaim "cluster-example-1" deleted
persistentvolumeclaim "cluster-example-1-wal" deleted
pod "cluster-example-1" deleted
```

13.2 障害モード

Cluster に属するポッドは、次の方法で障害が発生する場合があります。

- ポッドはユーザーによって明示的に削除されます。
- postgres コンテナの準備状況プローブは失敗します。
- postgres コンテナの liveness プローブは失敗します。
- Kubernetes ワーカーノードはドレインされます。
- ポッドがスケジュールされている Kubernetes ワーカーノードに障害が発生します。

これらの障害はそれぞれ、Cluster およびオペレーターが管理するサービスにさまざまな影響を与えます。

13.2.1 ユーザーによって削除されたポッド

オペレーターに削除が通知されます。 Cluster に属する新しいポッドは、既存の PVC がある場合は再利用して、それ以外の場合は**プライマリ** の物理バックアップから開始して自動的に作成されます。

重要: ポッドを意図的に削除した場合、*PodDisruptionBudget* ポリシーは適用されません。

自己修復は、 *apiserver* に通知されるとすぐに発生します。

次の汎用コマンドを使用して、クラスターの特定のポッドで突然の障害をトリガーできます。

```
kubectl delete -n [namespace] \
  pod/[cluster-name]-[serial] --grace-period=1
```

たとえば、プライマリで実際の障害をシミュレートし、フェールオーバープロセスをトリガーする場合は、次を実行できます。

```
kubectl delete pod [primary pod] --grace-period=1
```

警告: PostgreSQL クラスターで誤解を招く結果を生成する可能性があるため、フェールオーバーシミュレーションテストでは`--grace-period=0` を決して使用しないでください。猶予期間 0 は、実際の障害が発生した場合に何が起こるかとは逆に、*postgres* コンテナインスタンスマネージャーの PID 1 プ

ロセスがシャットダウンしていることを最初に確認せずに、ポッドが Kubernetes API サーバーからすぐに削除されることを保証します例 電源コードケーブルまたはネットワークパーティショニングを取り外します。その結果、オペレーターはプライマリのポッドを認識せず、プライマリがシャットダウンされた保証なしで、最も調整されたスタンバイをプロモーションするフェールオーバーをトリガーします。

13.2.2 Readiness プローブの失敗

3 回失敗すると、ポッドは **準備ができていない**とみなされます。ポッドは `Cluster` の一部のままで、新しいポッドは作成されません。

障害の原因を修正できない場合は、ポッドを手動で削除できます。それ以外の場合、ポッドは、障害が解決されたときに以前のロールを再開します。

自己修復は、プローブが 3 回失敗すると発生します。

13.2.3 Liveness プローブの障害

3 回障害が発生すると、`postgres` コンテナは障害が発生したとみなされます。ポッドは `Cluster` の一部であり、 *kubelet* はコンテナを再起動しようとします。障害の原因を修正できない場合は、ポッドを手動で削除できます。

自己修復は、プローブが 3 回失敗すると発生します。

13.2.4 ワーカーノードがドレインされました

ポッドはワーカーノードから削除され、サービスから削除されます。`nodeMaintenanceWindow` パラメーターの `reusePVC` オプションが `off` に設定されている場合、新しいポッドは **プライマリ**の物理バックアップから別のワーカーノードに作成されます。デフォルトメンテナンス時間帯は `on` 、それ以外の場合は `off` 。

`PodDisruptionBudget` は、準備ができていない少なくとも別のポッドが存在する場合、ポッドのエビクトを防止する場合があります。

注釈: *reusePVC* が `false` に設定されている場合、単一インスタンスクラスターはノードのドレインを防止します。 *Kubernetes Upgrade section* を参照してください。

自己修復は、 *apiserver* に通知されるとすぐに発生します。

13.2.5 ワーカーノードの障害

ノードに障害が発生しているため、 *kubelet* は liveness プローブと readness プローブを実行しません。ポッドは、その特定の障害原因に対して Kubernetes クラスター管理者が構成した許容秒後、削除用にマークされます。Kubernetes クラスターの構成に基づいて、ポッドはサービスから以前に削除される場合があります。

新しいポッドは、 **プライマリ**の物理バックアップから別のワーカーノードに作成されます。Kubernetes クラスターにおけるそのパラメーターのデフォルト値は 5 分です。

自己修復は `tolerationSeconds` の後に発生します。

13.3 自己修復

障害が発生したポッドがスタンバイの場合、ポッドは`-r` サービスおよび`-ro` サービスから削除されます。ポッドは、可能な場合は PVC 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。準備ができたら、ポッドは`-r` サービスと`-ro` サービスに再度追加されます。

障害が発生したポッドがプライマリの場合、オペレーターはステータスが準備ができており、レプリケーションラグが最も低いアクティブなポッドをプロモートし、`-rw` サービスをそこにポイントします。障害が発生したポッドは、`-r` サービスおよび`-rw` サービスから削除されます。他のスタンバイは、新しいプライマリからの複製を開始します。元のプライマリは、PVC が利用可能な場合、`pg_rewind` を使用して新しいプライマリと自分自身を同期します。それ以外の場合、現在の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。

13.4 手動介入

文書化されていない障害の場合、介入して問題を手動で解決する必要がある場合があります。

重要: このような場合、 professional support なしで手動操作を実行しないでください。

オペレーターのバージョン 1.11.0 から、次のように `cnpg.io/reconciliationLoop` アノテーションを使用して、選択した PostgreSQL クラスターでリコンシリエーションループを一時的に無効にすることができます。

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

`cnpg.io/reconciliationLoop` は、特別な/緊急操作の期間のみに、細心の注意を払って使用する必要があります。

警告: このアノテーションは限定的な期間のみ使用し、緊急事態が終了したら削除するようにしてください。このアノテーションをクラスターに残すと、オペレーターはフェールオーバーなどの自己修復操作を発行できません。

第14章　ローリングアップデート

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。
- 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* 場合を除きます。

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 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。

14.1 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、RPO 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、更新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。

14.2 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 cnpg promote [cluster] [new_primary]
```

次の方法で再起動をトリガーできます。

```
kubectl cnpg restart [cluster] [current_primary]
```

詳細については、 Backup {postgresql-cnpg-io-v1-Backup} をご覧ください。

第15章　レプリケーション

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 を参照してください。

セクション。

15.1 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、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で高可用性を構築し、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。

15.2 非常に成熟したテクノロジー

PostgreSQLには、プライマリインスタンスから1つ以上のレプリカにデータをレプリケートするための非常に堅牢で成熟したネイティブフレームワークがあり、WALライトアヘッドログに継続的に保存されるトランザクションの変更の概念を中心に構築されています。

クラッシュリカバリーとポイントインタイムリカバリーテクノロジーの進化として始まった物理的レプリケーションは、継続的リカバリーでのプライマリからウォームスタンバイへのWALシッピングを介してPostgreSQL 8.22006で最初に導入されました。

PostgreSQL 9.0 2010は、 **ホットスタンバイ**を介した読み取り専用レプリカでそれを強化しましたが、 9.1 2011は、トランザクションレベルでの同期レプリケーションRPO=0クラスターの場合。カスケードレプリケーションは、PostgreSQL 9.22012でリリースされました。論理レプリケーションの基礎はPostgreSQL 9.4で築かれましたが、バージョン102017では、オリジンからデスティネーションにデータをレプリケートするパブリッシャー/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。

15.3 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 は、 spec で提供された instances の数に基づいて、宣言的な方法でクラスター内の物理的ストリーミングレプリカを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
hostssl replication streaming_replica all cert
```

参考:

CloudNativePG が証明書を管理する方法の詳細については、 **証明書** 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 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。

15.3.1 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 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。

15.4 同期レプリケーション

CloudNativePG 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を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 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。

15.4.1 同期レプリケーションのノードを選択します

CloudNativePG 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、 スケジューリング を確認してください。

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堅牢性、特に目標復旧ポイント RPO の観点から向上します。

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データ損失ゼロのクラスター 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。

15.5 高可用性のための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 は、高可用性クラスターから始めて、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 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。

CloudNativePG 1.18 で導入されたこの機能は、デフォルトで有効になり、構成から無効にできます。詳細については、

> ReplicationSlotsConfiguration {postgresql-cnpg-io-v1-ReplicationSlotsConfiguration} を参照してください。以下に、メインオプションの簡単な説明を示します。

`.spec.replicationSlots.highAvailability.enabled` true の場合、機能は有効になります `true` は 1.21 以降のデフォルトです

`.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`_cnpg_`

`.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

重要: この機能は、 pg_replication_slot_advance() に依存しているため、PostgreSQL 11 以降が必要です。

レプリケーションスロットの位置を直接操作します。

警告: PostgreSQL 11 では、レプリケーションスロットを最初に無効にした場合に有効にしたり、最初に有効にした場合に無効にしたりすると、クラスターのローリング更新が必要になります起動時にのみ読み込まれる `recovery.conf` ファイルの存在のため。

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.cnpg.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3
  # Disable replication slots for HA in the cluster
  replicationSlots:
    highAvailability:
      enabled: false

  storage:
    size: 1Gi
```

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.cnpg.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:
    highAvailability:
      enabled: true
    updateInterval: 300

  storage:
    size: 1Gi
```

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 展開をモニタする方法の詳細については、 MonitoringConfiguration {postgresql-cnpg-io-v1-

MonitoringConfiguration} を参照してください。

15.5.1 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"
# ...
```

第16章　バックアップ リカバリー

CloudNativePG 1.20 までは、このページに PostgreSQL クラスターのバックアップとリカバリーフェーズの両方が含まれていました。その理由は、CloudNativePG がバックアップおよびリカバリーオブジェクトストアのみをサポートしているためです。

バージョン 1.21 では、Kubernetes `VolumeSnapshot` API のサポートを導入し、エンドユーザーにより多くの可能性を提供します。

その結果、

付録 *A*-バックアップ用の共通オブジェクトストア および バックアップ リカバリー は、2 つの別個のセクションになりました。

第17章　バックアップ

重要: バージョン 1.21 では、 *Kubernetes Volume Snapshots* のネイティブサポートの導入により、CloudNativePG のバックアップおよびリカバリー機能が大幅に変更されました。その時点まで、バックアップとリカバリーはオブジェクトストアでのみ利用できました。このセクションと **バックアップ リカバリー** を注意深くお読みください

CloudNativePG 1.15〜1.20 のユーザーである場合に 1 つ。

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。

注釈: PostgreSQL でデータベースをバックアップする別の方法があります。*pg_dump* ユーティリティは、物理的バックアップではなく論理バックアップに依存します。ただし、論理バックアップは事業継続のユースケースには適していないため、少なくともまだ CloudNativePG の対象になっていません。 *pg_dump* ユーティリティを使用する場合は、 *Troubleshooting / Emergency backup* からインスピレーションを得てください。

CloudNativePG 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

WAL アーカイブは、現時点ではオブジェクトストアにのみ保存できます。

一方、CloudNativePG は、物理ベースバックアップを保存する 2 つの方法をサポートしています。

- **付録 *A*-バックアップ用の共通オブジェクトストア** で、tarballs として - オプションで圧縮
- *Kubernetes Volume Snapshots* 上、基になるストレージクラスでサポートされている場合

重要: CloudNativePG でバックアップ戦略を選択する前に、時間をかけて WAL アーカイブ、ホットおよびコールドバックアップなどの基本的な概念を理解することが重要です。

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 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。

17.1 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できます。

> **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 を使用すると、CloudNativePG は、リージョンを超えても、ディザスターリカバリーに 5 分以下の RPO 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。

17.2 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。

17.3 オブジェクトストアまたはボリュームスナップショットどちらを使用しますか?

CloudNativePG では、オブジェクトストアベースのバックアップ

- 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
- インクリメンタルコピーをサポートしていません
- 差分コピーをサポートしていません

代わりにボリュームスナップショット

- WAL アーカイブは必要ありませんが、運用環境では常にお勧めします
- 基になるストレージクラスに応じて、インクリメンタルコピーをサポート
- 基になるストレージクラスに応じて、差分コピーをサポート
- コールドバックアップもサポート

どちらを使用するかは、次を含む特定の要件と環境によって異なります。

- Kubernetes クラスターでの実行可能なオブジェクトストアソリューションの可用性
- ボリュームスナップショットをサポートする信頼できるストレージクラスの可用性
- データベースのサイズオブジェクトストアの場合、データベースが大きくなると、バックアップ、そして最も重要なことに、リカバリ手順に時間がかかります後者は RTO に影響します。超大規模データベース VLDB が存在する場合、一般的なアドバイスは、コピーオンライトのおかげで、より速いリカバリを提供するため、ボリュームスナップショットに依存することです
- データのモビリティと、別のリージョンまたはその後のリージョンのセカンダリの場所にバックアップファイルを保存または中継する可能性

- その他の要素、主に基になるストレージソリューションへの信頼と精通に基づいています

以下の概要表は、物理ベースバックアップの保存に使用可能な 2 つの方法の主な違いの一部を強調表示しています。

	Object store	Volume Snapshots
WAL archiving	Required	Recommended (1)
Cold backup	☒	✓
Hot backup	✓	✓
Incremental copy	☒	✓ (2)
Differential copy	☒	✓ (2)
Backup from a standby	✓	✓
Snapshot recovery	☒ (3)	✓
Point In Time Recovery (PITR)	✓	Requires WAL archive
Underlying technology	Barman Cloud	Kubernetes API
上記の表の注意事項については、以下の説明を参照してください。> >		
1. 現在、WAL アーカイブはオブジェクトストア上にある必要があります >		
2.PostgreSQL ボリュームの基になるストレージクラスでサポートされている場合		
> 3. スナップショットリカバリーはエミュレート可能>		
`bootstrap.recovery.recoveryTarget.targetImmediate` オプションを使用する		

17.4 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 でバックアップ戦略を構成するための推奨される方法です。これらは ScheduledBackup リソースによって管理されます。

注釈: ScheduledBackupSpec {postgresql-cnpg-io-v1-ScheduledBackupSpec} 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。

schedule フィールドでは、 *Go `cron* package format <https://pkg.go.dev/github.com/robfig/cron#hdr-CRON_Expression_Format>`_ で表現される秒を含む *6* **期間** *cron* **スケジュール** 仕様を定義できます。

> **警告:** このフォーマットは `seconds` フィールドも受け入れます。Unix/Linux システムの `crontab` フォーマットとは異なることに注意してください。

これは、スケジュールされたバックアップの例です。

```yaml
apiVersion: postgresql.cnpg.io/v1
kind: ScheduledBackup
metadata:
  name: backup-example
spec:
  schedule: "0 0 0  **  *"
  backupOwnerReference: self
  cluster:
    name: pg-backup
```

上記の例では、スケジュールで秒、分、および時間を指定する一方、日、月、曜日にすべてを意味するワイルドカードを指定するため、毎日午前 0 時にバックアップをスケジュールします。

Kubernetes CronJob では、秒が含まれないため、同等の式は `0 0 ** *` になります。

ヒント: バックアップ頻度は、完全またはポイントインタイム復旧操作が必要なディザスターの発生後の復旧時間オブジェクト RTO に影響を与える可能性があります。私たちのアドバイスは、バックアップをリカバリーし、スクラッチからのリカバリーにかかる時間を測定することにより、バックアップを定期的にテストして、RTO の予測可能性を調整することです。リカバリ時間は、ベースバックアップのサイズと、リカバリ中にアーカイブから取得して再生する必要がある WAL ファイルの量に影響されます。WAL アーカイブは、PostgreSQL での継続的なバックアップを可能にするものであることに注意してください。私たちの経験に基づいて、ほとんどの場合、毎週のベースバックアップで十分です。ただし、1 日 1 回より頻繁にバックアップをスケジュールすることは非常にまれです。

.spec.method 属性、デフォルトでは barmanObjectStore に設定を介して、定義されたオブジェクトストアまたはボリュームスナップショットのバックアップをスケジュールするかどうかを選択できます。

オブジェクトストアまたはボリュームスナップショットどちらを使用しますか? を適切に定義している場合

クラスターの backup スタンザでは、ボリュームスナップショットのベースバックアップのスケジュールを開始するように method: volumeSnapshot を設定できます。

スケジュールされたバックアップは、必要に応じて .spec.suspend: true を設定することにより一時停止できます。これにより、オプションが削除されるか false に設定されるまで、新しいバックアップはスケジュールされなくなります。

ScheduledBackup リソースが作成されたらすぐにバックアップを発行したい場合は、.spec.immediate: true を設定できます。

注釈: *.spec.backupOwnerReference* は、作成され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

17.5 オンデマンド バックアップ

注釈: BackupSpec {postgresql-cnpg-io-v1-BackupSpec} 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。

新しいバックアップを要求するには、次のような新しい Backup リソースを作成する必要があります。

```
apiVersion: postgresql.cnpg.io/v1
kind: Backup
metadata:
  name: backup-example
spec:
  method: barmanObjectStore
  cluster:
    name: pg-backup
```

この場合、オペレーターはクラスターのオーケストレーションを開始し、barman-cloud-backup を使用してオブジェクトストアで必要なバックアップを取得します。通常の kubectl describe backup <name> コマンドを使用して、バックアップステータスを確認できます。

```
Name:         backup-example
Namespace:    default
Labels:       <none>
Annotations:  API Version:  postgresql.cnpg.io/v1
Kind:         Backup
```

(次のページに続く)

(前のページからの続き)

```
Metadata:
  Creation Timestamp:  2020-10-26T13:57:40Z
  Self Link:         /apis/postgresql.cnpg.io/v1/namespaces/default/backups/backup-
↪example
  UID:               ad5f855c-2ffd-454a-a157-900d5f1f6584
Spec:
  Cluster:
    Name:  pg-backup
Status:
  Phase:       running
  Started At:  2020-10-26T13:57:40Z
Events:        <none>
```

バックアップが完了すると、フェーズは次の例のように completed になります。

```
Name:         backup-example
Namespace:    default
Labels:       <none>
Annotations:  API Version:  postgresql.cnpg.io/v1
Kind:         Backup
Metadata:
  Creation Timestamp:  2020-10-26T13:57:40Z
  Self Link:         /apis/postgresql.cnpg.io/v1/namespaces/default/backups/backup-
↪example
  UID:               ad5f855c-2ffd-454a-a157-900d5f1f6584
Spec:
  Cluster:
    Name:  pg-backup
Status:
  Backup Id:         20201026T135740
  Destination Path:  s3://backups/
  Endpoint URL:      http://minio:9000
  Phase:             completed
  s3Credentials:
    Access Key Id:
      Key:   ACCESS_KEY_ID
      Name:  minio
    Secret Access Key:
      Key:      ACCESS_SECRET_KEY
      Name:     minio
  Server Name:  pg-backup
  Started At:   2020-10-26T13:57:40Z
  Stopped At:   2020-10-26T13:57:44Z
Events:         <none>
```

!!!重要この機能は、スーパーユーザーとアプリケーションユーザーのシークレットをバックアップしませ

ん。シークレットは、Kubernetes クラスターの標準のバックアップ手順の一部としてバックアップされることになっています。

17.6 スタンバイからのバックアップ

ベースバックアップを取るには、ディスク上の PostgreSQL インスタンスのデータコンテンツ全体をスクレイピングする必要があり、データベースの実際のワークロードとの I/O 競合が発生する可能性があります。

このため、CloudNativePG では、 PostgreSQL で直接利用可能な機能**スタンバイからのバックアップ** を利用できます。

デフォルトでは、バックアップは `Cluster` の最もアライメントの高い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で実行されます。

注釈: 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。

常にプライマリでバックアップを実行したい場合は、以下の例に示すようにバックアップターゲットを `primary` に設定できます。

```yaml
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  [...]
spec:
  backup:
    target: "primary"
```

警告: ボリュームスナップショットを使用してコールドバックアップを実行するときは、ターゲットをプライマリに設定しないように注意してください。これにより、スナップショットの取得に必要な時間プライマリがシャットダウンし、書き込み操作に影響します。これは、プライマリをターゲットとして明示的に設定していない場合でも、単一インスタンスのクラスターでコールドバックアップを取得する場合にも適用されます。

バックアップターゲットが `prefer-standby` に設定されている場合、このようなポリシーは、バックアップが最新の使用可能なセカンダリインスタンスで実行されるか、他のインスタンスが使用できない場合はプライマリインスタンスで実行されます。

デフォルトでは、特に指定しない限り、ターゲットはスタンバイからバックアップを取得するように自動的に設定されます。

Cluster で指定されたバックアップターゲットは、次の例のように、Backup および ScheduledBackup タイプでオーバーライドできます。

```
apiVersion: postgresql.cnpg.io/v1
kind: Backup
metadata:
  [...]
spec:
  cluster:
    name: [...]
  target: "primary"
```

前の例では、Cluster がレプリカを優先するように設定されている場合でも、CloudNativePG は常にプライマリインスタンスを選択します。

第18章　オブジェクトストアのバックアップ

CloudNativePG 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 `ghcr.io/cloudnative-pg/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、システムでオペランドの最新バージョン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 。

18.1 共通オブジェクトストア

> *AWS S3* 、 *Microsoft Azure Blob Storage* 、 *Google Cloud Storage* 、または *MinIO* ゲートウェイ

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、

> 付録 *A*-バックアップ用の共通オブジェクトストア を参照してください。

18.2 リテンション ポリシー

重要: 現在、保持ポリシーはボリュームスナップショットでは使用できません。

CloudNativePG 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.cnpg.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"
```

18.3 圧縮アルゴリズム

CloudNativePG 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
- きびきび

バックアップと WAL の圧縮設定は独立しています。API リファレンスの DataBackupConfiguration {postgresql-cnpg-io-v1-DataBackupConfiguration} および WalBackupConfiguration {postgresql-cnpg-io-v1-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

18.4 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 では、バックアップオブジェクト、つまりベースバックアップ、WAL ファイル、履歴ファイルのキーと値のペアとしてタグを指定できます。

`.spec.backup.barmanObjectStore` 定義では 2 つのプロパティを使用できます。

- `tags` バックアップオブジェクトとバックアップオブジェクトストアのアーカイブされた WAL ファイルに追加するキーと値のペアタグ
- `historyTags` バックアップオブジェクトストアのアーカイブされた履歴ファイルに追加されるキーと値のペアタグ

以下の YAML マニフェストの抜粋は、この機能の使用例を提供します。

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

第19章　WALアーカイブ

WAL アーカイブは、 *WAL アーカイブ* をフィードするプロセスです

CloudNativePG で。

重要: CloudNativePG は現在、オブジェクトストアの WAL アーカイブのみをサポートしています。このような WAL アーカイブは、オブジェクトストアのバックアップとボリュームスナップショットのバックアップの両方に役立ちます。

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。

> **オブジェクトストアのバックアップ** にあるものと同じ手順を進めて、WAL アーカイブを設定してください。

注釈: BarmanObjectStoreConfiguration {postgresql-cnpg-io-v1-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、API リファレンスを参照してください。

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.cnpg.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 は `archive_timeout` を `5min` に設定し、ワークロードが低い場合でも WAL ファイルが少なくとも 5 分ごとに閉じてアーカイブされ、目標リカバリポイント RPO の決定論的な時間ベースの値を提供します。 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。

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.cnpg.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。

第20章　ボリュームスナップショットのバックアップ

警告: ボリュームスナップショットバージョン 1.21.0 の初期リリースは、インスタンスのフェンシングが必要なコールドバックアップのみをサポートしていました。この制限は、バージョン 1.21.1 から免除されました。コードへの変更の影響が最小限に抑えられることを考慮して、メンテナーは、バージョン 1.22.0 がリリースされるのを待つ代わりにこの機能をすぐにバックポートし、オンラインバックアップをボリュームスナップショットのデフォルト動作にすることを決定しました。代わりにコールドバックアップに依存する予定がある場合は、以下の手順に従ってください。

警告: *backup document* で述べたように、プライマリをターゲット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、その間クラスターを読み取り専用にします 安全のため、既にフェンスされたインスタンスを含むクラスターでは、コールドスナップショットは拒否されます。

CloudNativePG は、完全に宣言的な方法で、バックアップとリカバリー操作の両方に Kubernetes ネイティブボリュームスナップショット API を直接利用するデータベースオペレーターの最初の既知のケースの 1 つです。

20.1 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。

20.2 要件

ボリュームスナップショットが CloudNativePG クラスターで動作するには、PostgreSQL ボリュームつまり `storage` および `walStorage` セクションの動的プロビジョニング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 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。

20.3 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 では、`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 {postgresql-cnpg-io-v1- 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.cnpg.io/v1
kind: Cluster
metadata:
  name: snapshot-cluster
spec:
  instances: 3

  storage:
    storageClass: @STORAGE_CLASS@
```

(次のページに続く)

(前のページからの続き)

```
    size: 10Gi
  walStorage:
    storageClass: @STORAGE_CLASS@
    size: 10Gi

  backup:
    # Volume snapshot backups
    volumeSnapshot:
       className: @VOLUME_SNAPSHOT_CLASS_NAME@
    # WAL archive
    barmanObjectStore:
       # ...
```

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 barmanObjectStore スタンザ *WAL アーカイブ* の制御の両方が含まれています。

注釈: *barmanObjectStore* を定義すると、ボリュームスナップショットとオブジェクトストアの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得することができます。

---

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。

20.4 ホットおよびコールドバックアップ

デフォルトでは、CloudNativePG 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
    # ...
```

20.4.1 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.cnpg.io/v1
kind: Backup
metadata:
  name: snapshot-cluster-cold-backup-example
```

(次のページに続く)

(前のページからの続き)

```
spec:
  cluster:
    name: snapshot-cluster
  method: volumeSnapshot
  online: false
```

同様に、 ScheduledBackup の場合

```
apiVersion: postgresql.cnpg.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
```

20.5 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 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、

20.6 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 <!--wakeignore:rule=master -->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 Cluster を作成します *AWS S3* を参照してください。

```
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: hendrix
spec:
  instances: 3

  storage:
    storageClass: ebs-sc
    size: 10Gi
  walStorage:
    storageClass: ebs-sc
    size: 10Gi

  backup:
    volumeSnapshot:
       className: csi-aws-vsc
    barmanObjectStore:
      destinationPath: s3://@BUCKET_NAME@/
      s3Credentials:
        inheritFromIAMRole: true
```

(次のページに続く)

(前のページからの続き)

```
      wal:
        compression: gzip
        maxParallel: 2

  serviceAccountTemplate:
    metadata:
      annotations:
        eks.amazonaws.com/role-arn: "@ARN@"
- --
apiVersion: postgresql.cnpg.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。

第21章　リカバリー

PostgreSQL の用語では、リカバリとは、既存のバックアップを使用して PostgreSQL インスタンスを起動するプロセスです。PostgreSQL のリカバリメカニズムは非常に堅牢で豊富です。また、ポイントインタイムリカバリー PITR もサポートしており、カタログ内の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、特定のクラスターを任意の時点に復元できます。(この場合、WAL アーカイブは必須です。)

CloudNativePG では、既存のクラスターでリカバリーインプレースを実行できません。リカバリーは、代わりに、使用可能な物理的バックアップから新しい Postgres クラスターをブートストラップする方法です。

注釈: *bootstrap* スタンザの詳細については、 *Bootstrap* を参照してください。

`recovery` ブートストラップモードでは、既存の物理ベースバックアップからクラスターを作成できます。次に、アーカイブから REDO ログを含む WAL ファイルを再適用します。

WAL ファイルは、定義された **リカバリオブジェクトストア**からプルされます。

ベースバックアップは、オブジェクトストアまたはボリュームスナップショットバージョン 1.21 以降を使用して取得できます。

> **警告:** ボリュームスナップショットを使用したリカバリーの初期リリースは 1.20.1 でした。 1.21.0 の機能の進歩の量のため、ボリュームスナップショットを使用するには、1.21 以上の高度なリリースにアップグレードすることを強くお勧めします。

2 つの方法で **リカバリオブジェクトストア** からのリカバリを実現できます。

- リカバリオブジェクトストア、つまり Barman Cloud によって作成され、`externalClusters` セクションの `barmanObjectStore` オプションを介して定義された別のクラスターのバックアップを使用することをお勧めします。
- または、同じ名前空間内の既存の `Backup` オブジェクトを使用できます。

どちらの復旧方法も、完全な復旧最後に使用可能な WAL までまたは最大
ポイント インタイム リカバリー *PITR*

までのいずれかが有効になります。完全なリカバリーを実行するときに、クラスターをレプリカモードで起動することもできます。 **レプリカクラスター** を参照してください。

重要: レプリカモードを使用する場合、復旧したクラスターの PostgreSQL 構成`.spec.postgresql.parameters`

が、物理レプリケーションの観点から元の構成と互換性があることを確認します。

ボリュームスナップショット を使用したリカバリの場合

- すべてが同じバックアップに属し、同じ `cnpg.io/cluster` および `cnpg.io/backupName` ラベルで識別される `VolumeSnapshot` オブジェクトの一貫したセットを使用します。次に、:ref:`VolumeSnapshot` オブジェクトからのリカバリー <`VolumeSnapshot` オブジェクトからのリカバリー>` の説明に従って、 ``.spec.bootstrap.recovery` スタンザの `volumeSnapshots` オプションを使用してリカバリします。

21.1 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.cnpg.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
```

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。

21.2 VolumeSnapshot オブジェクトからのリカバリー

警告: ボリュームスナップショット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なる場合があります。この動作により、データベースのサイズによっては、プロセスが遅くなります。この制限は、将来的にオンラインバックアップと PVC クローン作成のサポートが導入されると解除されます。

CloudNativePG は、 volume snapshot backups の宣言 API を使用して取得された既存の `Cluster` の PVC の `VolumeSnapshot` から新しいクラスターを作成できます。次の例のように、スナップショットの名前を指定する必要があります。

```yaml
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: cluster-restore
spec:
  [...]

  bootstrap:
    recovery:
      volumeSnapshots:
        storage:
          name: <snapshot name>
          kind: VolumeSnapshot
          apiGroup: snapshot.storage.k8s.io
```

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.cnpg.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
```

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使用可能な WAL を使用して復旧されます。ソースクラスタから。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。

21.3 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.cnpg.io/v1
kind: Cluster
metadata:
  name: cluster-example-initdb
spec:
  instances: 3

  bootstrap:
    recovery:
      backup:
```

(次のページに続く)

(前のページからの続き)

```
      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 であることを示しています。復元する PostgreSQL クラスターが別の名前を使用していた場合、*Configure the application database* の文書に従ってそれらを指定できます。

21.4 追加の考慮事項

リカバリ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するかどうかにかかわらず、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アプリケーションデータベースユーザーは、復元されるバックアップから保持されます。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を試みていません。
- 元の postgres ユーザーパスワードを保持するには、enableSuperuserAccess を適切に構成し、superuserSecret を提供する必要があります。
- 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 で最新の利用可能な 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。

21.5 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。

21.5.1 オブジェクトストアからの PITR

この例では、ベースバックアップと WAL アーカイブの両方を含む Azure のリカバリオブジェクトストアを使用します。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。

```
apiVersion: postgresql.cnpg.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: clusterBackup
      recoveryTarget:
        # Time base target for the recovery
        targetTime: "2023-08-11 11:14:21.00000+02"

  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
```

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。

21.5.2 VolumeSnapshot オブジェクトからの PITR

次の例では、以下を使用します。

- リカバリプロセスを開始するベースバックアップを含む `PGDATA` の Kubernetes ボリュームスナップショット。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` と呼ばれます。
- WAL アーカイブを含む MinIO のリカバリオブジェクトストア。オブジェクトストアは、外部クラスター定義の形式で `recovery.source` オプションで指定されます。

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。

```
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: cluster-example-snapshot
spec:
  # ...
  bootstrap:
    recovery:
      source: cluster-example-with-backup
      volumeSnapshots:
        storage:
          name: test-snapshot-1
          kind: VolumeSnapshot
          apiGroup: snapshot.storage.k8s.io
      recoveryTarget:
        targetTime: "2023-07-06T08:00:39"
  externalClusters:
    - name: cluster-example-with-backup
      barmanObjectStore:
        destinationPath: s3://backups/
        endpointURL: http://minio:9000
        s3Credentials:
          accessKeyId:
```

(次のページに続く)

(前のページからの続き)

```
      name: minio
      key: ACCESS_KEY_ID
    secretAccessKey:
      name: minio
      key: ACCESS_SECRET_KEY
```

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。

21.5.3 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。

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、カタログ内の最後の使用可能なバックアップが受け入れられる場合を除き、*backupID* を指定することが重要です。

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.cnpg.io/v1
kind: Cluster
[...]
  bootstrap:
    recovery:
      source: clusterBackup
      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.cnpg.io/v1
kind: Cluster
metadata:
  name: cluster-restore-pitr
spec:
  instances: 3

  storage:
    size: 5Gi

  bootstrap:
    recovery:
      source: clusterBackup
      recoveryTarget:
        backupID: 20220616T142236
        targetName: "maintenance-activity"
        exclusive: true

  externalClusters:
    - name: clusterBackup
      barmanObjectStore:
        destinationPath: https://STORAGEACCOUNTNAME.blob.core.windows.net/
↪CONTAINERNAME/
        azureCredentials:
          storageAccount:
```

(次のページに続く)

(前のページからの続き)

```
          name: recovery-object-store-secret
          key: storage_account_name
        storageKey:
          name: recovery-object-store-secret
          key: storage_account_key
      wal:
        maxParallel: 8
```

21.6 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、

この例では、所有者が app 、提供されたシークレットが app-secret を使用してアプリケーションデータベース app を構成します。

```
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
[...]
spec:
  bootstrap:
    recovery:
      database: app
      owner: app
      secret:
        name: app-secret
      [...]
```

この構成では、復旧の完了後に次が発生します。

1. データベース app が存在しない場合、新しいデータベース app が作成されます。
2. ユーザー app が存在しない場合、新しいユーザー app が作成されます。
3. ユーザー app がデータベースの所有者でない場合、ユーザー app はデータベース app の所有者として付与されます。
4. username の値がシークレットの owner の値と一致する場合、アプリケーションデータベースのパスワードはシークレットの password の値に変更されます。

重要: レプリカモードが有効になっているレプリカクラスターの場合、オペレーターは PostgreSQL インスタンスにデータベースもユーザーも作成しません。これらは、元のクラスターから復元されます。

21.7 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 の方法がよくわからない場合

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 の最後または 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、リカバリは上の最新の使用可能な WAL まで継続されます。デフォルトのターゲットタイムライン `latest`

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。

21.8 バックアップセクションを使用したクラスターへのリストア

クラスターリストアのマニフェストには、backup セクションが含まれる場合があります。これは、復旧後に、新しいクラスターが WAL のアーカイブと、構成されている場合にバックアップの取得を開始することを意味します。

たとえば、このセクションは、クラスター cluster-example-backup からブートストラップするクラスターのマニフェストの一部です。ストレージバケットに、recoveredCluster という名前のフォルダーを作成します。ここに、復旧したクラスターのベースバックアップと WAL が保存されます。

```
backup:
  barmanObjectStore:
    destinationPath: s3://backups/
    endpointURL: http://minio:9000
    serverName: "recoveredCluster"
    s3Credentials:
      accessKeyId:
        name: minio
        key: ACCESS_KEY_ID
      secretAccessKey:
        name: minio
        key: ACCESS_SECRET_KEY
  retentionPolicy: "30d"

externalClusters:
- name: cluster-example-backup
  barmanObjectStore:
    destinationPath: s3://backups/
    endpointURL: http://minio:9000
    s3Credentials:
```

異なるクラスターで同じ barmanObjectStore 構成を再利用しないでください。ストレージバケットの既存の情報が新しいクラスターによって上書きされる場合があります。

> **警告:** オペレーターには、クラスターが情報を含むストレージバケットを上書きしないことを確認する安全チェックが含まれます。既存のストレージを上書きするクラスターは、ポッドがエラー状態の `Setting up primary` 状態のままです。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 `cnpg.io/skipEmptyWalArchiveCheck` アノテーションを `enabled` に設定すると、安全性チェックをスキップできます。一般的な使用ケースでは、チェックは正常に機能するため、チェックをスキップすることはお勧めしません。重大なデータの損失が発生する可能性があるため、PostgreSQL リカバリシステムに精通している場合にのみこのチェックをスキップしてください。

第22章　PostgreSQLの構成

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。以下を参照してください。

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

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

(次のページに続く)

(前のページからの続き)

```
    - host all all all md5

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

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

。

22.1 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 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。

```
dynamic_shared_memory_type = posix
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 # for PostgreSQL >= 12 only
wal_keep_size = 512MB # for PostgreSQL >= 13 only
wal_keep_segments = 32 # for PostgreSQL <= 12 only
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` を適切に構成するのはあなたの義務です。

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

レプリケーションユーザーについて 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
archive_mode = on
full_page_writes = on
hot_standby = true
listen_addresses = *
port = 5432
restart_after_crash = false
```

(次のページに続く)

(前のページからの続き)

```
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
wal_level = logical
wal_log_hints = on
```

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。

22.1.1 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状態に応じて、オペレーターによって自動的に管理されます。

```
primary_conninfo = host=cluster-example-rw user=postgres dbname=postgres
recovery_target_timeline = latest
```

22.1.2 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、 ロギング を参照してください。

22.1.3 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 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 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 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。

22.1.4 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 は、一部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 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で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。

22.1.5 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。

22.1.6 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。

22.1.7 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 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、
PGAudit ログ 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。

yaml #postgresql

パラメーター pgaudit.log: “all, -misc” pgaudit.log_catalog: “off”pgaudit.log_parameter: “on” pgaudit.log_relation: “on” #

22.1.8 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 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。

the 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
```

22.2 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
hostssl replication streaming_replica all cert
```

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
hostssl replication streaming_replica all cert

<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。

22.2.1 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:
      baseDN: ou=org,dc=example,dc=com
      bindDN: cn=admin,dc=example,dc=com
      bindPassword:
        name: ldapBindPassword
        key: data
      searchAttribute: uid
```

22.3 pg_ident セクション

pg_ident は、CloudNativePG 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>`_ については、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"
```

22.4 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。

22.5 `ALTER SYSTEM` の有効化

CloudNativePG 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 バージョン 1.22 以降では、デフォルト構成では、新しい Postgres クラスターでの `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 は、特定の固定パラメーターに対する責任とその他の完全な制御を引き受け、注意深い検討の必要性を強調します。

`.spec.postgresql.enableAlterSystem` が `false` として構成されている場合、 `ALTER SYSTEM` を実行しようとするとエラーが発生します。エラーメッセージは次のような場合があります。

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

22.6 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 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。

22.6.1 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
```

22.6.2 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
```

22.7 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_system_table_mods
- archive_cleanup_command
- archive_command
- archive_mode
- bonjour
- bonjour_name
- cluster_name
- config_file
- data_directory

- data_sync_retry
- event_source
- external_pid_file
- full_page_writes
- 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
- wal_level
- wal_log_hints

第23章　データベースロール管理

CloudNativePG は、当初から、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` 、`cnpg_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、 *Bootstrap* 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 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。

宣言的な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、次のファイルにあります。

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

  managed:
    roles:
    - name: app
      createdb: true
      login: true
    - name: dante
      ensure: present
      comment: my database-side comment
      login: true
      superuser: false
      createdb: true
      createrole: false
      inherit: false
      replication: false
      bypassrls: false
```

(次のページに続く)

(前のページからの続き)

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

。

次に、そのファイルからの抜粋を示します。

```
apiVersion: postgresql.cnpg.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、CloudNativePG 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。

23.1 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:
    cnpg.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 CloudNativePG 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。

23.1.1 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: 宣言的ロール管理機能は、初期バージョン 1.20.0 から動作が変更されました。 1.20.0 では、*passwordSecret* のないロールは、PostgreSQL でパスワードを NULL に設定することになります。実際には、1.20.0 とほとんど違いはありません。 *passwordSecret* を使用せずに作成された新しいロールのパスワードは`NULL` になります。関連する変更は、管理対象ロールを使用して、以前に作成したロールを管理する場合です。 1.20.0 では、これを行うと、既存のパスワードを誤って`NULL` に設定してしまう場合があります。

23.1.2 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:
    cnpg.io/reload: "true"
apiVersion: v1
stringData:
  username: cavalcanti
  password: SCRAM-SHA-256$<iteration count>:<salt>$<StoredKey>:<ServerKey>
```

23.2 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、人間の管理者の説明がない限り、データベースまたは CloudNativePG オペレーターによって修正できません。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 は、このような基本的なエラーが発生すると記録し、クラスターステータスに表示します。どちらが続くか…

23.3 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 CloudNativePG 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 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 内で管理されるため、CloudNativePG ユーザーは自分の都合に合わせてこの機能を採用できます。

第24章　テーブルスペース

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。

24.1 宣言的テーブルスペース

CloudNativePG 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 CloudNativePG の宣言的テーブルスペースは、永続ボリューム要求および永続ボリュームを活用して実装されます。クラスターで定義された各テーブルスペースは、独自の永続ボリュームに格納されます。CloudNativePG 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 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。

24.2 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。完全な例は次のとおりです。

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

  storage:
    size: 1Gi

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

。

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
```

(次のページに続く)

(前のページからの続き)

```
    - 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 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
```

24.3 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.cnpg.io/v1
kind: Cluster
metadata:
  name: yardbirds
spec:
  instances: 3

  storage:
    size: 10Gi
  walStorage:
    size: 10Gi
  tablespaces:
    - name: current
      size: 100Gi
      storageClass: fastest
    - name: this_year
      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。

24.4 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 はテーブルスペースを作成できず、これをクラスターステータスに反映します。

```
spec:
  instances: 3

  # ...

  tablespaces:
    - name: tbs1
      storage:
        size: 1Gi
    - name: tbs2
      storage:
        size: 2Gi
    - name: tbs3
      owner:
        name: badhombre
      storage:
        size: 2Gi
        status:

  <- snipped ->
  tablespacesStatus:
```

(次のページに続く)

(前のページからの続き)

```
  - 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
```

24.5 バックアップ リカバリー

CloudNativePG 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。

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。

24.6 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:
```

(次のページに続く)

(前のページからの続き)

```
    - name: tbs1
      storage:
        size: 1Gi
    - name: tbs2
      storage:
        size: 2Gi
    - name: tbs3
      storage:
        size: 2Gi
```

24.7 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 は、 `.spec.tablespaces[*].temporary` オプションを提供して、テーブルスペースを `temp_tablespaces` PostgreSQL パラメーターに追加し、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かどうかを決定します。

```
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。

24.8 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
[...]
```

24.9 制限事項

現在、既存の CloudNativePG 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。

第25章　オペレーター設定

CloudNativePG 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 `cnpg-controller-manager` と呼ばれる Kubernetes `Deployment` として `cnpg-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 `cnpg-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。

25.1 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。

名前 |説明 —- | ——— – `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 `PULL_SECRET_NAME` |オペレーターの名前空間で定義され、画像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 `ENABLE_AZURE_PVC_UPDATES` | PVC がサイズ変更状態でスタックしている場合、Postgres ポッドを削除できます。この機能は、主に Azure 環境デフォルト `false` `ENABLE_INSTANCE_MANAGER_INPLACE_UPDATES` | `true` に設定している場合、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を有効にし、クラスターのローリング更新を回避しますデフォルト false 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 CREATE_ANY_SERVICE | オペレーターの名前空間のシークレットの名前 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

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。

> **警告:** オペレーターの以前のバージョンでは、PostgreSQL クラスターを展開する名前空間内に `PULL_SECRET_NAME` シークレットをコピーしました。バージョン「1.11.0」から、以前の説明と一致するように動作が変更されました。オペレーターの以前のバージョンによって作成されたプルシークレットは使用されません。

25.2 オペレーター構成マップの定義

次の例では、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し、in-place updates for the instance manager を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。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cnpg-controller-manager-config
  namespace: cnpg-system
data:
  INHERITED_ANNOTATIONS: categories
  INHERITED_LABELS: environment, workload, app
  ENABLE_INSTANCE_MANAGER_INPLACE_UPDATES: true
```

25.3 オペレーターシークレットの定義

次の例では、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し、in-place updates for the instance manager を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。

```
apiVersion: v1
kind: Secret
metadata:
  name: cnpg-controller-manager-config
  namespace: cnpg-system
type: Opaque
stringData:
  INHERITED_ANNOTATIONS: categories
  INHERITED_LABELS: environment, workload, app
  ENABLE_INSTANCE_MANAGER_INPLACE_UPDATES: true
```

25.4 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 cnpg-system \
    cnpg-controller-manager
```

通常、特定の名前空間を指定すると、次のコマンドでオペレーターポッドを削除できます。

```
kubectl delete pods -n [NAMESPACE_NAME_HERE] \
  -l app.kubernetes.io/name=cloudnative-pg
```

> **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。

25.5 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 cnpg-system cnpg-controller-manager
```

次に、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。例

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

変更を保存すると、展開でロールアウトが実行され、新しいポッドで 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 cnpg-system <pod name> -- bash
```

内部に入ったら、次を実行します。

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

第26章　インスタンスポッド構成

26.1 予測ボリューム

CloudNativePG 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 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。

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

投影されたボリュームテンプレートを使用してシークレットと ConfigMap をマウントする完全な例は、

```yaml
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: cluster-example-projected-volume
spec:
```

(次のページに続く)

(前のページからの続き)

```
  instances: 3
  projectedVolumeTemplate:
    sources:
      - secret:
          name: sample-secret
          items:
            - key: tls.crt
              path: certificate/tls.crt
            - key: tls.key
              path: certificate/tls.key
      - configMap:
          name: sample-configmap
          items:
            - key: key1
              path: config/key1
            - key: key2
              path: config/key2
  storage:
    size: 1Gi

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

展開マニフェスト。

26.2 エフェメラルボリューム

CloudNativePG 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。

第27章　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.cnpg.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。

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 の構成>` の場合にのみ使用します。

27.1 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 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.cnpg.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.cnpg.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3

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

  storage:
    size: 1Gi
```

オペレーターは、次の環境変数を設定することを許可していません。

- POD_NAME

- NAMESPACE
- 名前が PG で始まる環境変数。

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

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

第28章　ストレージ

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 で 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 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、 **カスタムポッドコントローラ** を参照してください。

28.1 バックアップ リカバリー

CloudNativePG 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。

28.2 CloudNativePG のベンチマーク

データベースを実稼働環境に展開する前に、制御された Kubernetes 環境で CloudNativePG のベンチマークを行うことをお勧めします。
ベンチマーク 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。

28.3 保存時の暗号化

CloudNativePG では、保存時の暗号化が可能です。オペレーターは、それを基になるストレージクラスに委任します。この重要なセキュリティ機能の詳細については、ストレージクラスを参照してください。

28.4 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
- TablespaceState {postgresql-cnpg-io-v1-TablespaceState} 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 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。

28.5 ストレージクラスを介した構成

重要: CloudNativePG 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ました。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.cnpg.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.cnpg.io/v1
kind: Cluster
metadata:
  name: postgresql-storage-class
spec:
  instances: 3
  storage:
    storageClass: standard
    size: 1Gi
```

28.6 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.cnpg.io/v1
kind: Cluster
metadata:
  name: postgresql-pvc-template
spec:
  instances: 3

  storage:
    pvcTemplate:
      accessModes:
        - ReadWriteOnce
      resources:
        requests:
          storage: 1Gi
      storageClassName: standard
      volumeMode: Filesystem
```

28.7 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し、WAL configuration とチェックポイントを微調整し、コスト最適化のために別のストレージクラスを使用することもできます。
- **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.cnpg.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。

28.8 テーブルスペースのボリューム

CloudNativePG 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、
TablespaceState {postgresql-cnpg-io-v1-TablespaceState} を参照してください。

28.9 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
```

28.9.1 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。

28.9.2 AKS での PVC ボリュームの拡張

現在、 Azure can resize the PVC's volume without restarting the pod only on specific regions .
CloudNativePG は、
オペレーター設定 の `ENABLE_AZURE_PVC_UPDATES`

環境変数を介してこの制限を克服しました。 `true` に設定すると、CloudNativePG は Postgres クラスターのローリング更新をトリガーします。

または、次の回避策を使用して、AKS のボリュームのサイズを手動で変更できます。

28.9.3 AKS のボリューム拡張の回避策

AKS で PVC のサイズを手動で変更できます。例として、3 つのレプリカを持つ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

`Kubernetes documentation <https://github.com/kubernetes-sigs/azuredisk-csi-driver/
↪blob/master/docs/known-issues/sizegrow.md>`_ で説明しているように、Azure ディスクは「未
接続」状態にあるときにのみ拡張できます。
```

これは、PostgreSQL クラスターが使用するディスクのサイズを変更するには、手動ロールアウトを実行し、最初にディスクにバインドされた PVC を使用してポッドをホストするノードを遮断する必要があることを意味します。これにより、バックグラウンドでのディスクのサイズ変更が完了する前に、オペレーターがポッドを再作成し、すぐに PVC に再接続することを防ぎます。

最初に、クラスター定義を編集し、新しいサイズを適用します。この例では、新しいサイズは `2Gi` です。

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

  storage:
    storageClass: default
    size: 2Gi
```

`cluster-example-1` ポッドがクラスターのプライマリであると仮定すると、最初にレプリカの使用を続行できます。たとえば、`cluster-example-3` ポッドをホストする Kubernetes ノードを遮断することから始めます。

```
kubectl cordon <node of cluster-example-3>
```

次に、 `cluster-example-3` ポッドを削除します。

```
$ kubectl delete pod/cluster-example-3
```

次のコマンドを実行します。

```
kubectl get pvc -w -o=jsonpath={.status.conditions[].message} cluster-example-3
```

次の出力が表示されるまで待ちます。

```
Waiting for user to (re-)start a Pod to finish file system resize of volume on node.
```

次に、ノードの遮断を解除できます。

```
kubectl uncordon <node of cluster-example-3>
```

ポッドが正しく再作成され、「Running and Ready」状態になるのを待ちます。

```
kubectl get pods -w cluster-example-3
cluster-example-3   0/1     Init:0/1   0          12m
cluster-example-3   1/1     Running    0          12m
```

次のコマンドを実行して、PVC 拡張を確認します。これは、構成どおりに `2Gi` を結果ます。

```
kubectl get pvc cluster-example-3 -o=jsonpath={.status.capacity.storage}
```

残りのポッドに対してこれらの手順を繰り返すことができます。

重要: *kubectl cnpg promote* を使用して、新しいサイズ変更されたポッドをスイッチオーバーでプロモートした後、プライマリインスタンスに関連付けられているディスクのサイズ変更を最後のディスクとして残します。たとえば、 *kubectl cnpg promote cluster-example 3* を使用して `cluster-example-3` をプライマリに昇格させます。

28.9.4 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。

```yaml
apiVersion: postgresql.cnpg.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
```

28.10 永続ボリュームの静的プロビジョニング

CloudNativePG 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 が構築された完全な宣言型モデルを中断します。 CloudNativePG 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` と一致し、CloudNativePG が必要な `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。

28.11 ブロックストレージの考慮事項 Ceph/Longhorn

Kubernetes のブロックストレージソリューションのほとんどは、復元性を向上させ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。これは、アプリケーションに復元力が組み込まれていないワークロードに適しています。ただし、CloudNativePG は、インスタンスの数とそれらに接続されている永続ボリュームを介して Postgres `Cluster` に直接組み込まれているこの復元力を備えています。

これらの場合、Postgres クラスターが使用するストレージクラスを 1 つのレプリカとして定義することが合理的です。Longhorn や Ceph などのストレージソリューションで追加のレプリカを定義すると、書き込み増幅と呼ばれる現象が発生し、ディスク I/O と使用領域が不必要に増加する場合があります。

第29章　ラベルとアノテーション

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 展開で使用する 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 によって採用された技術です。

29.1 事前定義されたラベル

これらの事前定義されたラベルは CloudNativePG によって管理されます。

`cnpg.io/backupName` バックアップ識別子、`Backup` および `VolumeSnapshot` リソースでのみ使用可能

`cnpg.io/cluster` クラスターの名前

`cnpg.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

`cnpg.io/instanceName` PostgreSQL インスタンスの名前古い非推奨の `postgresql` ラベルを置き換えます

`cnpg.io/jobRole` ジョブのロールつまり `import` 、`initdb` 、`join` , …

`cnpg.io/podRole` ポッドのロール `instance` 、または `pooler`

`cnpg.io/poolerName` PgBouncer プーラーの名前

cnpg.io/pvcRole PG_DATA や PG_WAL などの PVC の目的

cnpg.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。

cnpg.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

role 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

cnpg.io/backupTimeline バックアップが取られたインスタンスのタイムライン

cnpg.io/backupYear バックアップが取られた年

cnpg.io/backupMonth バックアップが取られた年/月

cnpg.io/backupDate ISO 8601 形式のバックアップの日付 YYYYMMDD

cnpg.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

29.2 事前定義されたアノテーション

これらの事前定義されたアノテーションは CloudNativePG によって管理されます。

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。

cnpg.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、

cnpg.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 cnpg.io/hibernateClusterManifest ラベルを置き換えます。

cnpg.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。

cnpg.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。

cnpg.io/hash リソースのハッシュ値。

cnpg.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。

cnpg.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。

cnpg.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。

cnpg.io/operatorVersion オペレーターのバージョン。

cnpg.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 cnpg.io/hibernatePgControlData アノテーションを置き換えます。

cnpg.io/podEnvHash cnpg.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。

cnpg.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 cnpg.io/podEnvHash アノテーションを置き換えます。

cnpg.io/poolerSpecHash プーラーリソースのハッシュ。

cnpg.io/pvcStatus PVC の現在のステータス initializing 、ready 、または detached 。

cnpg.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。

cnpg.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。

cnpg.io/skipEmptyWalArchiveCheck Cluster リソースで true 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。

cnpg.io/backupStartWAL バックアップ開始時の WAL。

cnpg.io/backupEndWAL バックアップ終了時の WAL。

cnpg.io/backupStartTime バックアップが開始された時刻。

cnpg.io/backupEndTime バックアップが終了した時刻。

cnpg.io/snapshotStartTime スナップショットが開始された時刻。

cnpg.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。

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。

29.3 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、
オペレーター設定 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。

29.4 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.cnpg.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
```

29.5 現在の制限

現在、CloudNativePG は、ラベルまたはアノテーション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。

第30章　モニタリング

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。

30.1 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 を介してポート 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: バージョン 1.11 から、CloudNativePG は既に`default-monitoring` と呼ばれる `ConfigMap` に by default a set of predefined metrics をインストールしています。

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。

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 `cnpg_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 と Prometheus および Grafana 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。

30.1.1 Prometheus Operator の例

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、オペレーターがクラスターを正しくポイントする PodMonitor を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、必要に応じて変更することができます。

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: cluster-example
spec:
  selector:
    matchLabels:
      "cnpg.io/cluster": cluster-example
  podMetricsEndpoints:
  - port: metrics
```

重要: 一意の名前と、正しいクラスターの名前空間とラベルで上記の例を変更してください *cluster-example* を使用しています。

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ラベル *postgresql* は非推奨であり、将来的に削除される予定です。代わりにラベル *cnpg.io/cluster* を使用して、インスタンスを選択してください。

30.1.2 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。

- cnpg_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 cnpg_collector_collection_duration_seconds Collection time duration in seconds

#  TYPE cnpg_collector_collection_duration_seconds gauge

cnpg_collector_collection_duration_seconds{collector="Collect.up"} 0.0031393

#  HELP cnpg_collector_collections_total Total number of times PostgreSQL was␣
↪accessed for metrics.

#  TYPE cnpg_collector_collections_total counter

cnpg_collector_collections_total 2

#  HELP cnpg_collector_fencing_on 1 if the instance is fenced, 0 otherwise

#  TYPE cnpg_collector_fencing_on gauge
```

(次のページに続く)

(前のページからの続き)

```
cnpg_collector_fencing_on 0

#  HELP cnpg_collector_nodes_used NodesUsed represents the count of distinct nodes␣
↪accommodating the instances. A value of -1 suggests that the metric is not␣
↪available. A value of 1 suggests that all instances are hosted on a single node,␣
↪implying the absence of High Availability (HA). Ideally this value should match the␣
↪number of instances in the cluster.

#  TYPE cnpg_collector_nodes_used gauge

cnpg_collector_nodes_used 3

#  HELP cnpg_collector_last_collection_error 1 if the last collection ended with␣
↪error, 0 otherwise.

#  TYPE cnpg_collector_last_collection_error gauge

cnpg_collector_last_collection_error 0

#  HELP cnpg_collector_manual_switchover_required 1 if a manual switchover is␣
↪required, 0 otherwise

#  TYPE cnpg_collector_manual_switchover_required gauge

cnpg_collector_manual_switchover_required 0

#  HELP cnpg_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 cnpg_collector_pg_wal gauge

cnpg_collector_pg_wal{value="count"} 9
cnpg_collector_pg_wal{value="slots_max"} NaN
cnpg_collector_pg_wal{value="keep"} 32
cnpg_collector_pg_wal{value="max"} 64
cnpg_collector_pg_wal{value="min"} 5
cnpg_collector_pg_wal{value="size"} 1.50994944e+08
cnpg_collector_pg_wal{value="volume_max"} 128
cnpg_collector_pg_wal{value="volume_size"} 2.147483648e+09

#  HELP cnpg_collector_pg_wal_archive_status Number of WAL segments in the /var/lib/
↪postgresql/data/pgdata/pg_wal/archive_status directory (ready, done)

#  TYPE cnpg_collector_pg_wal_archive_status gauge
```

(次のページに続く)

(前のページからの続き)

```
cnpg_collector_pg_wal_archive_status{value="done"} 6
cnpg_collector_pg_wal_archive_status{value="ready"} 0

#  HELP cnpg_collector_replica_mode 1 if the cluster is in replica mode, 0 otherwise

#  TYPE cnpg_collector_replica_mode gauge

cnpg_collector_replica_mode 0

#  HELP cnpg_collector_sync_replicas Number of requested synchronous replicas␣
↪(synchronous_standby_names)

#  TYPE cnpg_collector_sync_replicas gauge

cnpg_collector_sync_replicas{value="expected"} 0
cnpg_collector_sync_replicas{value="max"} 0
cnpg_collector_sync_replicas{value="min"} 0
cnpg_collector_sync_replicas{value="observed"} 0

#  HELP cnpg_collector_up 1 if PostgreSQL is up, 0 otherwise.

#  TYPE cnpg_collector_up gauge

cnpg_collector_up{cluster="cluster-example"} 1

#  HELP cnpg_collector_postgres_version Postgres version

#  TYPE cnpg_collector_postgres_version gauge

cnpg_collector_postgres_version{cluster="cluster-example",full="16.1"} 16.1

#  HELP cnpg_collector_last_failed_backup_timestamp The last failed backup as a unix␣
↪timestamp

#  TYPE cnpg_collector_last_failed_backup_timestamp gauge

cnpg_collector_last_failed_backup_timestamp 0

#  HELP cnpg_collector_last_available_backup_timestamp The last available backup as a␣
↪unix timestamp

#  TYPE cnpg_collector_last_available_backup_timestamp gauge

cnpg_collector_last_available_backup_timestamp 1.63238406e+09

```

(次のページに続く)

(前のページからの続き)

```
#  HELP cnpg_collector_first_recoverability_point The first point of recoverability␣
↪for the cluster as a unix timestamp

#  TYPE cnpg_collector_first_recoverability_point gauge

cnpg_collector_first_recoverability_point 1.63238406e+09

#  HELP cnpg_collector_lo_pages Estimated number of pages in the pg_largeobject table

#  TYPE cnpg_collector_lo_pages gauge

cnpg_collector_lo_pages{datname="app"} 0
cnpg_collector_lo_pages{datname="postgres"} 78

#  HELP cnpg_collector_wal_buffers_full Number of times WAL data was written to disk␣
↪because WAL buffers became full. Only available on PG 14+

#  TYPE cnpg_collector_wal_buffers_full gauge

cnpg_collector_wal_buffers_full{stats_reset="2023-06-19T10:51:27.473259Z"} 6472

#  HELP cnpg_collector_wal_bytes Total amount of WAL generated in bytes. Only␣
↪available on PG 14+

#  TYPE cnpg_collector_wal_bytes gauge

cnpg_collector_wal_bytes{stats_reset="2023-06-19T10:51:27.473259Z"} 1.0035147e+07

#  HELP cnpg_collector_wal_fpi Total number of WAL full page images generated. Only␣
↪available on PG 14+

#  TYPE cnpg_collector_wal_fpi gauge

cnpg_collector_wal_fpi{stats_reset="2023-06-19T10:51:27.473259Z"} 1474

#  HELP cnpg_collector_wal_records Total number of WAL records generated. Only␣
↪available on PG 14+

#  TYPE cnpg_collector_wal_records gauge

cnpg_collector_wal_records{stats_reset="2023-06-19T10:51:27.473259Z"} 26178

#  HELP cnpg_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+
```

(次のページに続く)

(前のページからの続き)

```
#  TYPE cnpg_collector_wal_sync gauge

cnpg_collector_wal_sync{stats_reset="2023-06-19T10:51:27.473259Z"} 37

#  HELP cnpg_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 cnpg_collector_wal_sync_time gauge

cnpg_collector_wal_sync_time{stats_reset="2023-06-19T10:51:27.473259Z"} 0

#  HELP cnpg_collector_wal_write Number of times WAL buffers were written out to disk␣
↪via XLogWrite request. Only available on PG 14+

#  TYPE cnpg_collector_wal_write gauge

cnpg_collector_wal_write{stats_reset="2023-06-19T10:51:27.473259Z"} 7243

#  HELP cnpg_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 cnpg_collector_wal_write_time gauge

cnpg_collector_wal_write_time{stats_reset="2023-06-19T10:51:27.473259Z"} 0

#  HELP cnpg_last_error 1 if the last collection ended with error, 0 otherwise.

#  TYPE cnpg_last_error gauge

cnpg_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
```

(次のページに続く)

(前のページからの続き)

```
go_gc_duration_seconds{quantile="1"} 0.0012776
go_gc_duration_seconds_sum 0.0035741
go_gc_duration_seconds_count 13

#  HELP go_goroutines Number of goroutines that currently exist.

#  TYPE go_goroutines gauge

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
```

(次のページに続く)

(前のページからの続き)

```
go_memstats_gc_cpu_fraction 1.0749468700447189e-05

#  HELP go_memstats_gc_sys_bytes Number of bytes used for garbage collection system␣
↪metadata.

#  TYPE go_memstats_gc_sys_bytes gauge

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
```

(次のページに続く)

(前のページからの続き)

```
# HELP go_memstats_last_gc_time_seconds Number of seconds since 1970 of last garbage␣
↪collection.

# TYPE go_memstats_last_gc_time_seconds gauge

go_memstats_last_gc_time_seconds 1.6311727586032727e+09

# 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
```

(次のページに続く)

(前のページからの続き)

```
# HELP go_memstats_next_gc_bytes Number of heap bytes when next garbage collection␣
↪will take place.

# TYPE go_memstats_next_gc_bytes gauge

go_memstats_next_gc_bytes 8.689632e+06

# 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: *cnpg_collector_postgres_version* は、PostgreSQL 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。

注釈: *cnpg_collector_first_recoverability_point* および `cnpg_collector_last_available_backup_timestamp` は、オブジェクトストアへの最初のバックアップまで 0 になります。これは WAL アーカイブとは別です。

30.1.3 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.cnpg.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: 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 `cnpg.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 cnpg 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。

30.1.4 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:
    cnpg.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 展開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。

30.1.5 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
  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。

```
cnpg_some_query_rows{datname="albert"} 2
cnpg_some_query_rows{datname="bb"} 5
cnpg_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: |
```

(次のページに続く)

(前のページからの続き)

```
  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。

```
cnpg_some_query_rows{datname="albert"} 2
cnpg_some_query_rows{datname="bb"} 5
cnpg_some_query_rows{datname="freddie"} 10
cnpg_some_query_rows{datname="template1"} 7
cnpg_some_query_rows{datname="postgres"} 42
```

30.1.6 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
  - 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
  - master primary と同じ Prometheus PostgreSQL エクスポータの構文との互換性のため- 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に対して、または shell-like pattern

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。- metrics 次のように定義された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- <ColumnName> クエリーによって返された列の名前 - usage 以下で説明する値のいずれか - description メトリックの説明 - metrics_mapping usage が MAPPEDMETRIC に設定されている場合のオプションの列マッピング

usage の可能な値は次のとおりです。

表 1: :header: Column Usage Label,Description :widths: 10,30 :align: left :class: longtable

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 。

30.1.7 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。

```
cnpg_<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 cnpg_pg_replication_in_recovery Whether the instance is in recovery

#  TYPE cnpg_pg_replication_in_recovery gauge

cnpg_pg_replication_in_recovery 0
#  HELP cnpg_pg_replication_lag Replication lag behind primary in seconds

#  TYPE cnpg_pg_replication_lag gauge
```

(次のページに続く)

(前のページからの続き)

```
cnpg_pg_replication_lag 0
#  HELP cnpg_pg_replication_streaming_replicas Number of streaming replicas connected␣
↪to the instance

#  TYPE cnpg_pg_replication_streaming_replicas gauge

cnpg_pg_replication_streaming_replicas 2
#  HELP cnpg_pg_replication_is_wal_receiver_up Whether the instance wal_receiver is up

#  TYPE cnpg_pg_replication_is_wal_receiver_up gauge

cnpg_pg_replication_is_wal_receiver_up 0
```

30.1.8 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。
オペレーター設定 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 cnpg-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 cnpg-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`cnpg-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。

30.1.9 Prometheus Postgres エクスポーターとの違い

CloudNativePG は、PostgreSQL Prometheus Exporter からインスピレーションを得ていますが、いくつかの違いがあります。特に、`cache_seconds` フィールドは CloudNativePG のエクスポーターに実装されていません。

30.2 オペレーターのモニタリング

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。詳細については、

> kubebuilder documentation を参照してください。

30.2.1 Prometheus Operator の例

オペレーターの展開は、次の PodMonitor を定義することにより、 Prometheus Operator を使用して監視できます。

リソース

```yaml
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: cnpg-controller-manager
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: cloudnative-pg
  podMetricsEndpoints:
    - port: metrics
```

30.3 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、同じ名前空間で curl を実行する一時ポッドを使用して、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、PostgreSQL クラスターと並んで、デフォルトの名前空間で作業していることを前提としています。 Kubernetes の基本的な知識を適用して、この例をユースケースに自由に調整してください。

次の内容で curl.yaml ファイルを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: curl
spec:
  containers:
  - name: curl
    image: curlimages/curl:8.2.1
    command: [sleep, 3600]
```

次に、ポッドを作成します。

```
kubectl apply -f curl.yaml
```

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場合、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。これは、実行する汎用コマンドです。ポッドの正しい IP を使用していることを確認します。

```
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
```

オペレーターのメトリックにアクセスする場合は、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。

インスペクションの最後に、curl ポッドを削除していることを確認してください。

```
kubectl delete -f curl.yaml
```

30.4 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 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 Operator 展開の `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。

Grafana dashboard

は公式の Helm Chart プロジェクトの一部になりました。

`kube-prometheus-stack` の構成では、`kube-stack-config.yaml` で提供するものより他のフィールドと設定が利用できることに注意してください。

`helm show values prometheus-community/kube-prometheus-stack` を実行して表示できます。詳細については、

kube-prometheus-stack を参照してください。

ページ。

第31章　ロギング

オペレーターは、PostgreSQL ログを含む標準出力に JSON 形式で直接ログを記録するように設計されています。

各ログエントリには次のフィールドがあります。

- `level` – ログレベル `info` 、`notice` 、… 。
- `ts` – タイムスタンプマイクロ秒のエポック。
- `logger` – レコードのタイプたとえば `postgres` または `pg_controldata` など。
- `msg` – 実際のメッセージまたはキーワード `record` メッセージが JSON 形式で解析される場合。
- `record` – `logger` タイプによって異なる構造を持つ実際のレコード。
- `logging_podName` – ログが作成されたポッド。

> **警告:** 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 インストールのレベルで提供される必要があります。 *Kubernetes Logging Architecture* を参照してください

ドキュメント。

注釈: ログ取り込みシステムで必要な場合は、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` フィールド名を変更できます。 *cloudnative-pg* オペレーターの `Deployment` 定義を編集します。

31.1 オペレーターログ

オプション `logLevel` を使用して、クラスター仕様のログレベルを指定できます。 `error` 、`warning` 、`info` default、`debug` 、または `trace` に設定できます。

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。実行時に変更することはできません。クラスターが起動された後にクラスター仕様の値を変更する場合、新しいポッドでのみ有効になり、古いポッドでは有効になりません。

31.2 PostgreSQL ログ

PostgreSQL ログの各エントリは、postgres に設定された logger キーと次の例で説明する構造を持つ JSON オブジェクトです。


```
{
  "level": "info",
  "ts": 1619781249.7188137,
  "logger": "postgres",
  "msg": "record",
  "record": {
    "log_time": "2021-04-30 11:14:09.718 UTC",
    "user_name": "",
    "database_name": "",
    "process_id": "25",
    "connection_from": "",
    "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",
}
```


内部的には、オペレーターは PostgreSQL CSV ログ形式に依存しています。 CSV log format の詳細については、PostgreSQL ドキュメントを参照してください。

31.3 PGAudit ログ

CloudNativePG 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。

このサポートを有効にするには、必要な `pgaudit` パラメーターをクラスターの構成の `postgresql` セクションに追加します。

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 は、postgresql 構成内の `pgaudit.*` パラメーターの存在に基づいて、ライブラリを追加します。オペレーターは、*shared_preload_libraries* からのライブラリの追加と削除を検出および管理します。

オペレーターは、クラスター内の使用可能なすべてのデータベースから拡張機能の作成と削除も行います。

重要: CloudNativePG 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。

この例は、要求された構成で `pgaudit` が有効になる PostgreSQL 13 `Cluster` 展開を示しています。

```yaml
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3
  imageName: ghcr.io/cloudnative-pg/postgresql:13

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

  storage:
    size: 1Gi
```

PGAudit によって返された監査 CSV ログエントリは、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` 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。これは、JSON オブジェクトとしてフォーマットされた PGAudit CSV メッセージが含まれる `.record.audit` を除き `logging_collector` ログに似ています。

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。

31.4 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るすべてのログは JSON 形式であり、生成されたプロセスに応じて `logger` が設定されます。したがって、考えられるすべての `logger` 値は次のとおりです。

- `barman-cloud-wal-archive` `barman-cloud-wal-archive` から直接
- `barman-cloud-wal-restore` `barman-cloud-wal-restore` から直接
- `initdb` `initdb` の実行から

- pg_basebackup pg_basebackup の実行から
- pg_controldata pg_controldata の実行から
- pg_ctl pg_ctl サブコマンドの実行から
- pg_rewind pg_rewind の実行から
- pgaudit PGAudit 拡張機能から
- postgres postgres インスタンスから record とは異なる msg を持っています
- 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
- 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

前述の構造を持つ postgres を除き 、他のすべての可能な値は msg をログに記録されるエスケープメッセージに設定します。

第32章　証明書

CloudNativePG 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 は、TLS 証明書に関して非常に柔軟です。主に 2 つのモードで動作します。

1. **Operator managed** –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 によって作成された CA を使用して署名されます。
2. *__User provided__* – 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 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 CNPG 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。

32.1 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 CA を生成し、クライアントとサーバーの両方の証明書に使用します。これらは自動的に管理および更新されます。

32.1.1 サーバー証明書

32.1.2 サーバー CA シークレット

オペレーターは、自己署名 CA を生成し、次のキーを含む汎用シークレットに保存します。

- `ca.crt` – サーバー証明書の検証に使用される CA 証明書。クライアントの接続文字列で `sslrootcert` として使用されます。
- `ca.key` – サーバー SSL 証明書に自動的に署名するために使用されるキー。

32.1.3 サーバー TLS シークレット

オペレーターは、生成された自己署名 CA を使用して、サーバー TLS 証明書に署名します。これは `kubernetes.io/tls` タイプのシークレットに保存され、インスタンスによって `ssl_cert_file` および `ssl_key_file` として使用されるように構成されます。このアプローチにより、クライアントは自分の ID を確認し、安全に接続できます。

32.1.4 サーバー代替 DNS 名

デフォルトのものに加えて、生成されたサーバー TLS シークレットの一部として DNS サーバーの代替名を指定できます。

32.1.5 クライアント証明書

32.1.6 クライアント CA シークレット

デフォルトでは、サーバー CA と同じ自己署名 CA が使用されます。パブリック部分は `ssl_ca_file` としてすべてのインスタンスに渡されるため、署名したクライアント証明書を検証できます。プライベートキーは同じシークレットに保存され、`kubectl cnpg` 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。

32.1.7 クライアント `streaming_replica` `証明書

オペレーターは、生成された自己署名 CA を使用して、ユーザー `streaming_replica` のクライアント証明書に署名し、`kubernetes.io/tls` タイプのシークレットに保存します。プライマリインスタンスへの安全な接続を許可するために、この証明書はレプリカの接続文字列で `sslcert` および `sslkey` として渡されます。

32.2 ユーザー提供の証明書モード

32.2.1 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 などの別個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: 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 *cnpg.io/reload* のラベルを追加できます。それ以外の場合は、 *kubectl cnpg 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。

32.2.2 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.cnpg.io/v1
kind: Cluster
```

(次のページに続く)

(前のページからの続き)

```
metadata:
  name: cluster-example
spec:
  instances: 3
  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。

32.2.3 Cert-manager の例

この簡単な例は、 cert-manager の使用方法を示しています

自己署名 CA をセットアップし、必要な TLS サーバー証明書を生成するには。

```
- --
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: selfsigned-issuer
spec:
  selfSigned: {}
- --
apiVersion: v1
kind: Secret
metadata:
  name: my-postgres-server-cert
  labels:
    cnpg.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:
```

(次のページに続く)

(前のページからの続き)

```
    - 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.cnpg.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3
  certificates:
    serverTLSSecret: my-postgres-server-cert
    serverCASecret: my-postgres-server-cert
  storage:
    size: 1Gi
```

cert-manager を使用して、サーバーとクライアントの両方の CA と証明書を管理する完全な例を見つけることができます。

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: selfsigned-issuer
spec:
  selfSigned: {}
- --
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: server-ca
spec:
```

(次のページに続く)

(前のページからの続き)

```
  isCA: true
  commonName: my-selfsigned-server-ca
  secretName: server-ca-key-pair
  privateKey:
    algorithm: ECDSA
    size: 256
  issuerRef:
    name: selfsigned-issuer
    kind: Issuer
    group: cert-manager.io
- --
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: server-ca-issuer
spec:
  ca:
    secretName: server-ca-key-pair
- --
apiVersion: v1
kind: Secret
metadata:
  name: my-postgres-server-cert
  labels:
    cnpg.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
```

(次のページに続く)

(前のページからの続き)

```
  issuerRef:
    name: server-ca-issuer
    kind: Issuer
    group: cert-manager.io
- --
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: client-ca
spec:
  isCA: true
  commonName: my-selfsigned-client-ca
  secretName: client-ca-key-pair
  privateKey:
    algorithm: ECDSA
    size: 256
  issuerRef:
    name: selfsigned-issuer
    kind: Issuer
    group: cert-manager.io
- --
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: client-ca-issuer
spec:
  ca:
    secretName: client-ca-key-pair
- --
apiVersion: v1
kind: Secret
metadata:
  name: my-postgres-client-cert
  labels:
    cnpg.io/reload: ""
- --
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: my-postgres-client-cert
spec:
  secretName: my-postgres-client-cert
  usages:
    - client auth
  commonName: streaming_replica
```

(次のページに続く)

(前のページからの続き)

```
  issuerRef:
    name: client-ca-issuer
    kind: Issuer
    group: cert-manager.io
- --
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3
  certificates:
    serverTLSSecret: my-postgres-server-cert
    serverCASecret: my-postgres-server-cert
    clientCASecret: my-postgres-client-cert
    replicationTLSSecret: my-postgres-client-cert
  storage:
    size: 1Gi
```

展開マニフェスト。

32.2.4 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 cnpg 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 *cnpg.io/reload* のラベルを追加できます。それ以外の場合、 *kubectl cnpg 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。

32.2.5 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:
    cnpg.io/reload: ""
- --
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: my-postgres-client-cert
spec:
  secretName: my-postgres-client-cert
  usages:
    - client auth
  commonName: streaming_replica
  issuerRef:
    name: selfsigned-issuer
    kind: Issuer
    group: cert-manager.io
```

Cert-manager は、必要なすべてのファイルを含む `my-postgres-client-cert` という名前のシークレットを作成します。次のようにクラスターから参照できます。

```
apiVersion: postgresql.cnpg.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
  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を見つけることができます。

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: selfsigned-issuer
spec:
  selfSigned: {}
- --
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: server-ca
spec:
  isCA: true
  commonName: my-selfsigned-server-ca
  secretName: server-ca-key-pair
  privateKey:
    algorithm: ECDSA
    size: 256
  issuerRef:
    name: selfsigned-issuer
    kind: Issuer
    group: cert-manager.io
- --
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: server-ca-issuer
spec:
  ca:
    secretName: server-ca-key-pair
- --
apiVersion: v1
kind: Secret
metadata:
  name: my-postgres-server-cert
```

(次のページに続く)

(前のページからの続き)

```
  labels:
    cnpg.io/reload: ""
- --
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: my-postgres-server-cert
spec:
  secretName: my-postgres-server-cert
  usages:
    - server auth
  dnsNames:
    - cluster-example-lb.internal.mydomain.net
    - cluster-example-rw
    - cluster-example-rw.default
    - cluster-example-rw.default.svc
    - cluster-example-r
    - cluster-example-r.default
    - cluster-example-r.default.svc
    - cluster-example-ro
    - cluster-example-ro.default
    - cluster-example-ro.default.svc
  issuerRef:
    name: server-ca-issuer
    kind: Issuer
    group: cert-manager.io
- --
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: client-ca
spec:
  isCA: true
  commonName: my-selfsigned-client-ca
  secretName: client-ca-key-pair
  privateKey:
    algorithm: ECDSA
    size: 256
  issuerRef:
    name: selfsigned-issuer
    kind: Issuer
    group: cert-manager.io
- --
apiVersion: cert-manager.io/v1
kind: Issuer
```

(次のページに続く)

(前のページからの続き)

```
metadata:
  name: client-ca-issuer
spec:
  ca:
    secretName: client-ca-key-pair
- --
apiVersion: v1
kind: Secret
metadata:
  name: my-postgres-client-cert
  labels:
    cnpg.io/reload: ""
- --
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: my-postgres-client-cert
spec:
  secretName: my-postgres-client-cert
  usages:
    - client auth
  commonName: streaming_replica
  issuerRef:
    name: client-ca-issuer
    kind: Issuer
    group: cert-manager.io
- --
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3
  certificates:
    serverTLSSecret: my-postgres-server-cert
    serverCASecret: my-postgres-server-cert
    clientCASecret: my-postgres-client-cert
    replicationTLSSecret: my-postgres-client-cert
  storage:
    size: 1Gi
```

展開マニフェスト。

第33章　クライアントTLS/SSL接続

参考:

証明書 を参照

CloudNativePG 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 オペレーターは、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。CNPG 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 が付属します。kubectl の cnpg 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では、

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

  storage:
    size: 1Gi
```

マニフェスト。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。

33.1 新しい証明書の発行

参考:

Certificates in the CloudNativePG 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 cnpg certificate cluster-app \
  --cnpg-cluster cluster-example \
  --cnpg-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。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。

33.2 TLS 証明書を介した接続のテスト

次に、CloudNativePG クラスター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。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cert-test
spec:
  replicas: 1
  selector:
    matchLabels:
```

(次のページに続く)

(前のページからの続き)

```
    app: webtest
template:
  metadata:
    labels:
      app: webtest
  spec:
    containers:
      - image: ghcr.io/cloudnative-pg/webtest:1.6.0
        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
      - name: secret-volume-app
        secret:
          secretName: cluster-app
          defaultMode: 0600
```

このポッドは、CloudNativePG 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 cnpg 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 16.1 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 8.3.1 20191121 (Red Hat
8. 3.1-5), 64-bit
(1 row)
```

33.3 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

と ssl_max_protocol_version

`TLSv1.3` に。

重要: PostgreSQL 11 では、これら 2 つの GUC は存在しません。したがって、これらの特定のバージョンでは、これらの値は設定されず、デフォルト値が使用されます。

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。

第34章　アプリケーションから接続する

アプリケーションは、同じ Kubernetes クラスター内の CloudNativePG によって作成されたサービスと動作することになっています。

- `[cluster name]-rw`
- `[cluster name]-ro`
- `[cluster name]-r`

これらのサービスは Kubernetes クラスターによって完全に管理され、
Service で説明されている形式の仮想 IP を実装します。

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、 接続プーリング を参照してください。

34.1 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。

34.2 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指すサービスの 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

34.3 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、.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
- 作業用の .pgpass file
- uri
- jdbc-uri

-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。バージョン 1.21 以降、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。

第35章　接続プーリング

CloudNativePG 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 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。

35.1 アーキテクチャ

次の図は、PgBouncer に基づいたデータベースアクセスレイヤーの導入が CloudNativePG 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。

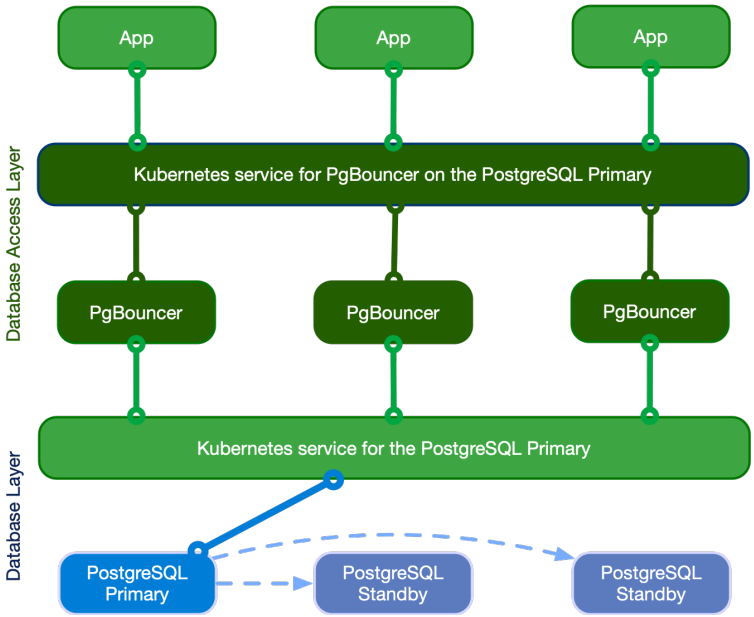


図 1: Applications writing to the single primary via PgBouncer

35.2 クイックスタート

この例は、CloudNativePG 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.cnpg.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 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 {postgresql-cnpg-io-v1-PgBouncerSpec}

API リファレンスで。

35.3 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。

35.4 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 サポート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。

35.4.1 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。

35.5 認証

パスワードベースの認証は、CloudNativePG 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 `cnpg_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、cnpg_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
- `cnpg_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 `cnpg_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 cnpg_pooler_pgbouncer WITH LOGIN;
```

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

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

最後に、各アプリケーションデータベースで接続し、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 user_search(uname TEXT)
  RETURNS TABLE (usename name, passwd text)
  LANGUAGE sql SECURITY DEFINER AS
  SELECT usename, passwd FROM pg_shadow WHERE usename=$1;;

REVOKE ALL ON FUNCTION user_search(text)
  FROM public;

GRANT EXECUTE ON FUNCTION user_search(text)
  TO cnpg_pooler_pgbouncer;
```

35.6 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 {postgresql-cnpg-io-v1-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 ghcr.io/cloudnative-pg/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。

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

  template:
    metadata:
      labels:
        app: pooler
    spec:
      containers: []
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - pooler
            topologyKey: "kubernetes.io/hostname"
```

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。

```yaml
apiVersion: postgresql.cnpg.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
```

35.7 高可用性 HA

Kubernetes の展開のため、単一のインスタンスまたは複数のポッドで実行するようにプーラーを構成できます。公開されたサービスにより、クライアントが PgBouncer を実行している利用可能なポッドにランダムに分散されるようになり、基になるサーバー `rw` サービスを使用する場合またはサーバー複数のレプリカで `ro` サービスを使用する場合への接続を管理および再利用します。

> **警告:** インフラストラクチャが複数のアベイラビリティーゾーンにまたがり、それらの間のレイテンシーが高い場合、ネットワークホップに注意してください。たとえば、ゾーン 2 で実行され、ゾーン 3 で実行されている PgBouncer に接続し、ゾーン 1 の PostgreSQL プライマリをポイントしている場合を検討します。

35.8 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
- client_idle_timeout
- client_login_timeout
- default_pool_size
- disable_pqexec
- idle_transaction_timeout
- ignore_startup_parameters `extra_float_digits,options` に追加します - CNP が必要
- 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 MonitoringConfiguration {postgresql-cnpg-io-v1-MonitoringConfiguration} 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_prepared_statements
- max_user_connections
- min_pool_size
- query_timeout
- query_wait_timeout
- reserve_pool_size
- reserve_pool_timeout
- 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
- stats_period
- tcp_keepalive
- tcp_keepcnt
- tcp_keepidle
- tcp_keepintvl
- tcp_user_timeout
- 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。

35.9 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、`cnpg_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 `cnpg_pgbouncer_lists`
- `SHOW POOLS` 接頭辞 `cnpg_pgbouncer_pools`
- `SHOW STATS` 接頭辞 `cnpg_pgbouncer_stats`

CloudNativePG インスタンスと同様に、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、cnpg_pgbouncer メトリックの出力を示しています。

```
#  HELP cnpg_pgbouncer_collection_duration_seconds Collection time duration in seconds

#  TYPE cnpg_pgbouncer_collection_duration_seconds gauge

cnpg_pgbouncer_collection_duration_seconds{collector="Collect.up"} 0.002443168

#  HELP cnpg_pgbouncer_collections_total Total number of times PostgreSQL was␣
↪accessed for metrics.

#  TYPE cnpg_pgbouncer_collections_total counter

cnpg_pgbouncer_collections_total 1

#  HELP cnpg_pgbouncer_last_collection_error 1 if the last collection ended with␣
↪error, 0 otherwise.

#  TYPE cnpg_pgbouncer_last_collection_error gauge

cnpg_pgbouncer_last_collection_error 0

#  HELP cnpg_pgbouncer_lists_databases Count of databases.

#  TYPE cnpg_pgbouncer_lists_databases gauge

cnpg_pgbouncer_lists_databases 1

#  HELP cnpg_pgbouncer_lists_dns_names Count of DNS names in the cache.

#  TYPE cnpg_pgbouncer_lists_dns_names gauge

cnpg_pgbouncer_lists_dns_names 0

#  HELP cnpg_pgbouncer_lists_dns_pending Not used.

#  TYPE cnpg_pgbouncer_lists_dns_pending gauge

cnpg_pgbouncer_lists_dns_pending 0

#  HELP cnpg_pgbouncer_lists_dns_queries Count of in-flight DNS queries.
```

(次のページに続く)

(前のページからの続き)

```
#  TYPE cnpg_pgbouncer_lists_dns_queries gauge
cnpg_pgbouncer_lists_dns_queries 0
#  HELP cnpg_pgbouncer_lists_dns_zones Count of DNS zones in the cache.
#  TYPE cnpg_pgbouncer_lists_dns_zones gauge
cnpg_pgbouncer_lists_dns_zones 0
#  HELP cnpg_pgbouncer_lists_free_clients Count of free clients.
#  TYPE cnpg_pgbouncer_lists_free_clients gauge
cnpg_pgbouncer_lists_free_clients 49
#  HELP cnpg_pgbouncer_lists_free_servers Count of free servers.
#  TYPE cnpg_pgbouncer_lists_free_servers gauge
cnpg_pgbouncer_lists_free_servers 0
#  HELP cnpg_pgbouncer_lists_login_clients Count of clients in login state.
#  TYPE cnpg_pgbouncer_lists_login_clients gauge
cnpg_pgbouncer_lists_login_clients 0
#  HELP cnpg_pgbouncer_lists_pools Count of pools.
#  TYPE cnpg_pgbouncer_lists_pools gauge
cnpg_pgbouncer_lists_pools 1
#  HELP cnpg_pgbouncer_lists_used_clients Count of used clients.
#  TYPE cnpg_pgbouncer_lists_used_clients gauge
cnpg_pgbouncer_lists_used_clients 1
#  HELP cnpg_pgbouncer_lists_used_servers Count of used servers.
#  TYPE cnpg_pgbouncer_lists_used_servers gauge
```

(次のページに続く)

(前のページからの続き)

```
cnpg_pgbouncer_lists_used_servers 0

#  HELP cnpg_pgbouncer_lists_users Count of users.

#  TYPE cnpg_pgbouncer_lists_users gauge

cnpg_pgbouncer_lists_users 2

#  HELP cnpg_pgbouncer_pools_cl_active Client connections that are linked to server␣
↪connection and can process queries.

#  TYPE cnpg_pgbouncer_pools_cl_active gauge

cnpg_pgbouncer_pools_cl_active{database="pgbouncer",user="pgbouncer"} 1

#  HELP cnpg_pgbouncer_pools_cl_cancel_req Client connections that have not forwarded␣
↪query cancellations to the server yet.

#  TYPE cnpg_pgbouncer_pools_cl_cancel_req gauge

cnpg_pgbouncer_pools_cl_cancel_req{database="pgbouncer",user="pgbouncer"} 0

#  HELP cnpg_pgbouncer_pools_cl_waiting Client connections that have sent queries but␣
↪have not yet got a server connection.

#  TYPE cnpg_pgbouncer_pools_cl_waiting gauge

cnpg_pgbouncer_pools_cl_waiting{database="pgbouncer",user="pgbouncer"} 0

#  HELP cnpg_pgbouncer_pools_maxwait How long the first (oldest) client in the queue␣
↪has waited, in seconds. If this starts increasing, then the current pool of servers␣
↪does not handle requests quickly enough. The reason may be either an overloaded␣
↪server or just too small of a pool_size setting.

#  TYPE cnpg_pgbouncer_pools_maxwait gauge

cnpg_pgbouncer_pools_maxwait{database="pgbouncer",user="pgbouncer"} 0

#  HELP cnpg_pgbouncer_pools_maxwait_us Microsecond part of the maximum waiting time.

#  TYPE cnpg_pgbouncer_pools_maxwait_us gauge

cnpg_pgbouncer_pools_maxwait_us{database="pgbouncer",user="pgbouncer"} 0

#  HELP cnpg_pgbouncer_pools_pool_mode The pooling mode in use. 1 for session, 2 for␣
```

(次のページに続く)

(前のページからの続き)

```
↪transaction, 3 for statement, -1 if unknown

#  TYPE cnpg_pgbouncer_pools_pool_mode gauge

cnpg_pgbouncer_pools_pool_mode{database="pgbouncer",user="pgbouncer"} 3

#  HELP cnpg_pgbouncer_pools_sv_active Server connections that are linked to a client.

#  TYPE cnpg_pgbouncer_pools_sv_active gauge

cnpg_pgbouncer_pools_sv_active{database="pgbouncer",user="pgbouncer"} 0

#  HELP cnpg_pgbouncer_pools_sv_idle Server connections that are unused and␣
↪immediately usable for client queries.

#  TYPE cnpg_pgbouncer_pools_sv_idle gauge

cnpg_pgbouncer_pools_sv_idle{database="pgbouncer",user="pgbouncer"} 0

#  HELP cnpg_pgbouncer_pools_sv_login Server connections currently in the process of␣
↪logging in.

#  TYPE cnpg_pgbouncer_pools_sv_login gauge

cnpg_pgbouncer_pools_sv_login{database="pgbouncer",user="pgbouncer"} 0

#  HELP cnpg_pgbouncer_pools_sv_tested Server connections that are currently running␣
↪either server_reset_query or server_check_query.

#  TYPE cnpg_pgbouncer_pools_sv_tested gauge

cnpg_pgbouncer_pools_sv_tested{database="pgbouncer",user="pgbouncer"} 0

#  HELP cnpg_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 cnpg_pgbouncer_pools_sv_used gauge

cnpg_pgbouncer_pools_sv_used{database="pgbouncer",user="pgbouncer"} 0

#  HELP cnpg_pgbouncer_stats_avg_query_count Average queries per second in last stat␣
↪period.

#  TYPE cnpg_pgbouncer_stats_avg_query_count gauge
```

(次のページに続く)

(前のページからの続き)

```
cnpg_pgbouncer_stats_avg_query_count{database="pgbouncer"} 1

#  HELP cnpg_pgbouncer_stats_avg_query_time Average query duration, in microseconds.

#  TYPE cnpg_pgbouncer_stats_avg_query_time gauge

cnpg_pgbouncer_stats_avg_query_time{database="pgbouncer"} 0

#  HELP cnpg_pgbouncer_stats_avg_recv Average received (from clients) bytes per␣
↪second.

#  TYPE cnpg_pgbouncer_stats_avg_recv gauge

cnpg_pgbouncer_stats_avg_recv{database="pgbouncer"} 0

#  HELP cnpg_pgbouncer_stats_avg_sent Average sent (to clients) bytes per second.

#  TYPE cnpg_pgbouncer_stats_avg_sent gauge

cnpg_pgbouncer_stats_avg_sent{database="pgbouncer"} 0

#  HELP cnpg_pgbouncer_stats_avg_wait_time Time spent by clients waiting for a server,
 in microseconds (average per second).

#  TYPE cnpg_pgbouncer_stats_avg_wait_time gauge

cnpg_pgbouncer_stats_avg_wait_time{database="pgbouncer"} 0

#  HELP cnpg_pgbouncer_stats_avg_xact_count Average transactions per second in last␣
↪stat period.

#  TYPE cnpg_pgbouncer_stats_avg_xact_count gauge

cnpg_pgbouncer_stats_avg_xact_count{database="pgbouncer"} 1

#  HELP cnpg_pgbouncer_stats_avg_xact_time Average transaction duration, in␣
↪microseconds.

#  TYPE cnpg_pgbouncer_stats_avg_xact_time gauge

cnpg_pgbouncer_stats_avg_xact_time{database="pgbouncer"} 0

#  HELP cnpg_pgbouncer_stats_total_query_count Total number of SQL queries pooled by␣
↪pgbouncer.
```

(次のページに続く)

(前のページからの続き)

```
#  TYPE cnpg_pgbouncer_stats_total_query_count gauge

cnpg_pgbouncer_stats_total_query_count{database="pgbouncer"} 3

#  HELP cnpg_pgbouncer_stats_total_query_time Total number of microseconds spent by␣
↪pgbouncer when actively connected to PostgreSQL, executing queries.

#  TYPE cnpg_pgbouncer_stats_total_query_time gauge

cnpg_pgbouncer_stats_total_query_time{database="pgbouncer"} 0

#  HELP cnpg_pgbouncer_stats_total_received Total volume in bytes of network traffic␣
↪received by pgbouncer.

#  TYPE cnpg_pgbouncer_stats_total_received gauge

cnpg_pgbouncer_stats_total_received{database="pgbouncer"} 0

#  HELP cnpg_pgbouncer_stats_total_sent Total volume in bytes of network traffic sent␣
↪by pgbouncer.

#  TYPE cnpg_pgbouncer_stats_total_sent gauge

cnpg_pgbouncer_stats_total_sent{database="pgbouncer"} 0

#  HELP cnpg_pgbouncer_stats_total_wait_time Time spent by clients waiting for a␣
↪server, in microseconds.

#  TYPE cnpg_pgbouncer_stats_total_wait_time gauge

cnpg_pgbouncer_stats_total_wait_time{database="pgbouncer"} 0

#  HELP cnpg_pgbouncer_stats_total_xact_count Total number of SQL transactions pooled␣
↪by pgbouncer.

#  TYPE cnpg_pgbouncer_stats_total_xact_count gauge

cnpg_pgbouncer_stats_total_xact_count{database="pgbouncer"} 3

#  HELP cnpg_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 cnpg_pgbouncer_stats_total_xact_time gauge
```

(次のページに続く)

(前のページからの続き)

```
cnpg_pgbouncer_stats_total_xact_time{database="pgbouncer"} 0
```

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。

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

35.10 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"
  }
}
```

35.11 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、 Backup {postgresql-cnpg-io-v1-Backup} を介して switchover コマンドを発行し、 *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。

35.12 制限事項

35.12.1 単一の PostgreSQL クラスター

プーラーの現在の実装は、特定の CloudNativePG クラスターサービス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。

35.12.2 制御された構成可能性

CloudNativePG は、PgBouncer レイヤーが PostgreSQL と通信するために使用されるいくつかの構成オプションを透過的に管理します。このようなオプションは外部から構成できず、TLS 証明書、認証設定、databases セクション、および users セクションが含まれます。また、単一の PostgreSQL クラスターの特定のユースケースを考慮して、採用される基準は、ユーザーが構成できるオプションを明示的にリストすることです。

注釈: 採用されたソリューションは、大部分のユースケースに対応する可能性があります。これにより、PgBouncer の別のオペレーターを将来的に実装して、より高度なカスタマイズされたシナリオでガンマを完了する余地が残されています。

第36章　レプリカクラスター

レプリカクラスターは、独立した CloudNativePG `Cluster` リソースであり、別の Postgres インスタンスからレプリカにあるという主な特徴があります、理想的には同じく CloudNativePG 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 クラスターにあります。レプリカクラスターはカスケードすることもでき、以下で説明するように、ソースからのデータのレプリケーションをオブジェクトストアにのみ依存できます。

次の図は、この機能の詳細を含む“Architecture”section から取得したもので、レプリカクラスターを使用して実装できるアーキテクチャの例を示しています。

36.1 基本的な概念

CloudNativePG は、PostgreSQL クラスターが既存のソースソースから作成され、**レプリカクラスター** 機能を介して同期が維持される場合でも、PostgreSQL レプリケーションフレームワークの基礎に依存します。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カクラスターです。

最初のステップは、レプリカクラスターをブートストラップし、使用可能な方法のいずれかを選択することです。

- ストリーミングレプリケーション `pg_basebackup` 経由
- ボリュームスナップショットからのリカバリー
- 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ー

  Bootstrap を参照してください。

`pg_basebackup` ストリーミングまたは `recovery` 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方法については、

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 の継続的リカバリーを介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。2 つのオプションがあります。

- レプリカクラスターとソース間でストリーミングレプリケーションを使用しますこれには、2 つのクラスター間のネットワーク接続が正しく設定されていることを確認するために、いくつかの管理およびセキュリティ関連の作業を行う必要があります。
- WAL アーカイブオブジェクトストア上を使用して、ソースからオブジェクトストアに定期的にシップされ、レプリカクラスタの `barman-cloud-wal-restore` によってプルされる WAL ファイルを取得します
- 2 つのいずれか

あなたがしなければならないのは、実際に外部クラスターを定義することだけです。

外部クラスターに barmanObjectStore セクションが含まれる場合

- WAL アーカイブを使用できるようになり、CloudNativePG は指定されたプライマリインスタンスに restore_command を自動的に設定します
- ボリュームスナップショットを利用できない場合、 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。

外部クラスターに connectionParameters セクションが含まれる場合

- pg_basebackup セクションを使用して、ストリーミングレプリケーションを介してレプリカクラスターをブートストラップできるようになります
- CloudNativePG は、指定されたプライマリインスタンスで primary_conninfo オプションを自動的に設定するため、WAL レシーバープロセスが開始され、ソースクラスターに接続してデータを受信します

作成された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でバックアップを実行でき、分散方法での対称アーキテクチャが有効になります。

以下を選択することにより、PostgreSQL データベースのお気に入りの分散アーキテクチャを決定する完全な柔軟性と自由があります。

- 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るプライベートクラウド
- さまざまなリージョンの複数の Kubernetes クラスターにまたがるパブリッククラウド
- 前の 2 つの混合ハイブリッド
- さまざまなリージョンおよびさまざまなクラウドサービスプロバイダーにある複数の Kubernetes クラスターにまたがるパブリッククラウド

36.2 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、クラスター YAML ファイルを作成し、それに応じて次の部分を定義する必要があります。

- レプリカクラスターに externalClusters セクションを定義します
- レプリカクラスターのブートストラップ部分を定義します。pg_basebackup セクションを使用してストリーミングを介してブートストラップするか、recovery セクションを使用して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からブートストラップできます。
- レプリカクラスターに継続的リカバリ部.spec.replica を定義します。行う必要があるのは、オプション.spec.replica.enabled でレプリカモードを有効にし、オプション.spec.replica.source で externalClusters 名前を設定することだけです

36.2.1 pg_basebackup を使用した例

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
```

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
```

36.2.2 オブジェクトストアからのバックアップを使用した例

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
```

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>
  barmanObjectStore:
    destinationPath: s3://backups/
    endpointURL: http://minio:9000
    s3Credentials:
      ...
  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。

36.2.3 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。

36.3 レプリカクラスターでの指定されたプライマリの昇格

指定されたプライマリ を **プライマリ** に昇格させるには、オプション `.spec.replica.enabled` を介してレプリカクラスターでレプリカモードを無効にするだけです

```
replica:
  enabled: false
  source: cluster-example
```

レプリカモードが無効になると、レプリカクラスターとソースクラスターは 2 つの別のクラスターになり、レプリカクラスターの**指定されたプライマリ** は、そのクラスターの **プライマリ**に昇格します。cnpg プラグインを使用して、以前にレプリカであった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 cnpg -n <cluster-name-space> status cluster-replica-example
```

注釈: レプリケーションの無効化は**不可逆**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。

第37章　Kubernetesのアップグレード

Kubernetes クラスターは常に更新する必要があります。これは、Kubernetes クラスター、特に**ベアメタル**を自己管理する場合、さらに重要になります。

定期的な更新を計画して実行することは、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取り出す制御されたダウンタイムの Kubernetes インフラストラクチャに導入されているにもかかわらず、組織が技術的負債をクリーンアップし、ビジネスリスクを削減する方法です。推奨読書
Embracing Risk

サイトリライアビリティーエンジニアリングの本より)。

たとえば、Kubernetes がインストールされている Linux サーバーにセキュリティ更新を適用したり、RAM、CPU、RAID コントローラなどの誤動作したハードウェアコンポーネントを交換したり、クラスターを最新バージョンの Kubernetes にアップグレードしたりする必要がある場合があります。

通常、クラスター内のメンテナンス操作は、次の方法で一度に 1 ノードずつ実行されます。

1. 更新するノードからのワークロードの削除 `drain`
2. 実際の操作の実行たとえば、システムのアップデート
3. ノードのクラスター `uncordon` への再参加

上記のプロセスでは、アップグレードの期間全体でワークロードを停止するか、別のノードに移行する必要があります。

最新のケースは、サービスの信頼性と Kubernetes の自己修復機能の観点から予想されるものですが、一時的に機能が低下したクラスターで動作し、アップグレードされたノードが再び起動するまで待機することをお勧めする状況が発生する場合があります。

特に、PostgreSQL クラスターが **ノードローカルストレージ**-つまり、PostgreSQL データベースが実行されている Kubernetes ワーカーノードのローカルな**ストレージに依存している場合** 。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上の共有ストレージにある場合、メンテナンスウィンドウを定義する必要はありません。現在ポッドが使用しているボリュームを、ドレイン後に別のノードで実行されているポッドで再利用できる場合、オペレーターのデフォルトの自己修復動作は正常に動作します。その場合、このセクションの残りの部分はスキップできます。

PostgreSQL のローカルストレージを使用する場合、たとえば、物理ノードまたはノード自分自身の更新。

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-local-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。

37.1 `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。

第38章　Postgresサービスの公開

このセクションでは、PostgreSQL サービスを外部に公開し、NGINX Ingress Controller を使用して **Kubernetes クラスターの外部から** PostgreSQL データベースへのアクセスを許可する方法について説明します。

> クイックスタート に従っている場合、今までに、 `default`

名前空間の `cluster-example-rw` プライマリおよび `cluster-example-r` 読み取り専用サービスを介してクラスター内にアクセスできるデータベースが完成しているはずです。どちらのサービスもポート `5432` を使用します。

ポート `5432` の外部アクセスからプライマリインスタンスにアクセスできるようにすることにします。Kubernetes インフラストラクチャに移行する場合の典型的なユースケースは、実際、簡単または持続的に「コンテナ化」できない**古いアプリケーション**に代表されるユースケースです。賢明な回避策は、仮想マシンまたは物理サーバーにある可能性が高いアプリケーションが、同じネットワーク内の Kubernetes クラスター内の PostgreSQL データベースにアクセスできるようにすることです。

> **警告:** 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ります。外部アクセスを許可する前にデータベースが保護されていること、または 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ます。

この例では、Kubernetes プロジェクトによって直接維持され、すべての Kubernetes クラスターで設定できるため、

> NGINX Ingress Controller を使用します。他の多くのコントローラが使用できます *Kubernetes documentation* を参照してください

包括的なリストについては。

次のことを前提としています。

- NGINX Ingress コントローラが展開され、正常に動作します
- クラスターに `LoadBalancer` タイプのサービスを作成することが可能です

重要: 入力は、HTTP および HTTPS トラフィックを公開する場合にのみ必要です。 NGINX Ingress コントローラはできますが、すべての Ingress オブジェクトが任意のポートまたはプロトコルを公開できるわけではありません。

最初のステップは、 `tcp-services ConfigMap` を作成することです。そのデータフィールドには、外部に公開されたポートと、内部でポイントする名前空間、サービス、ポートに関する情報が含まれています。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: tcp-services
  namespace: ingress-nginx
data:
  5432: default/cluster-example-rw:5432
```

次に、ドキュメントで提案されているとおりに NGINX Ingress Controller をインストールしている場合、ingress-nginx サービスが必要です。公開するには、5432 ポートを ingress-nginx サービスに追加する必要があります。イングレスは、ポート 5432 での着信接続をデータベースにリダイレクトします。

```
apiVersion: v1
kind: Service
metadata:
  name: ingress-nginx
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
spec:
  type: LoadBalancer
  ports:
    - name: http
      port: 80
      targetPort: 80
      protocol: TCP
    - name: https
      port: 443
      targetPort: 443
      protocol: TCP
    - name: postgres
      port: 5432
      targetPort: 5432
      protocol: TCP
  selector:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
```

使用できます

```
- --
apiVersion: v1
kind: ConfigMap
metadata:
  name: tcp-services
```

(次のページに続く)

(前のページからの続き)

```
  namespace: ingress-nginx
data:
  5432: default/cluster-example-rw:5432

- --
apiVersion: v1
kind: Service
metadata:
  name: ingress-nginx
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
spec:
  type: LoadBalancer
  ports:
    - name: http
      port: 80
      targetPort: 80
      protocol: TCP
    - name: https
      port: 443
      targetPort: 443
      protocol: TCP
    - name: postgres
      port: 5432
      targetPort: 5432
      protocol: TCP
  selector:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
```

kubectl を使用して適用します。

警告: このファイルを直接適用すると、Ingress の `ConfigMap` および `Service` の以前の変更が上書きされます

これで、Kubernetes クラスターの外部から PostgreSQL クラスターに到達できるようになります。

重要: Ingress からの接続を許可するように `pg_hba` を構成してください。

38.1 Minikube でのテスト

Minikube では、次を実行するイングレスコントローラをセットアップできます。

```
minikube addons enable ingress
```

次に、展開にパッチを適用して、ポート 5432 でのアクセスを許可できます。次の内容を含む patch.yaml という名前のファイルを作成します。

```
spec:
  template:
    spec:
      containers:
      - name: controller
        ports:
         - containerPort: 5432
           hostPort: 5432
```

そして、それを ingress-nginx-controller 展開に適用します。

```
kubectl patch deployment ingress-nginx-controller --patch "$(cat patch.yaml)" -n␣
↪ingress-nginx
```

次を実行しているマシンからプライマリにアクセスできます。

```
psql -h $(minikube ip) -p 5432 -U postgres
```

第39章　Kubectlプラグイン

CloudNativePG 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。

39.1 インストール

さまざまな方法を使用して cnpg 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。

39.1.1 インストールスクリプトを介して

```
curl -sSfL \
  https://github.com/cloudnative-pg/cloudnative-pg/raw/main/hack/install-cnpg-plugin.
↪sh | \
  sudo sh -s -- -b /usr/local/bin
```

39.1.2 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 CloudNativePG オペレーター
> と同じまたは新しいリリースを選択してくださいに移動できます。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。

39.1.3 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 1.18.1 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/cloudnative-pg/cloudnative-pg/releases/download/v1.18.1/
↪kubectl-cnpg_1.18.1_linux_x86_64.deb
```

次に、 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。

```
$ dpkg -i kubectl-cnpg_1.18.1_linux_x86_64.deb
(Reading database ... 16102 files and directories currently installed.)
Preparing to unpack kubectl-cnpg_1.18.1_linux_x86_64.deb ...
Unpacking cnpg (1.18.1) over (1.18.1) ...
Setting up cnpg (1.18.1) ...
```

39.1.4 RPM パッケージ

.deb パッケージの例と同様に、Intel 64 ビットマシンの 1.18.1 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/cloudnative-pg/cloudnative-pg/releases/download/v1.18.1/
↪kubectl-cnpg_1.18.1_linux_x86_64.rpm \
  --output kube-plugin.rpm
```

次に、 yum でインストールすると、使用する準備が整います。

```
$ yum --disablerepo=* localinstall kube-plugin.rpm
yum --disablerepo=* localinstall kube-plugin.rpm
Failed to set locale, defaulting to C.UTF-8
Dependencies resolved.
================================================================================
 Package            Architecture         Version                  Repository
         Size
================================================================================
Installing:
 cnpg               x86_64               1.18.1-1                 @commandline
         14 M

Transaction Summary
================================================================================
Install  1 Package

Total size: 14 M
Installed size: 43 M
Is this ok [y/N]: y
```

39.1.5 Krew を使用する

Krew を既にインストールしている場合は、単に実行できます。

```
kubectl krew install cnpg
```

プラグインの新しいバージョンがリリースされたら、次の方法で既存のインストールを更新できます。

```
kubectl krew update
kubectl krew upgrade cnpg
```

39.1.6 サポートされているアーキテクチャ

CloudNativePG 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

39.2 使用する

プラグインがインストールおよび展開されると、次のように使用を開始できます。

```
kubectl cnpg <command> <args...>
```

39.2.1 インストールマニフェストの生成

cnpg 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 cnpg install generate --help
```

主なオプションは次のとおりです。

- `-n` オペレーターをインストールする名前空間デフォルトでは `cnpg-system`
- `--replicas` 展開内のレプリカの数

- --version 1.17 など、インストールされるオペレーターのマイナーバージョン。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。
- --watch-namespace 監視する名前空間を含むコンマ区切りの文字列デフォルトではすべての名前空間

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 cnpg install generate \
  -n king \
  --version 1.17 \
  --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 CNPG オペレーターを king 名前空間にインストールします
- --version 1.17 マイナーバージョン 1.17 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

39.2.2 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 cnpg status sandbox
```

```
Cluster in healthy state
Name:               sandbox
Namespace:          default
System ID:          7039966298120953877
PostgreSQL Image:   ghcr.io/cloudnative-pg/postgresql:16.1
Primary instance:   sandbox-2
Instances:          3
Ready instances:    3
Current Write LSN:  3AF/EAFA6168 (Timeline: 8 - WAL File: 00000008000003AF00000075)

Continuous Backup status
First Point of Recoverability:  Not Available
Working WAL archiving:          OK
Last Archived WAL:              00000008000003AE00000079   @   2021-12-14T10:16:29.
↪340047Z
Last Failed WAL: -

Certificates Status
Certificate Name             Expiration Date                Days Left Until Expiration
- ---------------             ---------------                --------------------------
↪-
cluster-example-ca           2022-05-05 15:02:42 +0000 UTC  87.23
cluster-example-replication  2022-05-05 15:02:42 +0000 UTC  87.23
cluster-example-server       2022-05-05 15:02:42 +0000 UTC  87.23

Streaming Replication status
Name       Sent LSN      Write LSN     Flush LSN     Replay LSN    Write Lag       ␣
↪Flush Lag        Replay Lag       State      Sync State  Sync Priority
- ---       --------      ---------     ---------     ----------    ---------        -
↪--------        ----------       -----      ----------  -------------
sandbox-1  3AF/EB0524F0  3AF/EB011760  3AF/EAFEDE50  3AF/EAFEDE50  00:00:00.004461 ␣
↪00:00:00.007901  00:00:00.007901  streaming  quorum      1
sandbox-3  3AF/EB0524F0  3AF/EB030B00  3AF/EB030B00  3AF/EB011760  00:00:00.000977 ␣
↪00:00:00.004194  00:00:00.008252  streaming  quorum      1

Instances status
Name       Database Size  Current LSN   Replication role  Status  QoS         Manager␣
↪Version
- ---       -------------  -----------   ----------------  ------  ---         -------
↪--------
sandbox-1  302 GB         3AF/E9FFFFE0  Standby (sync)    OK      Guaranteed  1.11.0
sandbox-2  302 GB         3AF/EAFA6168  Primary           OK      Guaranteed  1.11.0
sandbox-3  302 GB         3AF/EBAD5D18  Standby (sync)    OK      Guaranteed  1.11.0
```

--verbose または-v だけを追加することにより、ステータスのより詳細なバージョンを取得することもできます。

```
kubectl cnpg status sandbox --verbose
```

```
Cluster in healthy state
Name:               sandbox
Namespace:          default
System ID:          7039966298120953877
PostgreSQL Image:   ghcr.io/cloudnative-pg/postgresql:16.1
Primary instance:   sandbox-2
Instances:          3
Ready instances:    3
Current Write LSN:  3B1/61DE3158 (Timeline: 8 - WAL File: 00000008000003B100000030)

PostgreSQL Configuration
archive_command = /controller/manager wal-archive --log-destination /controller/log/
↪postgres.json %p
archive_mode = on
archive_timeout = 5min
checkpoint_completion_target = 0.9
checkpoint_timeout = 900s
cluster_name = sandbox
dynamic_shared_memory_type = sysv
full_page_writes = on
hot_standby = true
jit = on
listen_addresses = *
log_autovacuum_min_duration = 1s
log_checkpoints = on
log_destination = csvlog
log_directory = /controller/log
log_filename = postgres
log_lock_waits = on
log_min_duration_statement = 1000
log_rotation_age = 0
log_rotation_size = 0
log_statement = ddl
log_temp_files = 1024
log_truncate_on_rotation = false
logging_collector = on
maintenance_work_mem = 2GB
max_connections = 1000
max_parallel_workers = 32
max_replication_slots = 32
max_wal_size = 15GB
```

(次のページに続く)

(前のページからの続き)

```
max_worker_processes = 32
pg_stat_statements.max = 10000
pg_stat_statements.track = all
port = 5432
shared_buffers = 16GB
shared_memory_type = sysv
shared_preload_libraries = pg_stat_statements
ssl = on
ssl_ca_file = /controller/certificates/client-ca.crt
ssl_cert_file = /controller/certificates/server.crt
ssl_key_file = /controller/certificates/server.key
synchronous_standby_names = ANY 1 ("sandbox-1","sandbox-3")
unix_socket_directories = /controller/run
wal_keep_size = 512MB
wal_level = logical
wal_log_hints = on
cnpg.config_sha256 = 3cfa683e23fe513afaee7c97b50ce0628e0cc634bca8b096517538a9a4428efc

PostgreSQL HBA Rules

#  Grant local access

local all all peer map=local

#  Require client certificate authentication for the streaming_replica user

hostssl postgres streaming_replica all cert
hostssl replication streaming_replica all cert
hostssl all cnpg_pooler_pgbouncer all cert

#  Otherwise use the default authentication method

host all all all scram-sha-256

Continuous Backup status
First Point of Recoverability:  Not Available
Working WAL archiving:          OK
Last Archived WAL:              000000080000003B00000001D   @   2021-12-14T10:20:42.
↪272815Z
Last Failed WAL: -

Streaming Replication status
Name       Sent LSN      Write LSN     Flush LSN     Replay LSN    Write Lag       ␣
↪Flush Lag        Replay Lag       State      Sync State  Sync Priority
- ---       --------      ---------     ---------     ----------    ---------        -
```

(次のページに続く)

(前のページからの続き)

```
↪--------        ----------       -----      ----------  -------------
sandbox-1  3B1/61E26448  3B1/61DF82F0  3B1/61DF82F0  3B1/61DF82F0  00:00:00.000333 ␣
↪00:00:00.000333  00:00:00.005484  streaming  quorum      1
sandbox-3  3B1/61E26448  3B1/61E26448  3B1/61DF82F0  3B1/61DF82F0  00:00:00.000756 ␣
↪00:00:00.000756  00:00:00.000756  streaming  quorum      1

Instances status
Name       Database Size  Current LSN   Replication role  Status  QoS         Manager␣
↪Version
- ---       -------------  -----------   ----------------  ------  ---         -------
↪--------
sandbox-1                 3B1/610204B8  Standby (sync)    OK      Guaranteed  1.11.0
sandbox-2                 3B1/61DE3158  Primary           OK      Guaranteed  1.11.0
sandbox-3                 3B1/62618470  Standby (sync)    OK      Guaranteed  1.11.0
```

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。

39.2.3 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 cnpg 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 cnpg promote cluster-example 2
```

39.2.4 証明書

CloudNativePG 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 cnpg certificate cluster-cert --cnpg-cluster cluster-example --cnpg-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) | .[]
```

39.2.5 再起動

kubectl cnpg 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 cnpg restart [clusterName]

#  this command will restart a single instance, according to the policy above

kubectl cnpg 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 `cnpg.io/reload` を持つラベルをそれに追加できます。

39.2.6 リロード

kubectl cnpg 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 cnpg reload [cluster_name]
```

39.2.7 メンテナンス

kubectl cnpg 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 cnpg 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
```

39.2.8 レポート

kubectl cnpg 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。

39.2.9 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。 CNPG 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 cnpg-system です

```
kubectl cnpg 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 cnpg 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
```

(次のページに続く)

(前のページからの続き)

```
  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/cnpg-ca-secret.yaml
  inflating: report_operator_<TIMESTAMP>/manifests/cnpg-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/cnpg-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 CloudNativePG Operator","version":"1.19.1","build":{"Version
↪":"1.19.0+dev107","Commit":"cc9bab17","Date":"2023-03-28"}}
2023-03-28T12:56:41.2518519009Z {"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 CloudNativePG Operator","version":"1.19.1","build":{"Version
↪":"1.19.0+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 cnpg-ca-secret.yaml
```

```
data:
  ca.crt: ""
```

(次のページに続く)

(前のページからの続き)

```
  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 cnpg 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 cnpg-ca-secret.yaml
```

```
data:
  ca.crt: LS0tLS1CRUdJTiBD…
  ca.key: LS0tLS1CRUdJTiBF…
metadata:
  creationTimestamp: "2022-03-22T10:42:28Z"
  managedFields:
  - apiVersion: v1
    fieldsType: FieldsV1
```

39.2.10 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 cnpg 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 cnpg 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 cnpg 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/
```

(次のページに続く)

(前のページからの続き)

```
  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
```

39.2.11 ログ

kubectl cnpg logs コマンドを使用すると、CloudNativePG 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 。

39.2.12 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 cnpg 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 cnpg 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 cnpg 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 cnpg 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 cnpg 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 cnpg 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-cnpg logs cluster cluster-example --output my-cluster.log

Successfully written logs to "my-cluster.log"
```

39.2.13 デストロイ

kubectl cnpg 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 cnpg.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 cnpg 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 cnpg destroy cluster-example 2
```

39.2.14 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 Cluster 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 cnpg hibernate [on|off] コマンドを介してのみ使用できます。

CloudNativePG クラスターを休止するとは、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 cnpg 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

ハイバーネーションしている場合、CloudNativePG クラスターは 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ハイバーネーションできません。

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 cnpg 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 cnpg 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 cnpg hibernate status <cluster-name>
```

39.2.15 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 cnpg 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。

39.2.16 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 cnpg 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。

39.2.17 新しい物理バックアップの要求

`kubectl cnpg 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 cnpg 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リリース 1.21 以降

```
kubectl cnpg 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 cnpg 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。

> 付録 *A*-バックアップ用の共通オブジェクトストア には、構成設定に関する詳細情報が含まれています。

39.2.18 psql の起動

kubectl cnpg 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 cnpg psql cluster-example

psql (16.1 (Debian 16.1-1.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 cnpg psql --replica cluster-example
psql (16.1 (Debian 16.1-1.pgdg110+1))

Type "help" for help.

postgres=# select pg_is_in_recovery();
 pg_is_in_recovery
- ------------------
 t
(1 row)

postgres=# \q
```

このコマンドは kubectl exec を起動します。正常に動作するには、PATH 変数で kubectl 実行可能ファイルに到達できる必要があります。

39.2.19 Postgres クラスターのスナップショット

警告: *kubectl cnpg snapshot* コマンドは削除されました。 付録 *A*-バックアップ用の共通オブジェクトストア 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。

39.2.20 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 CloudNativePG 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 は適切なソリューションを提供します。 `cnpg` 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 cnpg pgadmin4 cluster-example
```

このコマンドは次を生成します。

```
ConfigMap/cluster-example-pgadmin4 created
Deployment/cluster-example-pgadmin4 created
Service/cluster-example-pgadmin4 created
Secret/cluster-example-pgadmin4 created

[...]
```

pgAdmin を展開した後、kubectl を使用してポートを転送し、画面の指示に従ってブラウザーを介して接続します。

いつものように、`--dry-run` オプションを使用して YAML ファイルを生成できます。

```
kubectl cnpg 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

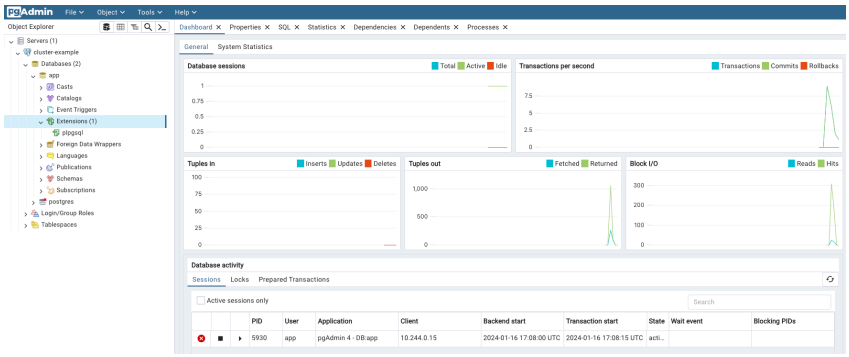


図 1: Screenshot of desktop installation of pgAdmin

に最適です。

```
kubectl cnpg 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 cnpg 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。

39.3 K9s との統合

cnpg プラグインは、Kubernetes クラスターと対話するための一般的な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。

第40章　自動フェイルオーバー

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。

40.1 RTO と RPO のインパクト

フェールオーバーにより、サービスが影響を受けたり、データが失われたりする可能性があります。

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 に悪影響を及ぼします。

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。

40.2 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。

第41章　障害対応

このページでは、Kubernetes クラスター展開で CloudNativePG をトラブルシューティングする方法に関する基本的な情報を見つけることができます。

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。

41.1 始める前に

41.1.1 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

41.1.2 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 Backup {postgresql-cnpg-io-v1-Backup}
- 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、好みの*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。

41.2 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。

41.3 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。

バックアップが設定されたクラスターでは、クラスターステータスにフィールド lastSuccessfulBackup および firstRecoverabilityPoint があります。最近の lastSuccessfulBackup があることを確認する必要があります。

.spec.backup スタンザが欠けているクラスターにはバックアップがありません。しつこいメッセージが PostgreSQL ログに表示されます。

```
Backup not configured, skip WAL archiving.
```

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。

41.4 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 cnpg プラグインに統合される可能性があります。

41.5 ログ

CloudNativePG によって作成および制御されるすべてのリソースは、Kubernetes が想定するとおりに標準出力にログし、JSON format に直接記録します。そのため、 `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する必要があります。

詳細については、次のように入力します。

```
kubectl logs --help
```

ヒント: JSON ログは、機械の読み取りには最適ですが、人間が読み取るのは困難です。 *jq* コマンドを使用して、使いやすさを向上させることをお勧めします。たとえば、 *| jq -C* を使用して `kubectl logs` コマンドを*パイプ*できます。

注釈: 以下のセクションでは、CloudNativePG のトラブルシューティングに関して、さまざまなリソースに関するログを取得する方法に関する例をいくつか示します。

41.6 オペレーター情報

デフォルトでは、CloudNativePG オペレーターは Kubernetes の `cnpg-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 cnpg-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、 *cnpg-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、 *cnpg-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 cnpg-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 cnpg-system <POD>
```

41.6.1 オペレーターに関する詳細情報を収集する

次のコマンドを実行して、CloudNativePG オペレーター展開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 cnpg-system \
  deployment/cnpg-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 cnpg-system \
  deployment/cnpg-controller-manager --all-containers=true | \
  jq -r . > cnpg_logs.json
```

kubectl-cnpg プラグインを使用して CloudNativePG オペレーターバージョンを取得します。

```
kubectl-cnpg status <CLUSTER>
```

出力

```
Cluster in healthy state
Name:               cluster-example
Namespace:          default
System ID:          7044925089871458324
PostgreSQL Image:   ghcr.io/cloudnative-pg/postgresql:16.1-3
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
```

(次のページに続く)

(前のページからの続き)

```
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
```

41.7 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、 cnpg プラグインが提供する status コマンドです。

```
kubectl cnpg 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。

注釈: クラスターのステータスを知ることに加えて、 cnpg プラグインを使用して次のこともできます。レプリカをプロモートします。
 証明書を管理します。
 ロールアウト・リスタートクラスターを作成して構成変更を適用します。
 リコンシリエーションを行う構成変更をリロードして適用するループ。
詳細については、 Backup {postgresql-cnpg-io-v1-Backup} ドキュメントを参照してください。

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:    ghcr.io/cloudnative-pg/postgresql:16.1-3
```

注釈: また、 *kubectl-cnpg status -n <NAMESPACE> <CLUSTER_NAME>* を使用して同じ情報を取得できます。

41.8 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。

```
kubectl get pod -l cnpg.io/cluster=<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、タイムスタンプを使いやすい形式で追加します。

```
kubectl logs -n <NAMESPACE> <CLUSTER>-<N> | \
  jq -r select(.logger=="postgres") | [(.ts|strflocaltime("%Y-%m-%dT%H:%M:%S %Z")), .
↪record.message] | @csv
```

41.8.1 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

```shell
2021-11-08 14:07:52.591 UTC [16] LOG:  redirecting log output to logging collector␣
↪process
2021-11-08 14:07:52.591 UTC [16] HINT:  Future log output will appear in directory "/
↪controller/log".
2021-11-08 14:07:52.591 UTC [16] LOG:  ending log output to stderr
2021-11-08 14:07:52.591 UTC [16] HINT:  Future log output will go to log destination
↪"csvlog".
```

値を含むフィールドでフィルタリングされた Pod ログを取得し、実行している | で区切って結合します。

```
kubectl logs -n <NAMESPACE> <CLUSTER>-<N> | \
  jq -r [.level, .ts, .logger, .msg] | join(" | ")
```

出力

```
info | 1636380469.5728037 | wal-archive | Backup not configured, skip WAL archiving
info | 1636383566.0664876 | postgres | record
```

41.9 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 cnpg.io/cluster=<CLUSTER>
```

重要: バックアップラベル付けは、CloudNativePG のバージョン 1.10.0 で導入されました。したがって、そのバージョン以降で作成されたリソースのみがこのようなラベルを含みます。

41.10 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。

41.11 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 cnpg.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。

41.12 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。

41.12.1 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。

edb_notranlate_40 以下は、失敗条件を含む `cluster.status` のスニペットです。

```
$ kubectl get cluster/<cluster-name> -o yaml
.
.
.
  status:
    conditions:
    - message: unexpected failure invoking barman-cloud-wal-archive: exit status
        2
      reason: ContinuousArchivingFailing
      status: "False"
      type: ContinuousArchiving

    - message: exit status 2
      reason: LastBackupFailed
      status: "False"
      type: LastBackupSucceeded

    - message: Cluster Is Not Ready
      reason: ClusterIsNotReady
      status: "False"
      type: Ready
```

41.13 ネットワーキング

CloudNativePG 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 をインストールする場合、ネットワークポリシー、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。これは、オペレーターとクラスターポッド間および/またはポッド間の必要な接続に影響を与える可能性があります。

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       cnpg.io/cluster=cluster-example   47m
default-deny-ingress   <none>                              57m
```

41.14 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 では、 cnpg.io/coredumpFilter アノテーションを介してコアダンプに含めるものを制御できます。

注釈: ラベルとアノテーション を参照してください。

CloudNativePG 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、cnpg.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。

41.15 よくある問題

41.15.1 ストレージがいっぱいです

クラスター内の 1 つ以上のポッドが CrashloopBackoff にあり、これがフルディスクが原因である可能性をログが示している場合、おそらくインスタンスの PersistentVolumeClaim のサイズを増やす必要があります。ドキュメントの ボリュームの拡張 を参照してください。

41.15.2 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
```

41.15.3 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
```

41.15.4 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

on connectivity

に関連していることがわかりました。リリース 1.20.1、1.19.3、および 1.18.5 から、ネットワークの問題は次のようにステータス列により明確に反映されます。

```
Instance Status Extraction Error: HTTP communication issue
```

41.15.5 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

networking section 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       cnpg.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

:ref:`networking page <ネットワーキング>` 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。

41.15.6 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: Not honoring huge_pages setting during initdb causes DB crash in Kubernetes 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

```yaml
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。

第42章　フェンシング

CloudNativePG 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発生しているように見える場合に、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。

42.1 インスタンスをフェンスする方法

CloudNativePG 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、`cnpg.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

- `cnpg.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
- `cnpg.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
- `cnpg.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 cnpg 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 cnpg fencing on cluster-example 1

#  to fence all the instances in a Cluster
```

(次のページに続く)

(前のページからの続き)

```
kubectl cnpg 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.cnpg.io/v1
kind: Cluster
metadata:
    annotations:
      cnpg.io/fencedInstances: ["cluster-example-1"]
[...]
```

42.2 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 cnpg 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 cnpg fencing off cluster-example 1

#  to lift the fencing for all the instances in a Cluster

kubectl cnpg fencing off cluster-example "*"
```

42.3 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 cnpg_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。

第43章　宣言ハイバーネーション

CloudNativePG は、PostgreSQL クラスターを起動、実行、いつでも利用できるように設計されています。

一部の種類のワークロードは、ワークロードがアクティブなときにのみデータベースを起動する必要があります。バッチ駆動のソリューションは、そのようなケースの 1 つです。

バッチ駆動ソリューションでは、バッチプロセスの実行中にのみデータベースを起動する必要があります。

宣言的ハイバーネーション機能を使用すると、データベース PVC を維持したままデータベースポッドを削除することにより、CPU パワーを節約できます。

注釈: 宣言的ハイバーネーションは、 imperative hibernation via the `cnpg plugin <Kubectl プラグイン>` の既存の実装とは異なります。必須ハイバーネーションは、高可用性クラスター内のすべての Postgres インスタンスをシャットダウンし、 *PGDATA* および WAL を含むプライマリの PVC の静的コピーを保持します。プラグインは、プライマリを再開し、すべてのレプリカが存在する場合を再作成することにより、ハイバーネーションフェーズを終了できます。

43.1 ハイバネーション

クラスターを休止状態にするには、 `cnpg.io/hibernation=on` アノテーションを設定します。

```
$ kubectl annotate cluster <cluster-name> --overwrite cnpg.io/hibernation=on
```

ハイバーネーション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状態ステータスは、 `cnpg.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==\"cnpg.
↪io/hibernation\")]}"

{
        "lastTransitionTime":"2023-03-05T16:43:35Z",
        "message":"Cluster has been hibernated",
        "reason":"Hibernated",
        "status":"True",
```

(次のページに続く)

(前のページからの続き)

```
        "type":"cnpg.io/hibernation"
}
```

ハイバーネーション状態ステータスは、kubectl の cnpg 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 cnpg status <cluster-name>
Cluster Summary
Name:               cluster-example
Namespace:          default
PostgreSQL Image:   ghcr.io/cloudnative-pg/postgresql:16.1
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
[..]
```

43.2 水分補給

クラスターをリハイドレートするには、 cnpg.io/hibernation アノテーションを off に設定します。

```
$ kubectl annotate cluster <cluster-name> --overwrite cnpg.io/hibernation=off
```

または、完全に設定を解除します。

```
$ kubectl annotate cluster <cluster-name> cnpg.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。

第44章　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 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*

44.1 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 は不変アプリケーションコンテナ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。

> コンテナイメージの要件 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* 。

44.2 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。

は

```
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: postgis-example
spec:
  instances: 3
  imageName: ghcr.io/cloudnative-pg/postgis:14
  bootstrap:
    initdb:
      postInitTemplateSQL:
        - CREATE EXTENSION postgis;
        - CREATE EXTENSION postgis_topology;
        - CREATE EXTENSION fuzzystrmatch;
        - CREATE EXTENSION postgis_tiger_geocoder;

  storage:
    size: 1Gi
```

以下は、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: CloudNativePG では構成より規約が適用されますが、本番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。

```
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: postgis-example
```

(次のページに続く)

(前のページからの続き)

```
spec:
  instances: 3
  imageName: ghcr.io/cloudnative-pg/postgis:14
  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 `.spec.bootstrap.initdb` 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 (16.1 (Debian 16.1-1.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␣
```

(次のページに続く)

(前のページからの続き)

```
↪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
- ------------------------+---------+------------+-------------------------------------
↪------------------------
 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)
```

第45章　エンドツーエンドテスト

CloudNativePG は、各コミットの後に、**エンドツーエンド E2E テスト**のスイートを介して自動的にテストされ、オペレーターが PostgreSQL クラスターを正しく展開および管理していることを保証します。

さらに、次の Kubernetes バージョンはコミットごとにテストされ、開発プロセスの初期段階でバグを検出するのに役立ちます。

- 1.27
- 1.26
- 1.25
- 1.24
- 1.23

次の PostgreSQL バージョンがテストされています。

- PostgreSQL 15
- PostgreSQL 14
- PostgreSQL 13
- PostgreSQL 12
- PostgreSQL 11

Kubernetes と PostgreSQL のテストされたバージョンごとに、
kind を使用して Kubernetes クラスターが作成され、そのクラスターで次の E2E テストスイートが実行されます。

- **基本 オペレーターのインストール。** クラスターの作成。*データストレージのための永続ボリュームの使用。
- **サービス接続 読み取り専用を含むサービスを介した接続。**ユーザー指定のサーバーおよび/またはクライアント証明書を介した接続。* PgBouncer;
- **自己修復 フェイルオーバー。** スイッチオーバー; *10* **秒以内にフェイルオーバーが発生した場合のプライマリエンドポイントスイッチ。**20 秒以内にスイッチオーバーの場合のプライマリエンドポイントスイッチ。*60* **秒未満で機能低下状態から回復します。** PVC の削除;
- **バックアップとリストア** *S3* **で** *Barman Cloud* **を使用したバックアップとスケジュールバックアップの実行。** Azure Blob ストレージでの Barman Cloud を使用したバックアップおよびスケジュールされたバックアップの実行。*S3* **で** *Barman Cloud* **を使用してバックアップから復元します。** Azure Blob ストレージで Barman Cloud を使用してバックアップから復元します。 * Wal-Restore;

- **Operator: オペレーター展開。**ConfigMap を介したオペレーター構成。 **オペレーターポッドの削除。**オペレーターポッドのエビクション。 **オペレーターのアップグレード**; オペレーターの高可用性。
- **可観測性 メトリック収集。** PgBouncer メトリック; * JSON ログ形式
- **レプリケーション 物理レプリカクラスター。**レプリケーションスロット; **同期レプリケーション。**クラスターのスケールアップおよびスケールダウン。
- **プラグイン** *CNPG* **プラグインを使用したクラスター休止状態。**フェンシング* 接続証明書の作成。
- **Postgres 構成** *PostgreSQL* **構成の変更を管理します。**PostgreSQL イメージを変更する際のローリング更新。
- **ポッドのスケジューリング 許容範囲とテイント。** `NodeSelector` を使用したポッドアフィニティ。
- **クラスターメタデータクラスターラベルとアノテーションの構成マップ。**オブジェクトのメタデータ。
- **リカバリー データの破損。** pg_basebackup;
- **データベースのインポート マイクロサービスアプローチ。**モノリスアプローチ。
- **ストレージ** *ストレージ拡張。
- **セキュリティ** * AppArmor アノテーションの伝播。Azure 環境でのみ実行されます。
- **メンテナンス** *ノードドレイン。

第46章　コンテナイメージの要件

Kubernetes の CloudNativePG オペレーターは、次の要件に準拠する PostgreSQL の互換性のあるコンテナイメージで動作するように設計されています。

- パスに存在する必要がある PostgreSQL 実行可能ファイル
  - `initdb`
  - `postgres`
  - `pg_ctl`
  - `pg_controldata`
  - `pg_basebackup`
- パスにある必要がある Barman クラウド実行可能ファイル
  - `barman-cloud-backup`
  - `barman-cloud-backup-delete`
  - `barman-cloud-backup-list`
  - `barman-cloud-check-wal-archive`
  - `barman-cloud-restore`
  - `barman-cloud-wal-archive`
  -
  -
  -
  -
  -
  -
  -
  -
  -
  -
  -

- PGAudit 拡張機能のインストールオプショナル-展開されたクラスターで PGAudit が必要な場合のみ
- 適切なロケール設定

重要: PostgreSQL versions supported by the PGDG のみが許可されます。

CloudNativePG はインスタンスマネージャーでオーバーライドするため、イメージ定義にエントリーポイントやコマンドは必要ありません。

> **警告:** アプリケーションコンテナイメージは、**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**でのみ CloudNativePG によって使用されます。

CloudNativePG コミュニティは、 public PostgreSQL container images を提供およびサポート

CloudNativePG で動作し、 ghcr.io で公開されます。

46.1 イメージタグの要件

イメージ名には Docker で有効な任意の名前を付けることができますが、CloudNativePG オペレーターは**イメージタグ**に依存して、イメージに含まれる Postgres メジャーバージョンを検出します。

イメージタグは、有効な PostgreSQL メジャーバージョン番号 14.5 または 15 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。

この後には、Docker タグで受け入れられる有効な文字の組み合わせが続き、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

- `11.1`
- `12.3.2.1-1`
- `12.4`
- `13`
- `14.5-10`
- `15.0`

> **警告:** *latest* は、画像の有効なタグとは見なされません。

第47章　オペレーター能力レベル

これらの機能は CloudNativePG 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。

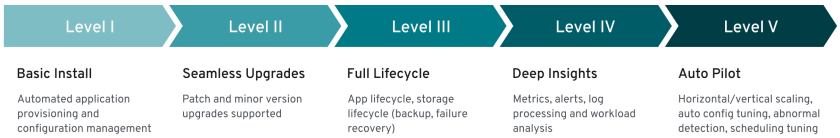


図 1: Operator Capability Levels

重要: *Operator Capability Levels model* に基づいて、CloudNativePG オペレーターからの「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。

47.1 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。

47.1.1 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。

47.1.2 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 。

47.1.3 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、ghcr.io で公開された最新の安定した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。

47.1.4 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 展開の構成を改善することです。

47.1.5 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。

47.1.6 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。

> cnp-bench オープンソースプロジェクトは、運用前にストレージとデータベースの両方のベンチマークを使用できます。

47.1.7 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 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ます。この実装は、以前に提案された PostgreSQL のパッチ
failover slots からインスピレーションを受けています。

47.1.8 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 クラスターを管理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、データベースへの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、データベース名とユーザー名の両方を指定できます。

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。

47.1.9 Postgres のロール、ユーザー、およびグループの構成

CloudNativePG 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。

47.1.10 ポッドセキュリティポリシー

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。

47.1.11 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

47.1.12 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。

47.1.13 コマンドラインインターフェイス

CloudNativePG 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。`cnpg` 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。

47.1.14 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。

47.1.15 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。

47.1.16 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 cnpg プラグインを使用して発行できます。

47.1.17 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。

47.1.18 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。

47.1.19 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。

47.1.20 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、既存の Postgres データベースを Kubernetes の新しい CloudNativePG クラスターに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。

47.1.21 PostGIS クラスター

CloudNativePG は、

PostGIS クラスター を使用したクラスターのインストールをサポートしています

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。

47.1.22 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、**シンプルバインド** 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。

47.1.23 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。CloudNativePG は、OperatorHub.io からの Helm チャートまたは OLM バンドルを介したインストールもサポートしています。

47.1.24 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。

47.2 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。

47.2.1 オペレーターのアップグレード

オペレーターを新しい展開としてシームレスにアップグレードできます。インスタンスマネージャーのインジェクションのため、演算子の変更はオペランドの変更を必要としません。オペレーターは、オペランドの古いバージョンを管理できます。

CloudNativePG は インスタンスマネージャーのインプレース更新 もサポート

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、クラスターのローリング更新およびその後のスイッチオーバーを必要としません。

47.2.2 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。オペレーターは、PostgreSQL のメジャーアップグレードを防止しながら、メジャーバージョン内のマイナー PostgreSQL リリースの点で両方向に行くことを可能にし、更新とロールバックを有効にします。

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 `cnpg` 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。

47.2.3 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`、および `Upgrading cluster` などを伝えます。

47.3 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 RPO < 5 分を達成することですが、長期的な目標は 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

47.3.1 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。

47.3.2 PostgreSQL のバックアップ

このオペレーターは、物理ベースのバックアップと継続的な WAL アーカイブに基づいた PostgreSQL のネイティブの継続的ホットバックアップテクノロジーを使用して、アプリケーションレベルのバックアップを提供するように設計されました。ベースバックアップは次の場所に保存できます。

- Kubernetes ボリュームのスナップショット
- 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

ベースバックアップは、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。

ベースバックアップは 2 つの方法で定義できます。

- オンデマンド、Backup カスタムリソース定義を介して
- cron のような構文を使用して、 ScheduledBackup カスタムリソース定義を介してスケジュール

ボリュームスナップショットは、この機能を基になるストレージクラスと CSI ドライバーに委任する Kubernetes API に直接依存します。ボリュームスナップショットバックアップは、非常に大規模なデータベース VLDB コンテキストに適しています。

オブジェクトストアのバックアップは、ジョブアプリケーションコンテナイメージの一部として配布の barman-cloud-backup に依存して、WAL ファイルと同じエンドポイントでバックアップを中継します。

barman-cloud-wal-restore と barman-cloud-backup は両方とも、GNU GPL 3 条件に基づいてアプリケーションコンテナイメージで配布されます。

オブジェクトストアのバックアップとボリュームスナップショットのバックアップは、PostgreSQL の稼働中に実行されますホットバックアップ。ボリュームスナップショットは、コールドバックアップでの一貫したデータベーススナップショットの取得もサポートしています。

47.3.3 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 RPO に影響を与えることなく、ベースバックアップのスタンバイへ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。

47.3.4 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またはオブジェクトストア内の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 は、アーカイブからの並列 WAL 取得をサポートしています。

47.3.5 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。

47.3.6 同期レプリケーションを介したデータ損失ゼロのクラスター

クォーラムベースの同期レプリケーションサポートを介して、ローカルの高可用性 CloudNativePG クラスターで**ゼロデータ損失** RPO=0 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最小および最大数を制御する 2 つの構成オプションを提供します。オペレーターは、クラスター内の使用可能な準備ができている PostgreSQL インスタンスの数に基づいて、それに応じて反応します。クォーラム `q` には次の式を使用します。

```
1 <= minSyncReplicas <= q <= maxSyncReplicas <= readyReplicas
```

47.3.7 レプリカクラスター

PostgreSQL の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を利用して、PostgreSQL クラスターのクロス Kubernetes クラスタートポロジを定義します。`replica` オプションを使用して、独立したクラスターをセットアップして、同じメジャーバージョンの別の PostgreSQL ソースからデータを継続的にレプリケートできます。このようなソースは、TLS を介した直接ストリーミング接続が 2 つのエンドポイントから許可されている限り、どこにでもあります。

さらに、ソースは Kubernetes の外部で、物理的または仮想環境で実行されている場合もあります。

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 形式のバックアップから、または `pg_basebackup` を使用したストリーミング方法で作成できます。WAL ファイルのシッピングと WAL ストリーミングの両方が許可されます。レプリカクラスターは、Kubernetes の PostgreSQL データベースのビジネス継続体制を劇的に向上させ、マルチプルのデータセンターにまたがり、ハイブリッドおよびマルチクラウドセットアップを受け入れます。現在、Kubernetes フェデレーションネイティブ機能を待機している間、データセンター間の手動スイッチオーバーが必要です。

47.3.8 テーブルスペースのサポート

CloudNativePG 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 は、テーブルスペースの透過的なオーケストレーションを通じて、PostgreSQL データベースのパフォーマンスとスケーラビリティを強化し、クラウドネイティブ環境で大規模なデータストレージを管理するための合理化され最適化されたエクスペリエンスを保証します。一時テーブルスペースのサポートも含まれています。

47.3.9 Liveness および readness プローブ

オペレーターは、kubelet によって呼び出される Postgres コンテナの liveness プローブと readness プローブを定義します。これらは、インスタンスマネージャーによって直接管理される Web サーバーの`/healthz` および`/readyz` エンドポイントにそれぞれマッピングされます。

liveness プローブは `pg_isready` 実行可能ファイルに基づいており、ポッドは終了コード 0 サーバーが通常に接続を受け入れるおよび 1 サーバーが起動中に接続を拒否している場合で正常であると考えられます。準備状況プローブは、単純なクエリー`;` を発行して、サーバーが接続を受け入れる準備ができていることを確認します。

47.3.10 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。

47.3.11 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。

47.3.12 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。

47.3.13 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。

47.3.14 ハイバネーション宣言的

CloudNativePG は *hibernation of a running PostgreSQL cluster* をサポート

`cnpg.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。

47.3.15 ハイバネーション (必須)

CloudNativePG は *hibernation of a running PostgreSQL cluster* をサポート

`cnpg` プラグインを介して。ハイバネーションは、高可用性クラスター内のすべての Postgres インスタンスをシャットダウンし、プライマリの PVC グループの静的コピーを保持します。コピーには、`PGDATA` および WAL が含まれています。プラグインを使用すると、プライマリを再開し、すべてのレプリカが存在する場合は再作成することにより、ハイバーネーションフェーズを終了できます。

47.3.16 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。

47.3.17 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。

47.3.18 PgBouncer を使用した接続プーリング

CloudNativePG 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

> PgBouncerIntegrationStatus {postgresql-cnpg-io-v1-PgBouncerIntegrationStatus}

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。

47.4 レベル 4 深い洞察

機能レベル 4 は、 **オブザーバビリティ**に関するモニタリング、アラート、トレンド、ログ処理です。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 は、柔軟なモニタリングとロギングのための業界標準のコミュニティで認められたツールと簡単に統合するために必要なすべてを提供するように設計されました。

47.4.1 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 は、コンテキストに統合して適応できる PostgreSQL の基本的なモニタリングクエリセットを提供します。

47.4.2 Grafana ダッシュボード

CloudNativePG には、PostgreSQL クラスターのすべての重要な側面を監視し、カスタマイズするためのベースとして使用できる Grafana ダッシュボードが付属しています。

47.4.3 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 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。

47.4.4 リアルタイムクエリモニタリング

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

- pg_failover_slots 物理的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

47.4.5 監査

CloudNativePG 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 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。

47.4.6 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。

47.5 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。

47.5.1 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。

47.5.2 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。

第48章　カスタムポッドコントローラ

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 は、 `StatefulSet` コントローラ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 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。

48.1 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 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。

48.2 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 CloudNativePG uses hot standby replicas のような構成パラメーターを増加する場合。

その際、CloudNativePG 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 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。

48.3 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 は、代わりに残りの PVC を使用不可として分類し、別のインスタンスがクラスターに正しく参加するように新しい PVC ペアの作成を開始します。

48.4 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 コントローラは、これらの戦略をすべて実装して、ユーザーがクラスターレベルで優先される動作を選択できるようにします。詳細については、
Kubernetes のアップグレード 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。

第49章　例

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。

49.1 基本

基本的なクラスター

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

  storage:
    size: 1Gi
```

クラスターの基本的な例。

カスタムクラスター

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

  # Parameters and pg_hba configuration will be append
  # to the default ones to make the cluster work
  postgresql:
```

(次のページに続く)

(前のページからの続き)

```
    parameters:
      max_worker_processes: "60"
    pg_hba:
      # To access through TCP/IP you will need to get username
      # and password from the secret cluster-example-custom-app
      - host all all all md5

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

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

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したクラスター

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

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

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

standard の指定されたストレージクラスを使用する基本クラスター。

永続的なボリューム要求 PVC テンプレートが構成されたクラスター

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

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

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

明示的な永続ボリューム要求テンプレートを使用した基本クラスター。

拡張構成例

```
#  Example of definition of a test cluster using all the elements available
#  in the CRD. Please change values appropriately for your environment.
#  Remember that you can take advantage of convention over configuration
#  and normally you dont need to use all these definitions.

apiVersion: v1
data:
  password:␣
↪VHhWZVE0bk44MlNTaVlIb3N3cU9VUlp2UURhTDRLcE5FbHNDRUVlOWJ3RHhNZDczS2NrSWVYelM1Y1U2TGlDMg==
  username: YXBw
kind: Secret
metadata:
  name: cluster-example-app-user
type: kubernetes.io/basic-auth
- --
apiVersion: v1
```

(次のページに続く)

(前のページからの続き)

```
data:
  password:␣
↪dU4zaTFIaDBiWWJDYzRUeVZBYWNCaG1TemdxdHpxeG1PVmpBbjBRSUNoc0pyU211OVBZMmZ3MnE4RUtLTHBaOQ==
  username: cG9zdGdyZXM=
kind: Secret
metadata:
  name: cluster-example-superuser
type: kubernetes.io/basic-auth
- --
apiVersion: v1
kind: Secret
metadata:
  name: backup-creds
data:
  ACCESS_KEY_ID: a2V5X2lk
  ACCESS_SECRET_KEY: c2VjcmV0X2tleQ==
- --
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: cluster-example-full
spec:
  description: "Example of cluster"
  imageName: ghcr.io/cloudnative-pg/postgresql:16.1
  # imagePullSecret is only required if the images are located in a private registry
  # imagePullSecrets:
  #   - name: private_registry_access
  instances: 3
  startDelay: 300
  stopDelay: 300
  primaryUpdateStrategy: unsupervised

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

  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: cluster-example-app-user
  # Alternative bootstrap method: start from a backup
  #recovery:
  #  backup:
  #    name: backup-example

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

storage:
  storageClass: standard
  size: 1Gi

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

resources:
  requests:
    memory: "512Mi"
    cpu: "1"
  limits:
    memory: "1Gi"
    cpu: "2"

affinity:
```

(次のページに続く)

(前のページからの続き)

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

  nodeMaintenanceWindow:
    inProgress: false
    reusePVC: false
```

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

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

  bootstrap:
    initdb:
      database: appdb
      owner: appuser
      postInitSQL:
        - create table numbers (i integer)
        - insert into numbers (select generate_series(1,10000))
      postInitTemplateSQL:
        - create extension intarray
```

(次のページに続く)

(前のページからの続き)

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

  storage:
    size: 1Gi
```

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セットを実行するクラスターの例。

カスタマイズされた ``pg_hba`` 構成を使用したサンプルクラスター

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

  storage:
    size: 1Gi
```

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 Secret と ConfigMap を含むサンプルクラスター

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

(次のページに続く)

(前のページからの続き)

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

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

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。

49.2 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。：

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

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

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

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

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**

```
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: pg-backup
spec:
```

(次のページに続く)

(前のページからの続き)

```
  instances: 3

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

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

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

適用可能で問題なし：

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

前のサンプルに対して実行されるバックアップの例。

バックアップが構成された単純なクラスター：**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。：

```
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
```

(次のページに続く)

(前のページからの続き)

```
metadata:
  name: cluster-example-with-backup
spec:
  instances: 3
  primaryUpdateStrategy: unsupervised

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

  # Backup properties
  # This assumes a local minio setup
  backup:
    barmanObjectStore:
      destinationPath: s3://backups/
      endpointURL: http://minio:9000
      s3Credentials:
        accessKeyId:
          name: minio
          key: ACCESS_KEY_ID
        secretAccessKey:
          name: minio
          key: ACCESS_SECRET_KEY
      wal:
        compression: gzip
```

バックアップが構成された基本的なクラスター。

49.3 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

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

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

(次のページに続く)

(前のページからの続き)

```
  #               the switchover of the primary
  primaryUpdateStrategy: unsupervised

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

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

適用され、正常、およびバックアップ

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

適用されて完了しました。:

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

  bootstrap:
    recovery:
      source: cluster-example-backup
```

(次のページに続く)

(前のページからの続き)

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

  storage:
    size: 1Gi

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

バックアップが構成されたクラスターの次の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

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

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

  # Backup properties
  backup:
    volumeSnapshot:
       className: csi-hostpath-snapclass
    barmanObjectStore:
```

(次のページに続く)

(前のページからの続き)

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

適用され、正常な、およびボリュームのスナップショット

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

適用されて完了しました。:

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

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

  bootstrap:
    recovery:
      source: cluster-example-with-volume-snapshot
      volumeSnapshots:
        storage:
          name: cluster-example-with-volume-snapshot-2-1692618163
```

(次のページに続く)

(前のページからの続き)

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

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

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

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

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

ボリュームスナップショットが構成されたクラスターの次の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*

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

(次のページに続く)

(前のページからの続き)

```
  storage:
    size: 1Gi
```

適用可能で問題なし：

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

  bootstrap:
    pg_basebackup:
      source: cluster-example

  replica:
    enabled: true
    source: cluster-example

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

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。

49.4 PostGIS

PostGIS の例

```
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: postgis-example
spec:
  instances: 3
  imageName: ghcr.io/cloudnative-pg/postgis:14
  bootstrap:
    initdb:
      postInitTemplateSQL:
        - CREATE EXTENSION postgis;
        - CREATE EXTENSION postgis_topology;
        - CREATE EXTENSION fuzzystrmatch;
        - CREATE EXTENSION postgis_tiger_geocoder;

  storage:
    size: 1Gi
```

PostGIS クラスターの例。詳細は、 *PostGIS クラスター* を参照してください。

49.5 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

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

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

(次のページに続く)

(前のページからの続き)

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

managed スタンザで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。

49.6 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

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

  storage:
    size: 1Gi

  tablespaces:
    - name: atablespace
      storage:
```

(次のページに続く)

(前のページからの続き)

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

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。：

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

  storage:
    size: 1Gi

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

  tablespaces:
    - name: atablespace
      storage:
        size: 1Gi
```

(次のページに続く)

(前のページからの続き)

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

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。：

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

  storage:
    size: 1Gi

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

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

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。

第50章　ネットワーキング

CloudNativePG 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ローカルの KinD または K3d クラスターに CloudNativePG を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、ネットワークの問題は発生しないはずです。

ただし、既存のインフラストラクチャに CloudNativePG を展開する場合、ネットワーク制限が適用される場合があり、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。

50.1 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 `cnpg-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。次の場所に例があります

```
#  This network policy allows ingress from the CNPG operator

#  installed in namespace cnpg-system into the pods corresponding

```

(次のページに続く)

(前のページからの続き)

```
#  to a cluster named cluster-example

apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-operator
spec:
  podSelector:
    matchLabels:
      cnpg.io/cluster: cluster-example # The label value must be the cluster name
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              kubernetes.io/metadata.name: cnpg-system # Namespace where the operator␣
↪is deployed
          podSelector:
            matchLabels:
              app.kubernetes.io/name: cloudnative-pg # Matches the Operator pod
      ports:
        - port: 8000
        - port: 5432
```

このリポジトリのファイル。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。

50.2 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* を参照します

セットアップ情報については。

第51章　ベンチマーク

CNPG kubectl プラグインは、CloudNativePG 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。

51.1 pgbench

`kubectl` CNPG 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 cnpg 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 cnpg pgbench \
  --job-name pgbench-init \
```

(次のページに続く)

(前のページからの続き)

```
  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 cnpg 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 cnpg pgbench \
  --db-name pgbench \
  --job-name pgbench-job \
  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 cnpg 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>
```

51.2 fio

kubectl CNPG 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 cnpg fio <fio-name>
``` カスタム値を使用した例

```shell
kubectl cnpg 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 cnpg 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 を作成します

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 <https://github.com/wallnerryan/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。

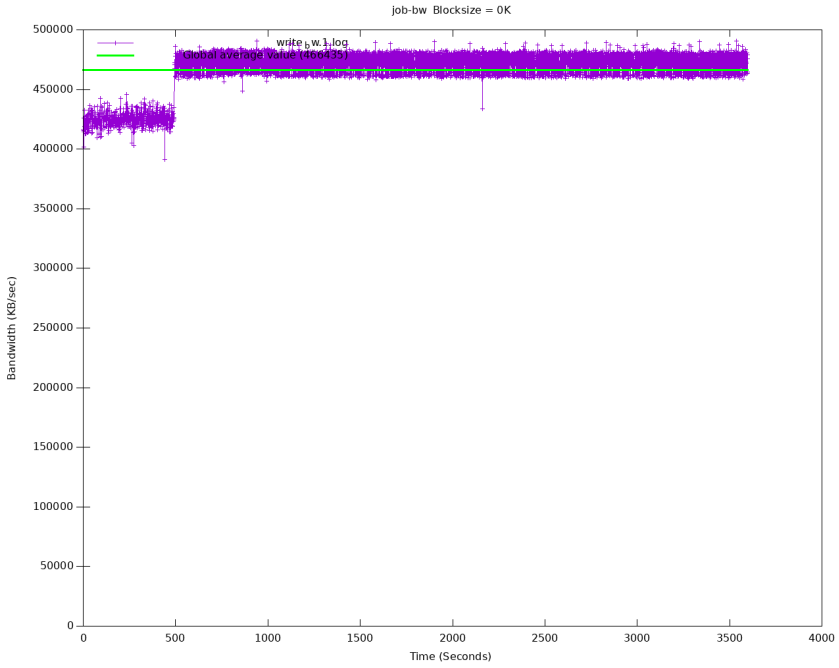


図 1: Sequential writes bandwidth

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。

`shell kubectl cnpg 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52章　コマーシャルサポート

CloudNativePG は、企業を支持しない独立したオープンソースプロジェクトです。

> Support

Web サイトには、CloudNativePG に関連する製品またはサービスを提供するサードパーティ企業および個人がリストされています。

CloudNativePG の商用サポートを提供している場合、自分自身または自分の組織をそのリストに追加します。

第53章　よくある質問FAQ

53.1 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 Kubernetes Community 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信じる理由です。

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。ただし、2ndQuadrantでCloud Native PostgreSQL / CloudNativePGオペレーターが開始されて以来、開発チームはクラウドネイ

ティブを 3 つの主要な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 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 の元となる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上のストレージと 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 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 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 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

edb_img_lk_1

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

一部のメンテナーを含むは、 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 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 はステートフルセットを使用しないのはなぜですか?

CloudNativePG 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 を直接管理します。この決定の背後にある理由と詳細については、

> カスタムポッドコントローラ セクションを参照してください。

53.2 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 オペレーターは、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 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理由は何ですか?

CloudNativePG 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 コントローラを直接拡張し、Kubernetes API サーバーに依存して Postgres クラスターのステータスを保持し、使用することにしました。以下の真実の唯一の情報源として

- 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。

API リファレンス

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

第54章　リソースタイプ

- 付録 *A*-バックアップ用の共通オブジェクトストア
- マルチアベイラビリティーゾーン *Kubernetes* クラスター
- プーラー {postgresql-cnpg-io-v1-Pooler}
- ScheduledBackup {postgresql-cnpg-io-v1-ScheduledBackup}

第55章　Backup

Backup はバックアップ API のスキーマです

Field	Description
apiVersion **[Required]** string	postgresql.cnpg.io/v1
kind **[Required]** string	バックアップ
metadata ** Required] [*meta/v1.ObjectMeta* **	説明はありません。メタデータ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 ** *archive_timeout* **	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 *archive_timeout*	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
	NA

第56章　クラスター

クラスターは PostgreSQL API のスキーマです

Field	Description
apiVersion **[Required]** string	postgresql.cnpg.io/v1
kind **[Required]** string	クラスター
metadata ** Required] [*meta/v1.ObjectMeta* **	説明はありません。メタデータ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] [*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
	NA

第57章　プーラー

Pooler は、プーラー API のスキーマです

Field	Description
apiVersion **[Required]** string	postgresql.cnpg.io/v1
kind **[Required]** string	プーラー
metadata ** Required] [*meta/v1.ObjectMeta* **	説明はありません。メタデータ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
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
	NA

第58章　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.cnpg.io/v1
kind **[Required]** string	スケジュールされたバックアップ
metadata ** Required] [*meta/v1.ObjectMeta* **	説明はありません。メタデータ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 ** *archive_timeout* **	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 *archive_timeout*	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
	NA

第59章　AffinityConfiguration

出演作品

- ClusterSpec {postgresql-cnpg-io-v1-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*	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
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。

次のページに続く

表 1 – 前のページからの続き

Field	Description
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。
	NA

第60章　AzureCredentials

出演作品

- BarmanCredentials {postgresql-cnpg-io-v1-BarmanCredentials}

AzureCredentials は、Azure Blob Storage にファイルをアップロードするために使用される資格情報のタイプです。接続文字列には、必要なすべての情報が含まれています。接続文字列が指定されない場合、ストレージアカウント名と、次のいずれか 1 つだけが必要です。

- storageKey
- storageSasToken
- manageFromAzureAD を true に設定することにより、ポッド環境から資格情報を継承する

Field	Description
connectionString *SecretKeySelector*	使用する接続文字列
storageAccount *SecretKeySelector*	データをアップロードするストレージアカウント
storageKey *SecretKeySelector*	ストレージアカウント名と組み合わせて使用されるストレージアカウントキー
storageSasToken *SecretKeySelector*	ストレージアカウント名と組み合わせて使用する共有アクセス署名
inheritFromAzureAD *bool*	キーを明示的に提供せずに Azure AD ベースの認証を使用します。
	NA

第61章　BackupConfiguration

出演作品

- ClusterSpec {postgresql-cnpg-io-v1-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 *VolumeSnapshot-Configuration*	VolumeSnapshot は、ボリュームスナップショットバックアップを実行するための構成を提供します。
barmanObjectStore *BarmanObject-StoreConfiguration*	barman-cloud ツールスイートの構成
retentionPolicy *string*	RetentionPolicy は、バックアップと 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。
target *archive_timeout*	バックアップを実行するインスタンスを決定するポリシー。使用可能なオプションは空の文字列で、デフォルトで prefer-standby ポリシー、プライマリインスタンスでバックアップを実行するプライマリ、使用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。
	NA

第62章　BackupMethod

`string` のエイリアス

出演作品

- BackupSpec {postgresql-cnpg-io-v1-BackupSpec}
- BackupStatus {postgresql-cnpg-io-v1-BackupStatus}
- ScheduledBackupSpec {postgresql-cnpg-io-v1-ScheduledBackupSpec}

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

第63章　BackupPhase

`string` のエイリアス

出演作品

- BackupStatus {postgresql-cnpg-io-v1-BackupStatus}

BackupPhase はバックアップのフェーズです

第64章　BackupSnapshotElementStatus

出演作品

- BackupSnapshotStatus {postgresql-cnpg-io-v1-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 **[Required]** *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
	NA

第65章　BackupSnapshotStatus

出演作品

- BackupStatus {postgresql-cnpg-io-v1-BackupStatus}

BackupSnapshotStatus volumeSnapshot 方法のバックアップ専用のフィールド

Field	Description
elements *archive_timeout*	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
	NA

第66章　BackupSource

出演作品

- BootstrapRecovery {postgresql-cnpg-io-v1-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 *LocalObjectReference*	LocalObjectReference のメンバーはこの型に埋め込まれます。説明は提供されていません。
endpointCA *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。
	NA

第67章　BackupSpec

出演作品

- 付録 *A*-バックアップ用の共通オブジェクトストア

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] [*LocalObjectReference* **	バックアップするクラスター
target *archive_timeout*	このバックアップを実行するインスタンスを決定するポリシー。空の場合、デフォルトの cluster.spec.backup.target。使用可能なオプションは、空の文字列、プライマリ、および優先スタンバイです。プライマリ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、prefer-standby は使用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 *archive_timeout*	使用するバックアップ方法、可能なオプションは barmanObjectStore および volumeSnapshot です。デフォルト barmanObjectStore。
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 かオフライン/コールド false かどうか クラスターフィールド '.spec.backup.volumeSnapshot.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
	NA

第68章　BackupStatus

出演作品

- 付録 *A*-バックアップ用の共通オブジェクトストア

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 *BarmanCredentials*	(BarmanCredentials のメンバーはこのタイプに埋め込まれています。) 各クラウドプロバイダーの潜在的な資格情報
endpointCA *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 *archive_timeout*	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

次のページに続く

表 1 – 前のページからの続き

Field	Description
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 *archive_timeout*	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 *archive_timeout*	使用されているバックアップ方法
online **[Required]** *bool*	バックアップがオンライン/ホット true かオフライン/コールド false か
	NA

第69章　BackupTarget

`string` のエイリアス

出演作品

- BackupConfiguration {postgresql-cnpg-io-v1-BackupConfiguration}
- BackupSpec {postgresql-cnpg-io-v1-BackupSpec}
- ScheduledBackupSpec {postgresql-cnpg-io-v1-ScheduledBackupSpec}

BackupTarget は、バックアップの優先ターゲットを説明します

第70章　BarmanCredentials

出演作品

- BackupStatus {postgresql-cnpg-io-v1-BackupStatus}
- BarmanObjectStoreConfiguration {postgresql-cnpg-io-v1-BarmanObjectStoreConfiguration}

BarmanCredentials 各クラウドプロバイダーの潜在的な資格情報を含むオブジェクト

Field	Description
googleCredentials *GoogleCredentials*	Google Cloud Storage にデータをアップロードするために使用する資格情報
s3Credentials *S3Credentials*	S3 にデータをアップロードするために使用する資格情報
azureCredentials *AzureCredentials*	Azure Blob Storage にデータをアップロードするために使用する資格情報
	NA

第71章　BarmanObjectStoreConfiguration

出演作品

- BackupConfiguration {postgresql-cnpg-io-v1-BackupConfiguration}
- :ref:``externalClusters` セクション <*externalClusters* セクション>`

BarmanObjectStoreConfiguration には、S3 互換オブジェクトストレージに対して Barman を使用したバックアップ構成が含まれます

Field	Description
BarmanCredentials　*BarmanCredentials*	(BarmanCredentials のメンバーはこのタイプに埋め込まれています。) 各クラウドプロバイダーの潜在的な資格情報
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
endpointCA *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
destinationPath **[Required]** *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
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
wal *archive_timeout*	WAL ストリームのバックアップの構成。定義されない場合、WAL ファイルは非圧縮で保存され、バケットのデフォルトポリシーに従って、オブジェクトストアで暗号化されない場合があります。
data *archive_timeout*	データファイルのバックアップに使用される構成 定義されていない場合、バケットのデフォルトポリシーに従って、ベースバックアップファイルは非圧縮で保存され、オブジェクトストアで暗号化されない場合があります。
tags *map[string]string*	Tags は、 Barman --tags オプションに渡されるキー値ペアのリストです。
historyTags *map[string]string*	HistoryTags は、Barman --history-tags オプションに渡されるキー値ペアのリストです。
	NA

第72章　BootstrapConfiguration

出演作品

- ClusterSpec {postgresql-cnpg-io-v1-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 *archive_timeout*	initdb を介してクラスターをブートストラップする
recovery *archive_timeout*	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 *archive_timeout*	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
	NA

第73章　BootstrapInitDB

出演作品

- BootstrapConfiguration {postgresql-cnpg-io-v1-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。データベースキーの値のデフォルト。
secret *LocalObjectReference*	ユーザーデータベースの所有者の初期資格情報を含むシークレットの名前。空の場合、新しいシークレットは最初から作成されます
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。
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
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
postInitApplicationSQL *[]string*	細心の注意を払って使用するように作成された直後に、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 クエリーのリストデフォルトでは空
postInitTemplateSQL *[]string*	細心の注意を払って使用するクラスターが作成された後、テンプレート 1 のスーパーユーザーとして実行される SQL クエリーのリストデフォルトでは空

次のページに続く

表 1 – 前のページからの続き

Field	Description
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 *archive_timeout*	PostInitApplicationSQLRefs は、SQL ファイルを含む ConfigMaps または Secrets への参照をポイントします。これらの参照への一般的な実装順序は、すべての Secret からすべての ConfigMap へであり、Secret または ConfigMap の内部では、実装順序は各配列デフォルトでは空の順序と同じです。
	NA

第74章　BootstrapPgBaseBackup

出演作品

- BootstrapConfiguration {postgresql-cnpg-io-v1-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。データベースキーの値のデフォルト。
secret *LocalObjectReference*	ユーザーデータベースの所有者の初期資格情報を含むシークレットの名前。空の場合、新しいシークレットは最初から作成されます
	NA

第75章　BootstrapRecovery

出演作品

- BootstrapConfiguration {postgresql-cnpg-io-v1-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、インスタンスが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。

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 *archive_timeout*	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。ソースおよびボリュームスナップショットと相互排他的。
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バックアップと相互排他的。
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 グループをマップし、CloudNativePG と互換性があり、フェンスで囲まれた 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た VolumeSnapshot および Persistent VolumeClaim リソースをサポートしていますこの制限は、将来オンラインバックアップが実装されるときに削除されます。バックアップと相互排他的。
recoveryTarget *archive_timeout*	デフォルトでは、復旧プロセスは、アーカイブ内の使用可能なすべての WAL ファイルを適用します完全復旧。ただし、一貫した状態に到達するとすぐにリカバリーを終了するか、PostgreSQL で予想される RecoveryTarget オブジェクトつまりタイムスタンプ、トランザクション ID、LSN、.. を指定することにより、ポイントインタイム PITR にリカバリーすることもできます。 .)。詳細 https://www.postgresql.org/docs/current/runtime-config-wal.html#RUNTIME-CONFIG-WAL-RECOVERY-TARGET
database *string*	アプリケーションが使用するデータベースの名前。デフォルト app.

次のページに続く

表 1 – 前のページからの続き

Field	Description
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。データベースキーの値のデフォルト。
secret *LocalObjectReference*	ユーザーデータベースの所有者の初期資格情報を含むシークレットの名前。空の場合、新しいシークレットは最初から作成されます
	NA

第76章　CertificatesConfiguration

出演作品

- CertificatesStatus {postgresql-cnpg-io-v1-CertificatesStatus}
- ClusterSpec {postgresql-cnpg-io-v1-ClusterSpec}

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*	サーバー 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。 - 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 サーバーを生成するために使用されるキー SSL 証明書、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。 - ca.crt クライアントの検証に使用する 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: キークライアント証明書を生成するために使用されます。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。
	NA

第77章　CertificatesStatus

出演作品

- ClusterStatus {postgresql-cnpg-io-v1-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 *archive_timeout*	(CertificatesConfiguration のメンバーはこのタイプに埋め込まれています。) 必要に応じてデフォルト値で初期化される、サーバー証明書を処理するために必要な構成。
expirations *map[string]string*	すべての証明書の有効期限。
	NA

第78章　ClusterSpec

出演作品

- マルチアベイラビリティーゾーン *Kubernetes* 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のためのダイジェスト<image>:<tag>@sha256:<digestValue>の両方をサポート
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 になります。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 26
postgresGID *int64*	画像内の postgres ユーザーの GID、デフォルト 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 *archive_timeout*	レプリケーションスロット管理構成
bootstrap *archive_timeout*	このクラスターをブートストラップする手順
replica *ReplicaClusterConfiguration*	レプリカクラスター構成
superuserSecret *LocalObjectReference*	スーパーユーザーのパスワードを含むシークレット。定義されていない場合、ランダムに生成されたパスワードで新しいシークレットが作成されます

次のページに続く

表 1 – 前のページからの続き

Field	Description
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、削除し、postgres ユーザーのパスワードを NULL に設定して空白にします。デフォルトでは無効になっています。
certificates *archive_timeout*	CA および関連する証明書の構成
imagePullSecrets *LocalObjectReference*	イメージのプルに使用されるプルシークレットのリスト
storage *archive_timeout*	インスタンスのストレージの構成
serviceAccountTemplate *ServiceAccountTemplate*	サービスアカウントの世代を構成する
walStorage *archive_timeout*	PostgreSQL WAL のストレージの構成書き込み先行ログ
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
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つまり stopDelay SmartShutdownTimeout を要求するための十分な時間を予約してください。
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
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*	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 ** Required] [*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

次のページに続く

表 1 – 前のページからの続き

Field	Description
priorityClassName *string*	生成されたすべてのポッドで使用される優先度クラスの名前。指定された PriorityClass が存在しない場合、ポッドはスケジュールできません。詳細については、https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption/#priorityclass を参照してください。
primaryUpdateStrategy *PrimaryUpdateStrategy*	すべてのレプリカが正常に更新された後、ローリング更新手順中にプライマリサーバーをアップグレードするために従う展開戦略。自動監視なしのデフォルトまたは手動監視ありが可能です。
primaryUpdateMethod *PrimaryUpdateMethod*	すべてのレプリカが正常に更新された後、ローリング更新手順中にプライマリサーバーをアップグレードするために従う方法。スイッチオーバー switchover またはインプレース restart デフォルトを使用できます。
backup *archive_timeout*	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
monitoring *archive_timeout*	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*	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*	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*	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*	テーブルスペース構成
	NA

第79章　ClusterStatus

出演作品

- マルチアベイラビリティーゾーン *Kubernetes* 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 *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 *archive_timeout*	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。
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
writeService *string*	現在の書き込みポッド

次のページに続く

表 1 – 前のページからの続き

Field	Description
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 *archive_timeout*	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 の内容から計算されます
firstRecoverabilityPointByMethod *meta/v1.Time*	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
lastSuccessfulBackup *string*	最後に成功したバックアップ、RFC3339 形式の日付として保存されますこのフィールドは、LastSuccessfulBackupByMethod の内容から計算されます
lastSuccessfulBackupByMethod meta/v1.Time*	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 形式で日付として保存
lastFailedBackup *string*	RFC3339 形式の日付として保存
cloudNativePGCommitHash *string*	このオペレーターが実行しているコミットハッシュ番号
currentPrimaryTimestamp *string*	最後にプライマリへの実際の昇格が発生したタイムスタンプ
currentPrimaryFailingSinceTimestamp *string*	プライマリが異常であることが検出されたときのタイムスタンプ このフィールドは、.spec.failoverDelay が設定されるとき、またはオンラインアップグレード中に報告されます
targetPrimaryTimestamp *string*	新しいプライマリの最後の要求が発生したタイムスタンプ
poolerIntegrations *PoolerIntegrations*	クラスターを参照するプーラーに必要な統合
cloudNativePGOperatorHash *string*	オペレーターのバイナリのハッシュ
conditions *meta/v1.Condition*	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
azurePVCUpdateEnabled *bool*	AzurePVCUpdateEnabled は、このクラスターで PVC 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
	NA

第80章　CompressionType

`string` のエイリアス

出演作品

- DataBackupConfiguration {postgresql-cnpg-io-v1-DataBackupConfiguration}
- WalBackupConfiguration {postgresql-cnpg-io-v1-WalBackupConfiguration}

CompressionType は、使用可能な圧縮の種類をカプセル化します

第81章　ConfigMapKeySelector

出演作品

- MonitoringConfiguration {postgresql-cnpg-io-v1-MonitoringConfiguration}
- PostInitApplicationSQLRefs {postgresql-cnpg-io-v1-PostInitApplicationSQLRefs}

ConfigMapKeySelector には、ConfigMap のキーを見つけられるように十分な情報が含まれています

Field		Description
LocalObjectReference	*LocalObjectReference*	LocalObjectReference のメンバーはこの型に埋め込まれています。 選択するポッドの名前空間のシークレットの名前。
key **[Required]** *string*		選択するキー
		NA

第82章　ConfigMapResourceVersion

出演作品

- ClusterStatus {postgresql-cnpg-io-v1-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
	NA

第83章　DataBackupConfiguration

出演作品

- BarmanObjectStoreConfiguration {postgresql-cnpg-io-v1-BarmanObjectStoreConfiguration}

DataBackupConfiguration は、データディレクトリのバックアップの構成です

Field	Description
compression *CompressionType*	バックアップファイルテーブルスペースごとの tar ファイルをオブジェクトストアにストリーミングしながら圧縮します。使用可能なオプションは、空の文字列圧縮なし、デフォルト、gzip、bzip2、または snappy です。
encryption *EncryptionType*	ファイルの暗号化を強制する場合バケットがそのようにまだ構成されていない場合。許可されるオプションは空の文字列バケットポリシーを使用しますデフォルト、AES256、および aws:kms です
jobs *int32*	バックアップのアップロードに使用される並列ジョブの数、デフォルト 2
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。
	NA

第84章　DataSource

出演作品

- BootstrapRecovery {postgresql-cnpg-io-v1-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 *core/v1.TypedLocalObjectReference*	PostgreSQL テーブルスペースのストレージの構成
	NA

第85章　DatabaseRoleRef

出演作品

- TablespaceConfiguration {postgresql-cnpg-io-v1-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。
	NA

第86章　EmbeddedObjectMetadata

出演作品

- ClusterSpec {postgresql-cnpg-io-v1-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。
	NA

第87章　EncryptionType

`string` のエイリアス

出演作品

- DataBackupConfiguration {postgresql-cnpg-io-v1-DataBackupConfiguration}
- WalBackupConfiguration {postgresql-cnpg-io-v1-WalBackupConfiguration}

EncryptionType は、使用可能な暗号化の種類をカプセル化しました

第88章　EnsureOption

`string` のエイリアス

出演作品

- RoleConfiguration {postgresql-cnpg-io-v1-RoleConfiguration}

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

第89章 Ephemeral VolumesSizeLimitConfiguration {#postgresql-cnpg-io-v1-EphemeralVolumesSizeLimitConfiguration}

出演作品

- ClusterSpec {postgresql-cnpg-io-v1-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 ** Required] [*k8s.io/apimachinery/pkg/api/resource.(**	Shm は、共有メモリボリュームのサイズ制限です
temporaryData ** Required] [*k8s.io/apimachinery/pkg/api/resource.(**	TemporaryData は、一時データボリュームのサイズ制限です
	NA

第90章　外部クラスター

出演作品

- ClusterSpec {postgresql-cnpg-io-v1-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 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 クラスターからリモート PostgreSQL サーバーへの接続を確立するときに、接続文字列で自動的に参照されます。これにより、外部クラスターの安全で効率的なパスワード管理が保証されます。
barmanObjectStore *BarmanObjectStoreConfiguration*	barman-cloud ツールスイートの構成
	NA

第91章　GoogleCredentials

出演作品

- BarmanCredentials {postgresql-cnpg-io-v1-BarmanCredentials}

GoogleCredentials は、Google Cloud Storage 資格情報のタイプです。これは、GKE 環境内で実行する場合でも指定する必要があります。

Field	Description
applicationCredentials　*SecretKeySelector*	資格情報を含む Google Cloud Storage JSON ファイルを含むシークレット
gkeEnvironment *bool*	true に設定されている場合、GKE 環境内で実行されていると想定されます。デフォルトは false です。
	NA

第92章　Import

出演作品

- BootstrapInitDB {postgresql-cnpg-io-v1-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。マイクロサービスまたはモノリスにすることができます。
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*	細心の注意を払って使用されるようにインポートされた直後に、アプリケーションデータベースでスーパーユーザーとして実行される SQL クエリのリストデフォルトでは空。マイクロサービスタイプでのみ使用できます。
schemaOnly *bool*	true に設定すると、 pg_restore の pre-data および post-data セクションのみが呼び出され、データのインポートが回避されます。デフォルト false。
	NA

第93章　ImportSource

出演作品

- Import {postgresql-cnpg-io-v1-Import}

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
	NA

第94章　インスタンスID

出演作品

- BackupStatus {postgresql-cnpg-io-v1-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
	NA

第95章　InstanceReportedState

出演作品

- ClusterStatus {postgresql-cnpg-io-v1-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
	NA

第96章　LDAPBindAsAuth

出演作品

- LDAPConfig {postgresql-cnpg-io-v1-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
	NA

第97章　LDAPBindSearchAuth

出演作品

- LDAPConfig {postgresql-cnpg-io-v1-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
	NA

第98章　LDAPConfig

出演作品

- PostgresConfiguration {postgresql-cnpg-io-v1-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、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
	NA

第99章　LDAPScheme

`string` のエイリアス

出演作品

- LDAPConfig {postgresql-cnpg-io-v1-LDAPConfig}

LDAPScheme は、 LDAP の使用可能なスキームを定義します

第100章　LocalObjectReference

出演作品

- BackupSource {postgresql-cnpg-io-v1-BackupSource}
- BackupSpec {postgresql-cnpg-io-v1-BackupSpec}
- BootstrapInitDB {postgresql-cnpg-io-v1-BootstrapInitDB}
- BootstrapPgBaseBackup {postgresql-cnpg-io-v1-BootstrapPgBaseBackup}
- BootstrapRecovery {postgresql-cnpg-io-v1-BootstrapRecovery}
- ClusterSpec {postgresql-cnpg-io-v1-ClusterSpec}
- ConfigMapKeySelector {postgresql-cnpg-io-v1-ConfigMapKeySelector}
- PgBouncerSpec {postgresql-cnpg-io-v1-PgBouncerSpec}
- PoolerSpec {postgresql-cnpg-io-v1-PoolerSpec}
- RoleConfiguration {postgresql-cnpg-io-v1-RoleConfiguration}
- ScheduledBackupSpec {postgresql-cnpg-io-v1-ScheduledBackupSpec}
- SecretKeySelector {postgresql-cnpg-io-v1-SecretKeySelector}

LocalObjectReference には、同じ名前空間内の既知のタイプを持つローカルオブジェクトを見つけるための十分な情報が含まれています

Field	Description
name **[Required]** *string*	参照先の名前。
	NA

第101章　ManchedConfiguration

出演作品

- ClusterSpec {postgresql-cnpg-io-v1-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*	クラスターによって管理されるデータベースロール
	NA

第102章　ManchedRoles

出演作品

- ClusterStatus {postgresql-cnpg-io-v1-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 *PasswordState*	PasswordStatus は、各管理対象ロールの最後のトランザクション ID とパスワードのシークレットバージョンを提供します
	NA

第103章　メタデータ

出演作品

- PodTemplateSpec {postgresql-cnpg-io-v1-PodTemplateSpec}
- ServiceAccountTemplate {postgresql-cnpg-io-v1-ServiceAccountTemplate}

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
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
	NA

第104章　MonitoringConfiguration

出演作品

- ClusterSpec {postgresql-cnpg-io-v1-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。
customQueriesConfigMap *ConfigMapKeySelector*	カスタムクエリを含む構成マップのリスト
customQueriesSecret *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する
podMonitorMetricRelabelings *archive_timeout*	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 *archive_timeout*	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。
	NA

第105章　NodeMaintenanceWindow

出演作品

- ClusterSpec {postgresql-cnpg-io-v1-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どうかインスタンスが 1 以上の場合、他の場所で再作成します
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?
	NA

第106章　OnlineConfiguration

出演作品

- BackupSpec {postgresql-cnpg-io-v1-BackupSpec}
- ScheduledBackupSpec {postgresql-cnpg-io-v1-ScheduledBackupSpec}
- VolumeSnapshotConfiguration {postgresql-cnpg-io-v1- VolumeSnapshotConfiguration}

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。
	NA

第107章　PasswordState

出演作品

- ManchedRoles {postgresql-cnpg-io-v1-ManadedRoles}

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
resourceVersion *string*	パスワードシークレットのリソースバージョン
	NA

第108章　PgBouncerIntegrationStatus

出演作品

- PoolerIntegrations {postgresql-cnpg-io-v1-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。
	NA

第109章　PgBouncerPoolMode

`string` のエイリアス

出演作品

- PgBouncerSpec {postgresql-cnpg-io-v1-PgBouncerSpec}

PgBouncerPoolMode は PgBouncer のモードです

第110章　PgBouncerSecrets

出演作品

- PoolerSecrets {postgresql-cnpg-io-v1-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
	NA

第111章　PgBouncerSpec

出演作品

- PoolerSpec {postgresql-cnpg-io-v1-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。デフォルトセッション。
authQuerySecret *LocalObjectReference*	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_shadow WHERE usename=$1" を指定する必要があり、自動 CNPG 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 user_search($1)」。指定される場合、AuthQuerySecret も指定する必要があり、自動 CNPG 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については、CNPG ドキュメントで構成可能なオプションのリス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 に設定されている場合、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。
	NA

第112章　PodTemplateSpec

出演作品

- PoolerSpec {postgresql-cnpg-io-v1-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
	NA

第113章　PodTopologyLabels

`map[string]string` のエイリアス

出演作品

- PodTopologyLabels {postgresql-cnpg-io-v1-PodTopologyLabels}

PodTopologyLabels は、ポッドのトポロジを表します。map[labelName]labelValue

第114章　PoolerIntegrations

出演作品

- ClusterStatus {postgresql-cnpg-io-v1-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。
	NA

第115章　PoolerMonitoringConfiguration

出演作品

- PoolerSpec {postgresql-cnpg-io-v1-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する
podMonitorMetricRelabelings *archive_timeout*	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 *archive_timeout*	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。
	NA

第116章　PoolerSecrets

出演作品

- PoolerStatus {postgresql-cnpg-io-v1-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 *PgBouncer-Secrets*	PgBouncer が使用するシークレットのバージョン
	NA

第117章　PoolerSpec

出演作品

- プーラー {postgresql-cnpg-io-v1-Pooler}

PoolerSpec は、プーラーの目的の状態を定義します

Field	Description
cluster ** Required] [*LocalObjectReference* **	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 *archive_timeout*	このプーラーの監視インフラストラクチャの構成。
	NA

第118章　PoolerStatus

出演作品

- プーラー {postgresql-cnpg-io-v1-Pooler}

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
	NA

第119章　PoolerType

`string` のエイリアス

出演作品

- PoolerSpec {postgresql-cnpg-io-v1-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。

第120章　PostInitApplicationSQLRefs

出演作品

- BootstrapInitDB {postgresql-cnpg-io-v1-BootstrapInitDB}

PostInitApplicationSQLRefs は、SQL ファイルを含む ConfigMaps または Secrets への参照をポイントします。これらの参照への一般的な実装順序は、すべての Secret からすべての ConfigMap へであり、Secret または ConfigMap の内部では、実装順序は各配列の順序と同じです。

Field	Description
secretRefs *SecretKeySelector*	SecretRefs は、シークレットへの参照のリストを保持します
configMapRefs *ConfigMapKeySelector*	ConfigMapRefs は、ConfigMaps への参照のリストを保持します
	NA

第121章　PostgresConfiguration

出演作品

- ClusterSpec {postgresql-cnpg-io-v1-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
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
syncReplicaElectionConstraint *SyncReplicaElectionConstraints*	同期レプリカが満たす要件。これは、"synchronous_standby_names"パラメーターの設定方法に影響します。
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 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。
	NA

第122章　PrimaryUpdateMethod

`string` のエイリアス

出演作品

- ClusterSpec {postgresql-cnpg-io-v1-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

第123章　PrimaryUpdateStrategy

`string` のエイリアス

出演作品

- ClusterSpec {postgresql-cnpg-io-v1-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

第124章　RecoveryTarget

出演作品

- BootstrapRecovery {postgresql-cnpg-io-v1-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
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
	NA

第125章　ReplicaClusterConfiguration

出演作品

- ClusterSpec {postgresql-cnpg-io-v1-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
source **[Required]** *string*	レプリケーション元の外部クラスターの名前
enabled **[Required]** *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。
	NA

第126章　ReplicationSlotsConfiguration

出演作品

- ClusterSpec {postgresql-cnpg-io-v1-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 *archive_timeout*	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 秒デフォルト 30 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。
	NA

第127章　ReplicationSlotsHAConfiguration

出演作品

- ReplicationSlotsConfiguration {postgresql-cnpg-io-v1-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_cnpg_に設定されます。
	NA

第128章　RoleConfiguration

出演作品

- ManchedConfiguration {postgresql-cnpg-io-v1-ManchedConfiguration}

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*	ロールが存在するかどうかを確認しますデフォルトで「present」
passwordSecret *LocalObjectReference*	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、データベースのスーパーユーザーステータス内のすべてのアクセス制限をオーバーライドできるスーパーユーザーであるかどうかは、危険であるため、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 に設定されている場合、定義されるロールは新しいデータベースを作成できます。 false デフォルトを指定すると、ロールがデータベースを作成する機能が拒否されます。

次のページに続く

表 1 – 前のページからの続き

Field	Description
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ることができます。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。
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。レプリケーション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。
	NA

第129章　S3Credentials

出演作品

- BarmanCredentials {postgresql-cnpg-io-v1-BarmanCredentials}

S3Credentials は、S3 にファイルをアップロードするために使用される資格情報のタイプです。これは、2 つの代替方法で提供できます。

- accessKeyId と secretAccessKey を明示的に渡す
- manageFromIAMRole を true に設定することにより、ポッド環境からロールを継承する

Field	Description
accessKeyId *SecretKeySelector*	アクセスキー ID への参照
secretAccessKey *SecretKeySelector*	シークレットアクセスキーへの参照
region *SecretKeySelector*	リージョン名を含むシークレットへの参照
sessionToken *SecretKeySelector*	セッションキーへの参照
inheritFromIAMRole *bool*	キーを明示的に提供せずにロールベースの認証を使用します。
	NA

第130章　ScheduledBackupSpec

出演作品

- ScheduledBackup {postgresql-cnpg-io-v1-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] [*LocalObjectReference* **	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の中にどの 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リファレンスなしフィールドが導入される前と同じ動作。 - self スケジュールされたバックアップオブジェクトを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 *archive_timeout*	このバックアップを実行するインスタンスを決定するポリシー。空の場合、デフォルトの cluster.spec.backup.target。使用可能なオプションは、空の文字列、プライマリ、および優先スタンバイです。プライマリ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、prefer-standby は使用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 *archive_timeout*	使用するバックアップ方法、可能なオプションは barmanObjectStore および volumeSnapshot です。デフォルト barmanObjectStore。
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 かオフライン/コールド false かどうか クラスターフィールド '.spec.backup.volumeSnapshot.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

次のページに続く

表 1 – 前のページからの続き

Field	Description
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
	NA

第131章　ScheduledBackupStatus

出演作品

- ScheduledBackup {postgresql-cnpg-io-v1-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
	NA

第132章　SecretKeySelector

出演作品

- AzureCredentials {postgresql-cnpg-io-v1-AzureCredentials}
- BackupSource {postgresql-cnpg-io-v1-BackupSource}
- BackupStatus {postgresql-cnpg-io-v1-BackupStatus}
- BarmanObjectStoreConfiguration {postgresql-cnpg-io-v1-BarmanObjectStoreConfiguration}
- GoogleCredentials {postgresql-cnpg-io-v1-GoogleCredentials}
- MonitoringConfiguration {postgresql-cnpg-io-v1-MonitoringConfiguration}
- PostInitApplicationSQLRefs {postgresql-cnpg-io-v1-PostInitApplicationSQLRefs}
- S3Credentials {postgresql-cnpg-io-v1-S3Credentials}

SecretKeySelector には、シークレットのキーを見つけるための十分な情報が含まれています

Field	Description
LocalObjectReference *LocalObjectReference*	LocalObjectReference のメンバーはこの型に埋め込まれています。 選択するポッドの名前空間のシークレットの名前。
key **[Required]** *string*	選択するキー
	NA

第133章　SecretVersion

出演作品

- PgBouncerSecrets {postgresql-cnpg-io-v1-PgBouncerSecrets}
- PoolerSecrets {postgresql-cnpg-io-v1-PoolerSecrets}

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
	NA

第134章　SecretsResourceVersion

出演作品

- ClusterStatus {postgresql-cnpg-io-v1-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
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
	NA

第135章　ServiceAccountTemplate

出演作品

- ClusterSpec {postgresql-cnpg-io-v1-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
	NA

第136章　SnapshotOwnerReference

`string` のエイリアス

出演作品

- VolumeSnapshotConfiguration {postgresql-cnpg-io-v1- 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。

第137章　SnapshotType

`string` のエイリアス

出演作品

- Import {postgresql-cnpg-io-v1-Import}

SnapshotType は許可されるインポートのタイプです

第138章　StorageConfiguration

出演作品

- ClusterSpec {postgresql-cnpg-io-v1-ClusterSpec}
- TablespaceConfiguration {postgresql-cnpg-io-v1-TablespaceConfiguration}

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
	NA

第139章　SyncReplicaElectionConstraints

出演作品

- PostgresConfiguration {postgresql-cnpg-io-v1-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
	NA

第140章　TablespaceConfiguration

出演作品

- ClusterSpec {postgresql-cnpg-io-v1-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 ** *archive_timeout* **	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 に設定されている場合、テーブルスペースは 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。
	NA

第141章　TablespaceState

出演作品

- ClusterStatus {postgresql-cnpg-io-v1-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
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 ** *archive_timeout* **	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
	NA

第142章　TablespaceStatus

`string` のエイリアス

出演作品

- TablespaceState {postgresql-cnpg-io-v1-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

第143章　トポロジ

出演作品

- ClusterStatus {postgresql-cnpg-io-v1-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 *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
	NA

第144章　VolumeSnapshotConfiguration {#postgresql-cnpg-io-v1-VolumeSnapshotConfiguration}

出演作品

- BackupConfiguration {postgresql-cnpg-io-v1-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種類がオンライン/ホット true、default かオフライン/コールド false かどうか
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
	NA

第145章　WalBackupConfiguration

出演作品

- BarmanObjectStoreConfiguration {postgresql-cnpg-io-v1-BarmanObjectStoreConfiguration}

WalBackupConfiguration は、WAL ストリームのバックアップの構成です

Field	Description
compression *CompressionType*	WAL ファイルをオブジェクトストアに送信する前に圧縮します。使用可能なオプションは、空の文字列圧縮なし、デフォルト、gzip、bzip2、または snappy です。
encryption *EncryptionType*	ファイルの暗号化を強制する場合バケットがそのようにまだ構成されていない場合。許可されるオプションは空の文字列バケットポリシーを使用しますデフォルト、AES256、および aws:kms です
maxParallel *int*	並列アーカイブする PostgreSQL インスタンスがバックアップオブジェクトストアにアーカイブしている場合、または並列リストアする WAL ファイルの数 PostgreSQL スタンバイがリカバリオブジェクトストアから WAL ファイルを取得している場合。指定しない場合、WAL ファイルは一度に 1 つずつ処理されます。値として正の整数を受け入れ、最小受け入れ値は 1 です。
	NA

第146章　サポートされているリリース

このページには、*CloudNativePG* の現在サポートされているリリースのステータス、タイムライン、およびポリシーがリストされています。

私たちは最新の 2 つのマイナーリリースをサポートし、2 か月ごとに新しいマイナーリリースを作成することを目指しています。各リリースは、
table below で宣言された「サポート終了」日までサポートされます。これは、アップグレード計画を可能にする追加の 1 か月のサポートを考慮します。

CloudNativePG のサポートされているリリースには、アクティブなメンテナンス期間にあり、セキュリティとバグ修正のためのパッチが適用されているリリースが含まれます。

マイナーリリースの後続のパッチリリースには、下位互換性の変更のみが含まれます。

- サポートポリシー
- 命名スキーム
- *CloudNativePG* リリースのサポートステータス
- サポートとは

146.1 サポートポリシー

コミットごとに CloudNativePG の新しいビルドを生成します。

約 2 か月ごとに、マイナーリリースをビルドし、いくつかの追加テストとリリース認定を実行します。マイナーリリースで見つかった問題のパッチバージョンをリリースします。

さまざまな種類のリリースは、さまざまな製品品質レベルと CloudNativePG コミュニティからの支援レベルを表しています。コミュニティが提供するサポートの詳細については、
サポートとは を参照してください。

Type	Support level	Quality and recommended Use
Development Build	No support	Dangerous, might not be fully reliable. Useful to experiment with.
Minor Release	Support provided until 1 month after the N+2 minor release (ex. 1.15 supported until 1 month after 1.17.0 is released)	

次のページに続く

表 1 – 前のページからの続き

Type	Support level	Quality and recommended Use
Patch	Same as the corresponding minor release	Users are encouraged to adopt patch releases as soon as they are available for a given release.
Security Patch	Same as a patch, however, it doesn't contain any additional code other than the security fix from the previous patch	Given the nature of security fixes, users are **strongly** encouraged to adopt security patches after release.

利用可能なリリースは releases page で見つけることができます。

> リリースノート では、各マイナーリリースとパッチリリースの高レベルの詳細情報を見つけることができます。

146.2 命名スキーム

命名スキームは Semantic Versioning 2.0.0 に基づいています

次のように

```
<major>.<minor>.<patch>
```

<minor> はリリースごとに増加し、 <patch> は現在の<minor> リリースのパッチの数をカウントします。パッチは、通常、<minor> リリースと比較した小さな変更です。

バージョンの Git タグには、v が先頭に追加されます。

146.3 CloudNativePG リリースのサポートステータス

Version	Currently supported	Release date	End of life	Supported Kubernetes versions	Tested, but not supported	Supported Postgres versions
1.22.x	Yes	December 21, 2023	~ May 21, 2024	1.26, 1.27, 1.28	1.23, 1.24, 1.25	12 - 16
1.21.x	Yes	October 12, 2023	~ April 12, 2024	1.25, 1.26, 1.27, 1.28	1.23, 1.24	12 - 16

次のページに続く

表 2 – 前のページからの続き

Version	Currently supported	Release date	End of life	Supported Kubernetes versions	Tested, but not supported	Supported Postgres versions
main	No, development only					11 - 16

表内のサポートされている Kubernetes バージョンのリストは、CloudNativePG メンテナーが何をサポートおよびテストするのが合理的であると考えているかによって異なります。

現時点では、CloudNativePG コミュニティは、Red Hat OpenShift のような追加の Kubernetes ディストリビューションをサポートまたはテストしていません。これは将来的に変わるかもしれません、その場合、それは CloudNativePG メンテナーが書いた公式ポリシーに反映されます。

146.3.1 サポートされている PostgreSQL バージョン

前の表でサポートされている Postgres バージョンのリストは、一般に、CloudNativePG のマイナーバージョンがリリースされたときにコミュニティでサポートされていた PostgreSQL バージョンによって異なります。

PostgreSQL を参照してください Versioning Policy

サポートされているバージョンの詳細については、ページをご覧ください。

注釈: 2023 年 11 月 9 日より、Postgres 11 はサポートされなくなりました。

また、アップグレードしないのはアップグレードするよりリスクが高いため、PostgreSQL オペランドイメージを定期的に更新し、使用しているメジャーバージョンの最新のマイナーリリースを使用することをお勧めします。その結果、PostgreSQL の古いマイナーバージョンに関する問題をオープンする場合、私たちはあなたを助けることができない場合があります。

146.4 今後のリリース

Version	Release date	End of life	Supported Kubernetes versions
1.23.0	February 22, 2024	•	•

次のページに続く

表 3 – 前のページからの続き

Version	Release date	End of life	Supported Kubernetes versions
1.24.0	April 23, 2024	•	•

注釈: 機能のフリーズは、リリースの 1 週間前に発生します

重要: 将来の日付は不確定であり、変更される可能性があります。これは、Kubernetes バージョンにも適用されます。リリーススケジュールの更新と変更は、 Release updates で通知されます

メイン GitHub リポジトリでのディスカッション。

146.5 古いリリース

Version	Release date	End of life	Compatible Kubernetes versions
1.20.x	April 27, 2023	January 21, 2024	1.24, 1.25, 1.26, 1.27
1.19.x	February 14, 2023	November 3, 2023	1.23, 1.24, 1.25, 1.26
1.18.x	Nov 10, 2022	June 12, 2023	1.23, 1.24, 1.25, 1.26, 1.27
1.17.x	September 6, 2022	March 20, 2023	1.22, 1.23, 1.24
1.16.x	July 7, 2022	December 21, 2022	1.22, 1.23, 1.24
1.15.x	April 21, 2022	October 6, 2022	1.21, 1.22, 1.23

146.6 サポートとは

2 か月ごとに新しい最終リリースを生成することを考慮すると、サポートウィンドウは各リリースブランチ最新の 2 つのマイナーリリースと追加 1 か月ごとに約 5 か月です。

次の図では、`release-1.22` はリリースブランチの例です。

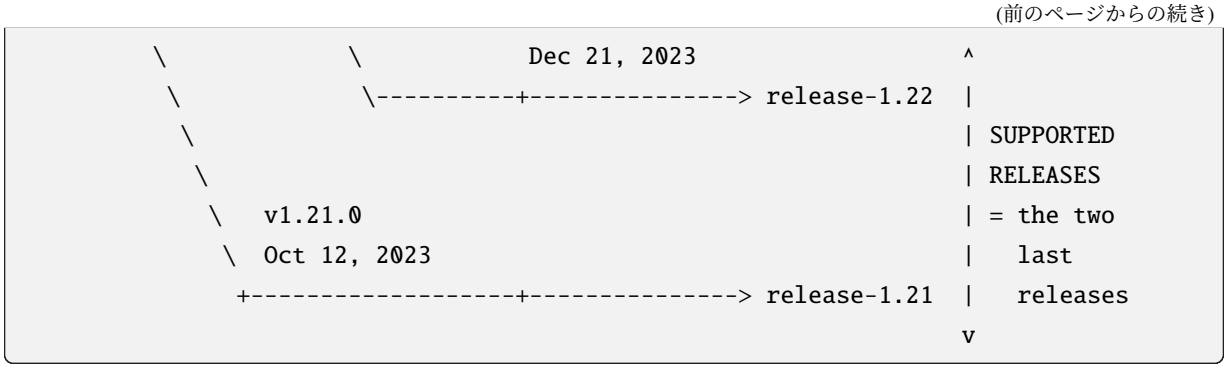
たとえば、最新のリリースが `v1.22.0` の場合、`v1.22.0` と `v1.21.0` の両方のサポートが期待できます。

各ブランチの最後のパッチリリースのみがサポートされています。

```
- -----+-----------------------------------------------> main (trunk development)
       \             \
        \             \
         \             \             v1.22.0
```

(次のページに続く)

(前のページからの続き)



私たちは 2 種類のサポートを提供します。

テクニカルサポートテクニカルサポートは、サポートされているリリースについてのみベストエフォートベースで提供されています。

> CloudNativePG Slack `#general` チャンネル内、または GitHub Discussions を使用して、コミュニティからのサポートを要求できます。

セキュリティとバグ修正重要なバグ修正セキュリティ修正を含むを、現在サポートされているすべてのリリースにバックポートします。パッチをバックポートする前に、私たちは自分自身に問います。**「既にリリースされたバージョンの安定性を本当に重視していることを念頭に置いて、このバックポートは ``*CloudNativePG*`` を向上させますか？」**

専門的なサポートが必要な場合は、

> Support page in the website を参照してください。そこにリストされているベンダーは、サポート期間の延長を含むサービスレベル契約を提供する場合があります。

第147章　Release notes

CloudNativePG のユーザーに表示される変更の履歴、マイナーリリースごとに分類されます。

- *CloudNativePG 1.22* のリリースノート
- *CloudNativePG 1.21* のリリースノート
- *CloudNativePG 1.20* のリリースノート

CloudNativePG のコミュニティサポートポリシーの詳細については、
サポートされているリリース を参照してください。

古いリリース

- *CloudNativePG 1.19* のリリースノート
- *CloudNativePG 1.18* のリリースノート
- *CloudNativePG 1.17* のリリースノート
- *CloudNativePG 1.16* のリリースノート
- *CloudNativePG 1.15* のリリースノート

また、CloudNativePG の前身であるすべての release notes from 1.0.0 to 1.14.0 of Cloud Native PostgreSQL by EDB の記録も保持しています。

147.1 CloudNativePG 1.22 のリリースノート

CloudNativePG の 1.22 マイナーリリースのユーザーに表示される変更の履歴。

変更の完全なリストについては、 commits を参照してください。

GitHub のリリースブランチで。

147.1.1 バージョン 1.22.1

発売日 2024 年 2 月 2 日

機能強化

- `ephemeralVolumeSource` オプション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する#3678

- kubectl の cnpg 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デモンストレーションし、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
- ユーザー名マップのリストを介して、.spec.postgresql.pg_ident スタンザを介して PostgreSQL の ident マップファイルのカスタマイズを許可します#3534

修正

- 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を防ぐ - デフォルト #3728
- ボリュームスナップショットからのテーブルスペースの適切なリカバリー#3682
- インポートプロセス中に 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不足のリスクを削減します。#3743
- 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
- pgpass 接続フィールドの予約文字を適切にエスケープする #3713
- .spec.projectedVolumeTemplate.sources 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#3647
- override.conf に組み込まれたオプションを 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 #3773

147.1.2 バージョン 1.22.0

発売日 2023 年 12 月 21 日

特徴

- **宣言的テーブルスペース** Cluster 仕様に tablespaces スタンザを導入し、PostgreSQL テーブルスペースの包括的なライフサイクル管理を有効にして、垂直スケーラビリティを強化します#3410
- **一時テーブルスペース** .spec.tablespaces[*].temporary オプションを追加して、名前を temp_tablespaces PostgreSQL パラメーター#3464 に組み込むことにより、一時的なデータベース操作でのテーブルスペースの利用を促進します。

セキュリティ

- 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。
- セキュリティ対策を強化するための Docker イメージスキャンの Dockle および Snyk の統合#3300

機能強化

- 外部クラスターのリコンシリエーションの向上#3533
- ALTER SYSTEM コマンドを有効/無効にする機能の導入 #3535

- Cluster および Pooler リソースの.spec.monitoring スタンザの podMonitorMetricRelabelings および podMonitorRelabelings オプションを介した Prometheus の動的なリラベル付けのサポート #3075
- オブジェクトストアバックアップ#2940 とともにボリュームスナップショットを考慮することにより、最初の復旧可能ポイントと最後に成功したバックアップの計算を強化しました。
- PostgreSQL へのネットワーク接続を確立する際の PGPASSFILE 環境変数の使用を削除 #3522
- クラスターの PVC を収集することにより cnpg report プラグインコマンドを改善しました #3357
- cnpg status プラグインコマンドの強化、アラートを含む管理対象ロールに関する情報を提供します#3310
- OLM 展開に適した、オペレーター用の Red Hat UBI 8 コンテナイメージの導入。
- 接続プーラー
  - 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.
  - 値が’pooler’である cnpg.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と区別します#3396

修正

- PodDisruptionBudget リソース#3312 および#3434 のメタデータ、アノテーション、およびラベルの調整。
- 管理対象資格情報シークレットのメタデータの調整 #3316
- 本文の読み取りエラーが全体的なエラーとして処理され、バックアッププロセスが無期限にスタックしたままになるバックアップスナップショットコードのバグの解決#3321。
- immediate-checkpoint または wait-for-archive オプションが要求された場合の cnpg backup プラグインコマンドを使用したオンラインバックアップの暗黙的な設定#3449
- pg_basebackup を介して参加する場合の wal_sender_timeout の無効化 #3586
- 外部クラスターが使用するシークレットのリロード #3565
- 接続プーラー
  - コントローラーが Pooler リソースが所有するすべてのシークレットを監視するように保証#3428.
  - Pooler リソースの RoleBinding の調整#3391.
  - Pooler リソースの imagePullSecret の調整#3389 Reconciliation of16_service Pooler の、必要なラベルの追加#3349.
  - ポッドだけでなく展開への Pooler ラベルの拡張#3350。

変更点

- デフォルトのオペランドイメージを PostgreSQL 16.1#3270 に設定します。
- ALTER SYSTEM コマンドはデフォルトで無効になりました#3545

147.2 CloudNativePG 1.21 のリリースノート

CloudNativePG の 1.21 マイナーリリースのユーザーに表示される変更の履歴。

変更の完全なリストについては、 commits を参照してください。

GitHub のリリースブランチで。

147.2.1 バージョン 1.21.3

発売日 2024 年 2 月 2 日

機能強化

- `ephemeralVolumeSource` オプション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する#3678
- `kubectl` の `cnpg` 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デモンストレーションし、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
- ユーザー名マップのリストを介して、`.spec.postgresql.pg_ident` スタンザを介して PostgreSQL の ident マップファイルのカスタマイズを許可します#3534

修正

- `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を防ぐ - デフォルト #3728
- インポートプロセス中に 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不足のリスクを削減します。#3743
- 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
- `pgpass` 接続フィールドの予約文字を適切にエスケープする #3713
- `.spec.projectedVolumeTemplate.sources` 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#3647
- `override.conf` に組み込まれたオプションを `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 #3773

147.2.2 バージョン 1.21.2

発売日 2023 年 12 月 21 日

セキュリティ

- 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。
- セキュリティ対策を強化するための Docker イメージスキャンの Dockle および Snyk の統合#3300

機能強化

- 外部クラスターのリコンシリエーションの向上#3533
- `ALTER SYSTEM` コマンドを有効/無効にする機能の導入 #3535
- `Cluster` および `Pooler` リソースの `.spec.monitoring` スタンザの `podMonitorMetricRelabelings` および `podMonitorRelabelings` オプションを介した Prometheus の動的なリラベル付けのサポート #3075
- オブジェクトストアバックアップ#2940 とともにボリュームスナップショットを考慮することにより、最初の復旧可能ポイントと最後に成功したバックアップの計算を強化しました。
- PostgreSQL へのネットワーク接続を確立する際の `PGPASSFILE` 環境変数の使用を削除 #3522
- クラスターの PVC を収集することにより `cnpg report` プラグインコマンドを改善しました #3357
- `cnpg status` プラグインコマンドの強化、アラートを含む管理対象ロールに関する情報を提供します#3310
- OLM 展開に適した、オペレーター用の Red Hat UBI 8 コンテナイメージの導入。
- 接続プーラー
  - `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.
  - 値が’pooler’である `cnpg.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と区別します#3396

修正

- `PodDisruptionBudget` リソース#3312 および#3434 のメタデータ、アノテーション、およびラベルの調整。
- 管理対象資格情報シークレットのメタデータの調整 #3316
- 本文の読み取りエラーが全体的なエラーとして処理され、バックアッププロセスが無期限にスタックしたままになるバックアップスナップショットコードのバグの解決#3321。
- `immediate-checkpoint` または `wait-for-archive` オプションが要求された場合の `cnpg backup` プラグインコマンドを使用したオンラインバックアップの暗黙的な設定#3449
- pg_basebackup を介して参加する場合の wal_sender_timeout の無効化 #3586
- 外部クラスターが使用するシークレットのリロード #3565

- 接続プーラー
  - コントローラーがPooler リソースが所有するすべてのシークレットを監視するように保証#3428.
  - Pooler リソースの RoleBinding の調整#3391.
  - Pooler リソースの imagePullSecret の調整#3389 Reconciliation of16_service Pooler の、必要なラベルの追加#3349.
  - ポッドだけでなく展開への Pooler ラベルの拡張#3350。

変更点

- デフォルトのオペランドイメージを PostgreSQL 16.1#3270 に設定します。

147.2.3 バージョン 1.21.1

発売日 2023 年 11 月 3 日

機能強化

- 物理的オンラインベースバックアップ用の PostgreSQL API を使用して、ボリュームスナップショットを使用したオンライン/ホットバックアップのサポートを導入します。特定の Postgres クラスターのホット/コールドバックアップのデフォルト構成は、online オプションと.spec.backup.volumeSnapshot の onlineConfiguration スタンザを介して制御できます。明示的に設定しない限り、ボリュームスナップショットのバックアップはデフォルトでオンラインで取得されるようになりました#3102
- ScheduledBackup および Backup リソースを使用して、ボリュームスナップショットバックアップで上記のデフォルト設定をオーバーライドする機能を導入します#3208、#3226
- ターゲットインスタンススタンバイまたはプライマリがフェンスされる時間枠を削減することにより、ボリュームスナップショットのコールドバックアップを強化します。ボリュームスナップショットがカットおよびプロビジョニングされたらすぐにリフトすることにより、ボリュームスナップショットのコールドバックアップを強化します#3210
- ボリュームスナップショットからのリカバリー中に、既存のラベルと注釈を検証することにより、提供されたボリュームスナップショットがコヒーレントであることを確認します
- kubectl の cnpg プラグインの backup コマンドは、--online 、--immediate-checkpoint 、および--wait-for-archive ランタイムオプションを介してボリュームスナップショットバックアップエクスペリエンスを向上させます
- アクティブなストリーミングベースバックアップの進行状況情報を使用して、kubectl の cnpg プラグインの status コマンドを強化します#3101
- pgBouncer Pooler リソースを使用した max_prepared_statements の構成を許可します #3174

修正

- スイッチオーバー中に WAL アーカイブを一時停止し、完了時に再開します#3227
- 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 を使用するように保証#3143

- `.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#3209
- `cnpg.io/cluster` ラベルを `Pooler` ポッドに設定します #3153
- `VolumeSnapshots` リソースのラベルの数を減らし、より適切なアノテーションにレンダリングします#3151

変更点

- 1.21.0 で導入されたボリュームスナップショットバックアップは、デフォルトでオンライン/ホットになりました。オフライン/コールドバックアップを復元するには、`.spec.backup.volumeSnapshot` を `false` に設定します
- 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` ファイルの使用を停止し、`override.conf` という名前のファイルに置き換えます #2812

技術的な強化

- 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する#3152

147.2.4 バージョン 1.21.0

発売日 2023 年 10 月 12 日

特徴

- **バックアップとリカバリーのボリュームスナップショットのサポート**ボリュームスナップショットで標準の Kubernetes API を活用して、バックアップとリカバリー操作の両方でインクリメンタルおよび差分コピーなどの機能を利用します。スタンバイからのコールドバックアップをカバーするこの最初のステップは、1.22 でも継続され、PostgreSQL API とテーブルスペースを使用したホットバックアップのサポートになります。
- **OLM インストール方法** *stable* チャネルを介して最新のマイナーリリースの最新パッチバージョンの OperatorHub.io を介して Operator Lifecycle Manager のサポートを導入します。「EDB Postgres for Kubernetes 」オペレーターのバンドルを寄付し、CloudNativePG に適応させてくださった EDB に感謝します。

重要な変更

- `stopDelay` のデフォルト値を 30 秒ではなく 1800 秒に変更します #2848
- 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ための `smartShutdownTimeout` と呼ばれる新しいパラメーターを導入します。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)` #2848
- `startDelay` のデフォルト値を 30 秒ではなく 3600 に変更します #2847
- 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処する#2847
- `switchoverDelay` のデフォルト値を 40000000 秒ではなく 3600 秒に変更します #2846

- セキュリティのためにデフォルトでスーパーユーザーアクセスを無効にする#2905
- デフォルトで HA のレプリケーションスロットを有効にする #2903
- `postgresql` label のサポートを停止します - 1.18 の `cnpg.io/cluster` に置き換えられました#2744

セキュリティ

- デフォルトの `seccompProfile` をオペレーター展開に追加します #2926

機能強化

- ボリュームスナップショットの一貫したセットからのレプリカクラスターのブートストラップを有効にする#2647
- 一貫したボリュームスナップショットのセットからの完全なポイントインタイムリカバリーを有効にする#2390
- `cnpg.io/coredumpFilter` アノテーションを導入して、PostgreSQL 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します。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設定されています#2733
- 共有メモリと一時データのエフェメラルボリュームのエフェメラルストレージ制限を構成できます #2830
- Webhook を介したリソース制限と要求を検証する #2663
- 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#2840
- 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ときに容易にします#2186
- 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` を追加します #2612
- `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を向上させます#2243
- 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す #2953

修正

- 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#2544
- リカバリー中に `cnpg.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#2731
- PostgreSQL サーバーはダウンしているが、ポッドがプライマリで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ぐ#2966
- レプリケーションスロットが有効になっているときに、レプリカクラスター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#2960
- 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#2918

- 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をデフォルト app に設定する #2957
- 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する#2884
- 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 #2875
- 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 #2625
- バックアップが完了または失敗した場合にのみ停止することにより、中断に対する Backup リソース調整サイクルの復元力を高めます #2591
- PodMonitor labels および annotations を調整する #2583
- Role オブジェクトの RBAC resourceNames の欠落によるバックアップ失敗を修正しました #2956
- 可観測性
  - 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す #2961
  - プロメテウスの pg_wal_stat デフォルトメトリックを修正します#2569
  - プロメテウスの pg_replication のデフォルトメトリックを改善します #2744 および#2750
  - 代わりに edb_traned_13 edb_13 を使用します提供された Grafana ダッシュボード
  - メトリック収集で standard_conforming_strings を適用する #2888

変更点

- 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
- 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#3051
- Webhook の名前を kb.io から cnpg.io グループに変更します #2851
- kubectl プラグインの cnpg snapshot コマンドを cnpg backup -m volumeSnapshot に置き換えます
- cnpg hibernate プラグインコマンドで PVC の ClusterManifestAnnotationName および PgControldataAnnotationName アノテーションを使用させましょう #2657
- cnpg.io/instanceRole ラベルを追加しますが、既存の role ラベルを非推奨にします #2915

技術的な強化

- 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 に置き換えます #2606

147.3 CloudNativePG 1.20 のリリースノート

CloudNativePG の 1.20 マイナーリリースのユーザーに表示される変更の履歴。

変更の完全なリストについては、 commits を参照してください。

GitHub のリリースブランチで。

147.3.1 バージョン 1.20.6

発売日 2024 年 2 月 2 日

> **警告:** これは、1.20.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。

機能強化

- `ephemeralVolumeSource` オプション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する#3678
- `kubectl` の `cnpg` 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デモンストレーションし、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
- ユーザー名マップのリストを介して、`.spec.postgresql.pg_ident` スタンザを介して PostgreSQL の ident マップファイルのカスタマイズを許可します#3534

修正

- `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を防ぐ - デフォルト #3728
- インポートプロセス中に 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不足のリスクを削減します。#3743
- 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
- `pgpass` 接続フィールドの予約文字を適切にエスケープする #3713
- `.spec.projectedVolumeTemplate.sources` 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#3647

147.3.2 バージョン 1.20.5

発売日 2023 年 12 月 21 日

セキュリティ

- 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。
- セキュリティ対策を強化するための Docker イメージスキャンの Dockle および Snyk の統合#3300

機能強化

- 外部クラスターのリコンシリエーションの向上#3533
- `ALTER SYSTEM` コマンドを有効/無効にする機能の導入 #3535
- `Cluster` および `Pooler` リソースの `.spec.monitoring` スタンザの `podMonitorMetricRelabelings` および `podMonitorRelabelings` オプションを介した Prometheus の動的なリラベル付けのサポート #3075
- PostgreSQL へのネットワーク接続を確立する際の `PGPASSFILE` 環境変数の使用を削除 #3522
- クラスターの PVC を収集することにより `cnpg report` プラグインコマンドを改善しました #3357
- `cnpg status` プラグインコマンドの強化、アラートを含む管理対象ロールに関する情報を提供します#3310
- 接続プーラー
  - `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.
  - 値が’pooler’である `cnpg.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と区別します#3396

修正

- `PodDisruptionBudget` リソース#3312 および#3434 のメタデータ、アノテーション、およびラベルの調整。
- 管理対象資格情報シークレットのメタデータの調整 #3316
- 本文の読み取りエラーが全体的なエラーとして処理され、バックアッププロセスが無期限にスタックしたままになるバックアップスナップショットコードのバグの解決#3321。
- `immediate-checkpoint` または `wait-for-archive` オプションが要求された場合の `cnpg backup` プラグインコマンドを使用したオンラインバックアップの暗黙的な設定#3449
- pg_basebackup を介して参加する場合の wal_sender_timeout の無効化 #3586
- 外部クラスターが使用するシークレットのリロード #3565
- 接続プーラー
  - コントローラーが`Pooler` リソースが所有するすべてのシークレットを監視するように保証#3428.
  - `Pooler` リソースの `RoleBinding` の調整#3391.

  - Pooler リソースの imagePullSecret の調整#3389 Reconciliation of16_service Pooler の、必要なラベルの追加#3349.
  - ポッドだけでなく展開への Pooler ラベルの拡張#3350。

変更点

- デフォルトのオペランドイメージを PostgreSQL 16.1#3270 に設定します。

147.3.3 バージョン 1.20.4

発売日 2023 年 11 月 3 日

機能強化

- アクティブなストリーミングベースバックアップの進行状況情報を使用して、kubectl の cnpg プラグインの status コマンドを強化します#3101
- pgBouncer Pooler リソースを使用した max_prepared_statements の構成を許可します #3174

修正

- スイッチオーバー中に WAL アーカイブを一時停止し、完了時に再開します#3227
- 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 を使用するように保証#3143
- .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#3209
- cnpg.io/cluster ラベルを Pooler ポッドに設定します #3153

変更点

- 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 ファイルの使用を停止し、override.conf という名前のファイルに置き換えます #2812

技術的な強化

- 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する#3152

147.3.4 バージョン 1.20.3

発売日 2023 年 10 月 11 日

重要な変更

- stopDelay のデフォルト値を 30 秒ではなく 1800 秒に変更します #2848
- 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ための smartShutdownTimeout と呼ばれる新しいパラメーターを導入します。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) #2848

- `startDelay` のデフォルト値を 30 秒ではなく 3600 に変更します #2847
- 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処する#2847
- `switchoverDelay` のデフォルト値を 40000000 秒ではなく 3600 秒に変更します #2846
- `postgresql` label のサポートを停止します - 1.18 の `cnpg.io/cluster` に置き換えられました#2744

セキュリティ

- デフォルトの `seccompProfile` をオペレーター展開に追加します #2926

機能強化

- `cnpg.io/coredumpFilter` アノテーションを導入して、PostgreSQL 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します。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設定されています#2733
- 共有メモリと一時データのエフェメラルボリュームのエフェメラルストレージ制限を構成できます #2830
- Webhook を介したリソース制限と要求を検証する #2663
- 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#2840
- 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ときに容易にします#2186
- 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` を追加します #2612
- `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を向上させます#2243
- 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す #2953

修正

- 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#2544
- リカバリー中に `cnpg.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#2731
- PostgreSQL サーバーはダウンしているが、ポッドがプライマリで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ぐ#2966
- レプリケーションスロットが有効になっているときに、レプリカクラスター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#2960
- 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#2918
- 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をデフォルト `app` に設定する #2957
- 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する#2884

- 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 #2875
- 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 #2625
- バックアップが完了または失敗した場合にのみ停止することにより、中断に対する Backup リソース調整サイクルの復元力を高めます #2591
- PodMonitor labels および annotations を調整する #2583
- Role オブジェクトの RBAC resourceNames の欠落によるバックアップ失敗を修正しました #2956
- 可観測性
  - 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す #2961
  - プロメテウスの pg_wal_stat デフォルトメトリックを修正します#2569
  - プロメテウスの pg_replication のデフォルトメトリックを改善します #2744 および#2750
  - 代わりに edb_traned_13 edb_13 を使用します提供された Grafana ダッシュボード
  - メトリック収集で standard_conforming_strings を適用する #2888

変更点

- 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
- 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#3051
- Webhook の名前を kb.io から cnpg.io グループに変更します #2851
- kubectl プラグインの cnpg snapshot コマンドを cnpg backup -m volumeSnapshot に置き換えます
- cnpg hibernate プラグインコマンドで PVC の ClusterManifestAnnotationName および PgControldataAnnotationName アノテーションを使用させましょう #2657
- cnpg.io/instanceRole ラベルを追加しますが、既存の role ラベルを非推奨にします #2915

技術的な強化

- 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 に置き換えます #2606

147.3.5 バージョン 1.20.2

発売日 2023 年 7 月 27 日

機能強化

- kubectl プラグインの新しい logs コマンド、クラスター内のすべてのポッドのログを取得または追跡する#2375

- ポッドで priorityClassName を指定するサポートを追加し、Kubernetes のスケジュール決定を支援します#2043
- CloudNativePG クラスターによるノードの使用状況をモニタするためのメトリックとステータスフィールドを追加します#2257
- ドキュメントのさまざまな機能強化
  - hugepages に関連するトラブルシューティング手順を追加します#1390
  - FAQ ページを拡張します#2344

技術的な強化

- 復元プロセスの先頭にチェックを追加して、続行できることを確認します。できない場合は、改善されたエラー診断を提供します#2419
- 管理対象ロールでの無期限パスワードの処理を改善します#2334

修正

- リカバリターゲット設定のロジックが Postgres のロジックと一致することを確認します#2460
- 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す#2462
- initdb およびリストアのための PGDATA ディレクトリの正しいアクセス許可を確認します#2384
- 構成のリロード中にインスタンスマネージャーのクラッシュの可能性を防止#2393
- 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 #1751
- サービスが非インスタンスポッドをターゲットとしないようにする #2336

セキュリティ

- 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

147.3.6 バージョン 1.20.1

発売日 2023 年 6 月 12 日

機能強化

- `snapshot` コマンドを `cnpg` プラグインに追加して、Kubernetes `VolumeSnapshot` 標準リソースを使用してスタンバイからクラスターの一貫したコールドバックアップを作成します#1960
- `.spec.bootstrap.recovery.volumeSnapshot` スタンザを介した CSI VolumeSnapshot リソースのセットからのリカバリの最初の実装#1960
- `pg_failover_slots` をマネージド拡張機能に追加します #2057
- ドキュメントとクイックスタートガイドの指示を更新した Grafana ダッシュボードを改善しました #1916

- `import` スタンザに `schemaOnly` オプションを導入して、1 つ以上の既存のデータベースから新しい Postgres クラスターをブートストラップするときにデータのエクスポートおよびインポートを回避します#2234
- インスタンスポッドのスケジュールを管理するための TopologySpreadConstraints のサポートを追加します#2202
- PgBouncer の `Pooler` に `PodMonitor` サポートを追加します#2034
- デフォルトの Kubernetes スケジューラーをオーバーライドするオプションを追加します#2013
- `Pooler` リソースのデプロイメント戦略の構成を許可します #1983
- デフォルトの PostgreSQL バージョンを 15.3 に更新します#2022
- デフォルトで PgBouncer 1.19 を使用する#2018

技術的な強化

- 更新された k8s 種類のテスト済みバージョン#2054
- 宣言的ロールはパスワードが設定されていない場合を無視し、以前に存在したロールの管理を容易にする必要があります#2029
- 別のトランザクションを使用して、ロールの資格情報を調整します。このパッチの前は、オペレーターは、いずれかが失敗した場合、すべてのロールの同期を元に戻していました#2004
- クラスターの復元中にフェンシングが削除されることを確認します #1987
- ポッド削除時のロギングを改善します #2136

修正

- K3 でのセットアップを妨げる可能性がある k3s エンジンのバインドされていない変数を修正しました#2157
- メトリックで正しい PG バージョンをレポートする #2126
- ドキュメントで正しい walStorage キーを使用する #2140
- オペレーターがインスタンスに接続できない場合に調整を停止し、このような場合に明確な診断を提供します。これは、ネットワークの問題が CloudNativePG の通常のオペレーションを妨げるケースを明確にするのに役立ちます#2145、#2233、および#2242

147.3.7 バージョン 1.20.0

発売日 2023 年 4 月 27 日

特徴

- **宣言的ロール管理** `Cluster` 仕様に `managed.roles` スタンザを導入して、`CREATE ROLE` や `ALTER ROLE` などの PostgreSQL の関連 DDL コマンドに抽象化を提供することにより、データベースロールの完全なライフサイクル管理を提供します#1524、#1793 および#1815)

- **PostgreSQL クラスターの宣言的ハイバーネーション** `cnpg.io/hibernation` と呼ばれる新しいアノテーションを導入して、すべてのポッドを削除し、PVC のみを保持することにより PostgreSQL クラスターを宣言的にハイバーネーションさせます。この機能は、インバースプロシージャも実装します #1657

機能強化

- `kubectl` の `cnpg` プラグインの `report` コマンドの`--logs` オプションを改善して、可能な場合は以前のログも含めます #1811
- `-any` サービスがデフォルトで無効になりました #1755

セキュリティ

- ローカルファイルを介したオーバーライドを介して `SeccompProfile` のカスタマイズを有効にします #1827

修正

- サーバーが初めて起動される前に、`initdb` ブートストラップフェーズ中にユーザーが提供した PostgreSQL 構成を適用します#1858

147.4 1.14.0 以前のリリースノート

CloudNativePG の最初のパブリックリリースはバージョン 1.15.0 です。それまでは、製品は EDB が完全に所有しており、
Cloud Native PostgreSQL という名前で配布されていました。

このページの変更のリストは情報提供のみを目的とており、コミットに加えて製品の履歴を示します。**ここにリストされているバージョンは、CloudNativePG には存在しません。**

147.4.1 バージョン 1.14.0

発売日 2022 年 3 月 25 日

特徴

- 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
- `Cluster` オブジェクトの `LastBackupSucceeded` 条件の導入によるバックアップの可観測性の向上
- ホットスタンバイの重要なパラメーターの更新のサポート `max_connections` 、`max_prepared_transactions` 、`max_locks_per_transaction` 、`max_wal_senders` 、`max_worker_processes`
- `Cluster` 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
- 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
- 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

- デフォルトの PostgreSQL バージョンを 14.2 に更新しました
- 新しいコマンドを `maintenance` という名前の `kubectl cnp` プラグインに追加して、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

コンテナイメージ

- 最新の PostgreSQL コンテナには Barman Cloud 2.19 が含まれます

セキュリティの強化

- 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。

修正

- 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
- `pg_rewind` を実行する前に古い PID ファイルをクリーンアップする
- `primary_conninfo` のキーによるソートを強制して、13 より前の PostgreSQL バージョンでのランダムな再起動を回避します
- リコンシリエーション時に `ServiceAccount` 変更ラベル、アノテーションなどを保存します
- imagePullPolicy のデフォルト値の適用を無効にする
- WAL セグメントサイズの initDB 検証を改善します
- 指定された LSN で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する
- 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

147.4.2 バージョン 1.13.0

リリース日 2022 年 2 月 17 日

特徴

- Snappy 圧縮のサポート。Snappy は、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の速度を向上させるバックアップの高速圧縮オプションです
- 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
- `status.conditions` によるクラスターのステータスの拡張。条件 `ContinuousArchiving` 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
- 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 `Cluster Summary` 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 `Certificates Status` セクションを追加します

- 新しいクラスターの作成時に空ではないバックアップ先を検出する新しい barman-cloud-check-wal-archive コマンドをサポート
- Secret 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ーを追加するサポートを追加します。
- 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
- 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
- Kubernetes 1.18 および非推奨の API バージョンのサポートを終了

コンテナイメージ

- PostgreSQL コンテナには Barman 2.18 が含まれます

セキュリティ修正

- 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

修正

- Postgres ポッドからのコード取得ステータスのメモリリークを修正
- 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
- 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 が起動できなくなります
- 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冗長性を削減します
- まだ初期化されていないときにクラスターが Healthy としてマークされるバグを修正
- 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
- まれにクラッシュにつながる可能性がある PostgreSQL ポッドの起動中の競合状態を修正します
- 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
- 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
- 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
- クラスターの作成中に PostgreSQL 構成の検証が欠落していたのを修正

147.4.3 バージョン 1.12.0

発売日 2022 年 1 月 11 日

特徴

- Kubernetes 1.23 をサポートされる Kubernetes ディストリビューションのリストに追加し、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。
- 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を強化します
- `.spec.inheritedMetadata` フィールドをクラスターに導入して、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
- 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
- PgBouncer の準備状況プローブを追加します
- 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

修正

- 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した
- 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
- レプリカクラスターソースの 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
- 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する
- 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もデフォルトでクラスターを更新するようになりました
- 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稀な問題を修正します
- 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
- 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
- `enablePodMonitor` が false の場合、調整サイクルごとに PodMonitor 削除のロギングを停止します
- 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

147.4.4 バージョン 1.11.0

発売日 2021 年 12 月 15 日

特徴

- **パラレル WAL アーカイブとリストア** backupObjectStore.maxParallel オプションを導入して、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。PostgreSQL の archive_command によると、両方の WAL アーカイブ中に実行するパラレルジョブの最大数を設定します。WAL リストア restore_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速く準備状態に到達できます。
- **モニタリングのため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、または.spec.monitoring.disableDefaultQueries オプションデフォルトで false に設定を介してクラスターレベルで変更できます。
- 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
- ポッドが Kubernetes によって強制終了される前に、目的の stopDelay 時間の前半にスマートシャットダウンを実行し、残りの半分に高速シャットダウンを実行することにより、PostgreSQL のシャットダウンプロシージャを改善します
- 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、CloudNativePG は無期限に待機して特権データの耐久性
- ローリング更新を発行することにより、PostgreSQL Cluster のリソース要求と制限の変更を処理します
- kubectl の cnp プラグインの status コマンドを改善し、追加情報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
- PostgreSQL の共有メモリ管理の 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、多くの並列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
- 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
- Kubelet によって追い出されたポッドを自動的に削除します
- 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無効
- 特定の Postgres クラスターで disabled に設定されている場合、調整ループの実行を防止する cnpg.io/reconciliationLoop アノテーションを導入します

- 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

修正

- 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しますが、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 に残しましたステータス。
- Liveness プローブは、タイムライン流用後に 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。
- Postgres ポッドからクライアントサイドの調整を削減します Waited for 1.182388649s due to client-side throttling, not priority and fairness, request: GET など
- 提供されたものを使用して、OpenShift および OLM インストールでの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
- 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
- ジョブが正常に完了した後にのみ準備ができている PVC をマークし、PVC の初期化の競合状態を防ぎます
- 有効期限が切れた Webhook TLS シークレットを更新するときに、正しい公開キーを使用します。
- LSN を解析する際のオーバーフローを修正
- 起動時に古い PID ファイルを削除する
- Pooler リソースに、オペレーターで定義された imagePullSecret が存在する場合を継承させます

147.4.5 バージョン 1.10.0

発売日 2021 年 11 月 11 日

特徴

- **PgBouncer を使用した接続プーリング** Pooler リソースとコントローラを導入して、ローカル PostgreSQL Cluster 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 エクスポータが含まれます
- **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'
- **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。*documentation for more detailed information* を参照してください。

- `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`、`encoding`、`localeCollate`、`localeCType`、`walSegmentSize` に制限します。これにより、`options` 配列は時代遅れになり、v2 API で削除される予定です
- `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 クエリー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
- インスタンス Prometheus エクスポーターに新しいデフォルトメトリックを追加しました。バックアップカタログに基づく Postgres バージョン、クラスター名、およびリカバリ可能の最初のポイント
- 障害後のバックアップの再試行
- オペレーターが、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、Azure Blob Container ストレージなどを呼び出さないようにするために、 Barman Cloud 機能に関する認識を高めます。
- `cnp` プラグインの `status` コマンドの出力とバックアップに関する情報を統合
- 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
- `Backup` オブジェクトを含む `Cluster` で生成されたすべてのオブジェクトの `k8s.enterprisedb.io/cluster` ラベルにクラスター名を設定します
- `Cluster`、`Backup`、および `ScheduledBackup` 種類での非推奨の API バージョン `postgresql.cnpg.io/v1alpha1` のサポートをドロップ
- 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

セキュリティ

- オペレーターコンテナ securityContext の allowPrivilegeEscalation を `false` に設定します

修正

- 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 を無効にする
- 復旧操作中に正しい証明書認証局 CA を使用する
- 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
- ログをフラッディングする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

147.4.6 バージョン 1.9.2

発売日 2021 年 10 月 15 日

特徴

- 2 つの新しいロガーで JSON ログを強化します。PostgreSQL の `archive_command` の場合は `wal-archive`、スタンバイの `restore_command` の `wal-restore`

修正

- スタンバイプロモーション中に WAL アーカイブを有効にする `.history` ファイルがアーカイブされるのを防止
- 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
- 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

147.4.7 バージョン 1.9.1

発売日 2021 年 9 月 30 日

*このリリースは、 PostgreSQL 14 のローンチを記念するものです

特定のイメージ名を定義せずに新しい `Cluster` 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。*

修正

- `Error while getting barman endpoint CA secret` メッセージがプライマリポッドのログに表示され、バックアップが正しく機能しない問題を修正
- 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

147.4.8 バージョン 1.9.0

発売日 2021 年 9 月 28 日

バージョン *1.9.0* は、リリースプロセスとバージョン *1.9.1* のその後のリリースの遅延により、*OpenShift* では利用できません。

特徴

- 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
- 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` に`--restore-target-wal` オプションのサポートを導入します PostgreSQL 13 以降でのみ利用可能

- 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する
- 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

修正

- 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 の制限された SCC で必要
- ステータスの FirstRecoverabilityPoint を適切に更新します
- 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 を設定します
- マイナーなバグ修正

147.4.9 バージョン 1.8.0

発売日 2021 年 9 月 13 日

特徴

- 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。
- スケジュールされたバックアップに immediate 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
- 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
- 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。
  1. ディスク上の WAL ファイルの数と計算された合計サイズ b_tran_3 アーカイブステータスフォルダー内の.ready および.done ファイルの数 b_tran_6 レプリカモードのフラグ
  2. 要求された最小/最大同期の数レプリカ、および予想および実際に観察されたレプリカ
- 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さまざまな PostgreSQL バージョンに制限します
- 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
- liveness プローブに pg_isready に依存する
- 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
- .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

- OpenShift 4.5 に代わる OpenShift 4.8 のサポートを追加します
- PostgreSQL 14 ベータ版のサポート
- ストリーミング接続を必要とせず、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でレプリカクラスター機能を強化します実験
- クラスターの仕様に `logLevel` オプションを導入して、エラー、情報、デバッグ、またはトレースのいずれかのレベルを指定します

セキュリティの強化

- `.spec.enableSuperuserAccess` を導入して、パスワード認証を介して `postgres` ユーザーのネットワークアクセスを有効/無効にします

修正

- 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

147.4.10 バージョン 1.7.1

発売日 2021 年 8 月 11 日

特徴

- 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
- `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 クエリーのリストを指定します

修正

- 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を許可します 1.7.0 で導入されたバグ
- `LOCAL` 同期コミットレベルを使用して管理照会を実行します
- PGAudit の複数行のログエントリを正しく解析する

147.4.11 バージョン 1.7.0

発売日 2021 年 7 月 28 日

特徴

- JSON 出力で直接使用可能な `pgaudit` と呼ばれる新しいタイプの `logger` を使用して、PGAudit にネイティブサポートを追加します
- – によるモニタリングと可観測機能の強化 `pg_stat_statements` および `auto_explain` 拡張機能のネイティブサポート

  - 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 Prometheus エクスポータの `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)
  - `primaryUpdateStrategy` が `supervised` に設定された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するための `manual_switchover_required` メトリックの公開
- `pg_audit` 、`auto_explain` 、および `pg_stat_statements` の `shared_preload_libraries` の透過的ハンドル
- `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の PostgreSQL の `shared_preload_libraries` の自動構成
- `cnpg.io/reload` ラベルをサポートして、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。プロメテウスエクスポータ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もの、または証明書を保存するために使用されるものを含む
- `reload` コマンドを `kubectl` の `cnp` プラグインに追加して、インスタンスで調整ループをトリガーする
- `additionalPodAffinity` 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
- 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

セキュリティの強化

- `.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認証局と PostgreSQL サーバーの証明書を定義します。クライアント証明書の認証と PostgreSQL ノード間の安全な通信に使用されます
- `.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

修正

- Prometheus エクスポーターでヒストグラムを正しく解析する
- クラスター用にオペレーターが作成したサービスの調整

147.4.12 バージョン 1.6.0

発売日 2021 年 7 月 12 日

特徴

- レプリカモード **実験**ソースクラスターのレプリカとしてクラスターを作成できます。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。
- `.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。

- `.spec.affinity.podAntiAffinityType` パラメーターを追加します。`preferred` デフォルトに設定して `preferredDuringSchedulingIgnoredDuringExecution` が使用されるか、`required` に設定して `requiredDuringSchedulingIgnoredDuringExecution` になります。

変更点

- PVC とポッドを削除する際の競合状態を修正し、オペレーターが新しいポッドを作成できませんでした。
- 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。
- 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。
- 一部のログメッセージのレベルを `debug` に下げました。
- サーバー CA および TLS インジェクションの E2E テスト。

147.4.13 バージョン 1.5.1

発売日 2021 年 6 月 17 日

変更します

- Red Hat OpenShift でのオペレーター展開での自動更新を防止する CRD 検証のバグを修正
- 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。

147.4.14 バージョン 1.5.0

発売日 2021 年 6 月 11 日

特徴

- `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
- `Cluster` 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。
- より決定的で反復可能な展開のために、`<image>:<tag>@sha256:<digestValue>` 形式を介してイメージ名へのダイジェストの仕様を有効にします

セキュリティの強化

- 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
- 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 `sslmode` を `require` から `verify-ca` に引き上げます

変更点

- kubectl の cnp プラグインの promote サブコマンドを強化して、ポッドの名前全体ではなくノード番号のみを受け入れるようにしました
- クラスター名に DNS-1035 検証スキームを採用サービス名が継承されます
- 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する
- 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 、stoppedAt と統合
- ドキュメントの改善
  - オペレーターとオペランドによって公開されるポートのリストを提供します container
  - データベースワークロードのストレージと PostgreSQL のベンチマークのための cnp-bench helpm チャートとガイドラインを紹介します
- E2E テストの機能強化
  - Kubernetes 1.21
  - をテストする オペレーターの高可用性のテストを追加します
  - ノードドレインのテストを追加します
- マイナーバグ修正、以下を含むマイナーバグ修正
  - リカバリ操作中の pg_ctl start へのタイムアウトが短すぎます b_tran_8 オペレーターが PVCs で直接イベントを監視していない
  - barmanObjectStore バックアップの immediateCheckpoint および jobs パラメーターの処理を修正します
  - バックアップから復旧するときに空のログ

147.4.15 バージョン 1.4.0

発売日 2021 年 5 月 18 日

特徴

- JSON 形式の PostgreSQL エラーメッセージの標準出力ログ
- 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
- restart コマンドを kubectl の cnp プラグインに追加して、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

セキュリティの強化

- ポッドの readOnlyRootFilesystem セキュリティコンテキストを設定します

変更点

- **重要** YAML マニフェストを使用して CloudNativePG 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、CloudNativePG ドキュメントを参照してください。
- 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: cloudnative-pg` に変更します
- Prometheus エクスポーターの TCP ポートに `metrics` 名前を割り当てます
- PostgreSQL のメトリックエクスポータ接続に `application_name` として `cnp_metrics_exporter` を設定します
- 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
- ドキュメントの改善
  - モニタリングクエリーのカスタマイズ
  - Operator のアップグレード手順
- E2E テストの機能強化
- マイナーバグ修正など
  - `pg_basebackup`
  - を呼び出すときに`-R` の使用を回避します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

147.4.16 バージョン 1.3.0

発売日 2021 年 4 月 23 日

特徴

- ラベルとアノテーションの継承
- すべてのコンテナにリソース制限を設定する

セキュリティの強化

- 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で、ポッドの実行を名前空間に割り当てられた UID および SELinux コンテキストに制限します
- 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

変更点

- 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

- ドキュメントの改善
- GKE 環境を含む E2E テストの機能強化
- マイナーなバグ修正

147.4.17 バージョン 1.2.1

発売日 2021 年 4 月 6 日

- ScheduledBackup はバックアップの所有者でなくなりました。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
- 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

147.4.18 バージョン 1.2.0

発売日 2021 年 3 月 31 日

- 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
- Operator Lifecycle ManagerOLM 展開をサポートし、その後の OperatorHub.io でのプレゼンス
- 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 から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
- AKS での E2E テストの改善
- マイナーなバグ修正

147.4.19 バージョン 1.1.0

発売日 2021 年 3 月 3 日

- JSON および YAML 出力を含むクラスターのステータスを整形印刷するために `kubectl cnp status` を追加
- `kubectl cnp certificate` 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
- `-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
- 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
- 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する
- `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
- マイナーなバグ修正

147.4.20 バージョン 1.0.0

発売日 2021 年 2 月 4 日

CloudNativePG の最初のメジャー安定リリースは、API グループ `postgresql.cnpg.io/v1` の `Cluster` 、`Backup` 、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。

- 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
- 最も調整されたレプリカ
  - レプリカの自動再作成を促進す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
- 選択したレプリカの昇格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
- スケールアップ/ダウン機能
- 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
- 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
- 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 *read* サービスの定義
- 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
- ポッドの永続ボリュームストレージの再利用
- 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アップデート
- TLS 接続とクライアント証明書認証
- S3 互換オブジェクトストアへの継続的なバックアップ
- 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
- 同期レプリカのサポート
- `nodeSelector` プロパティを介したノードアフィニティのサポート
- PostgreSQL エラーメッセージの標準出力ログ

147.5 Release notes old version

147.5.1 CloudNativePG 1.19 のリリースノート

CloudNativePG の 1.19 マイナーリリースのユーザーに表示される変更の履歴。

変更の完全なリストについては、 commits を参照してください。

GitHub のリリースブランチで。

バージョン 1.19.6

発売日 2023 年 11 月 3 日

> **警告:** これは、1.19.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。

機能強化

- アクティブなストリーミングベースバックアップの進行状況情報を使用して、`kubectl` の `cnpg` プラグインの `status` コマンドを強化します#3101
- pgBouncer `Pooler` リソースを使用した `max_prepared_statements` の構成を許可します #3174

修正

- スイッチオーバー中に WAL アーカイブを一時停止し、完了時に再開します#3227
- 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` を使用するように保証#3143
- `.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#3209
- `cnpg.io/cluster` ラベルを `Pooler` ポッドに設定します #3153

変更点

- 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` ファイルの使用を停止し、`override.conf` (#2812) という名前のファイルに置き換えます。

技術的な強化

- 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する#3152

バージョン 1.19.5

発売日 2023 年 10 月 11 日

> **警告:** バージョン 1.19 は、2023 年 11 月 9 日にサポート終了 EOL に達します。まだ行っていない場合は、できるだけ早くアップグレードの計画を開始してください。

重要な変更

- `stopDelay` のデフォルト値を 30 秒ではなく 1800 秒に変更します #2848
- 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ための `smartShutdownTimeout` と呼ばれる新しいパラメーターを導入します。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)` #2848

- `startDelay` のデフォルト値を 30 秒ではなく 3600 に変更します #2847
- 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処する#2847
- `switchoverDelay` のデフォルト値を 40000000 秒ではなく 3600 秒に変更します #2846
- `postgresql` label のサポートを停止します - 1.18 の `cnpg.io/cluster` に置き換えられました#2744

セキュリティ

- デフォルトの `seccompProfile` をオペレーター展開に追加します #2926

機能強化

- `cnpg.io/coredumpFilter` アノテーションを導入して、PostgreSQL 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します。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設定されています#2733
- 共有メモリと一時データのエフェメラルボリュームのエフェメラルストレージ制限を構成できます #2830
- Webhook を介したリソース制限と要求を検証する #2663
- 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#2840
- 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ときに容易にします#2186
- 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` を追加します #2612
- `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を向上させます#2243
- 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す #2953

修正

- 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#2544
- リカバリー中に `cnpg.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#2731
- PostgreSQL サーバーはダウンしているが、ポッドがプライマリで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ぐ#2966
- レプリケーションスロットが有効になっているときに、レプリカクラスター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#2960
- 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#2918
- 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をデフォルト `app` に設定する #2957
- 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する#2884

- 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 #2875
- 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 #2625
- バックアップが完了または失敗した場合にのみ停止することにより、中断に対する Backup リソース調整サイクルの復元力を高めます #2591
- PodMonitor labels および annotations を調整する #2583
- Role オブジェクトの RBAC resourceNames の欠落によるバックアップ失敗を修正しました #2956
- 可観測性
  - 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す #2961
  - プロメテウスの pg_wal_stat デフォルトメトリックを修正します#2569
  - プロメテウスの pg_replication のデフォルトメトリックを改善します #2744 および#2750
  - 代わりに edb_traned_13 edb_13 を使用します提供された Grafana ダッシュボード
  - メトリック収集で standard_conforming_strings を適用する #2888

変更点

- 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
- 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#3051
- Webhook の名前を kb.io から cnpg.io グループに変更します #2851
- cnpg hibernate プラグインコマンドで PVC の ClusterManifestAnnotationName および PgControldataAnnotationName アノテーションを使用させましょう #2657
- cnpg.io/instanceRole ラベルを追加しますが、既存の role ラベルを非推奨にします #2915

技術的な強化

- 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 に置き換えます #2606

バージョン 1.19.4

発売日 2023 年 7 月 27 日

機能強化

- kubectl プラグインの新しい logs コマンド、クラスター内のすべてのポッドのログを取得または追跡する#2375
- ポッドで priorityClassName を指定するサポートを追加し、Kubernetes のスケジュール決定を支援します#2043
- CloudNativePG クラスターによるノードの使用状況をモニタするためのメトリックとステータスフィールドを追加します#2257

- ドキュメントのさまざまな機能強化
  - hugepages に関連するトラブルシューティング手順を追加します#1390
  - FAQ ページを拡張します#2344

技術的な強化

- 復元プロセスの先頭にチェックを追加して、続行できることを確認します。できない場合は、改善されたエラー診断を提供します#2419

修正

- リカバリターゲット設定のロジックが Postgres のロジックと一致することを確認します#2460
- 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す#2462
- initdb およびリストアのための PGDATA ディレクトリの正しいアクセス許可を確認します#2384
- 構成のリロード中にインスタンスマネージャーのクラッシュの可能性を防止#2393
- 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 #1751
- サービスが非インスタンスポッドをターゲットとしないようにする #2336

セキュリティ

- 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

バージョン 1.19.3

発売日 2023 年 6 月 12 日

機能強化

- `snapshot` コマンドを `cnpg` プラグインに追加して、Kubernetes `VolumeSnapshot` 標準リソースを使用してスタンバイからクラスターの一貫したコールドバックアップを作成します#1960
- `.spec.bootstrap.recovery.volumeSnapshot` スタンザを介した CSI VolumeSnapshot リソースのセットからのリカバリの最初の実装#1960
- `pg_failover_slots` をマネージド拡張機能に追加します #2057
- ドキュメントとクイックスタートガイドの指示を更新した Grafana ダッシュボードを改善しました #1916
- `import` スタンザに `schemaOnly` オプションを導入して、1 つ以上の既存のデータベースから新しい Postgres クラスターをブートストラップするときにデータのエクスポートおよびインポートを回避します#2234
- インスタンスポッドのスケジュールを管理するための TopologySpreadConstraints のサポートを追加します#2202
- PgBouncer の `Pooler` に `PodMonitor` サポートを追加します#2034

- デフォルトの Kubernetes スケジューラーをオーバーライドするオプションを追加します#2013
- Pooler リソースのデプロイメント戦略の構成を許可します #1983
- デフォルトの PostgreSQL バージョンを 15.3 に更新します#2022
- デフォルトで PgBouncer 1.19 を使用する#2018

技術的な強化

- 更新された k8s 種類のテスト済みバージョン#2054
- 別のトランザクションを使用して、ロールの資格情報を調整します。このパッチの前は、オペレーターは、いずれかが失敗した場合、すべてのロールの同期を元に戻していました#2004
- クラスターの復元中にフェンシングが削除されることを確認します #1987
- ポッド削除時のロギングを改善します #2136

修正

- k3d でのセットアップを妨げる可能性がある k3d エンジンでのバインドされていない変数を修正しました#2157
- メトリックで正しい PG バージョンをレポートする #2126
- ドキュメントで正しい walStorage キーを使用する #2140
- オペレーターがインスタンスに接続できない場合に調整を停止し、このような場合に明確な診断を提供します。これは、ネットワークの問題が CloudNativePG の通常のオペレーションを妨げるケースを明確にするのに役立ちます#2145、#2233、および#2242

バージョン 1.19.2

発売日 2023 年 4 月 27 日

機能強化

- kubectl の cnpg プラグインの report コマンドの--logs オプションを改善して、可能な場合は以前のログも含めます #1811
- -any サービスがデフォルトで無効になりました #1755

セキュリティ

- ローカルファイルを介したオーバーライドを介して SeccompProfile のカスタマイズを有効にします #1827

修正

- サーバーが初めて起動される前に、initdb ブートストラップフェーズ中にユーザーが提供した PostgreSQL 構成を適用します#1858

バージョン 1.19.1

発売日 2023 年 3 月 20 日

機能強化

- デフォルトのバックアップターゲットポリシー#1602 のオーバーライドを許可します。以前は、すべてのバックアップとスケジュールされたバックアップがクラスターレベルのターゲットポリシーを使用していました
- debug クラスターのログレベルを `initdb` ジョブに拡張します#1503
- PgBouncer プーラーの IPv6 とカスタム `pg_hba` をサポート#1395
- 2 つの新しいメトリックとステータスの追加情報でバックアップの可観測性を強化します#1428
- インスタンスマネージャーからのドキュメント API 呼び出し #1641
- Helm を介して展開名を明確にする #1505
- `psql` コマンドを `kubectl` #1668 の `cnpg` プラグインに追加して、ユーザーはポッドデフォルトでプライマリとの `psql` セッションを開始できます。

技術的な強化

- 依存関係の追跡/更新に Renovate を採用する #1367、#1473
- オペレーターイメージにサポートされているすべてのアーキテクチャのバイナリを注入します#1513
- バックアップ名を使用してバックアップオブジェクトストアのリソースと一致する#1650 Barman 3.3 で導入された`--name` オプションを活用して、バックアップとオブジェクトストアの関連付けをより堅牢にします。

修正

- プローブのエラー処理でパニックを防止 #1716
- HTTP パッケージとコントローラのランタイムログが JSON 形式であることを確認します#1442
- 単一インスタンスのクラスターに WAL ストレージを追加します クラスター#1570
- バックアップコードをより堅牢にするためのさまざまな改善#1536、#1564、#1588、#1466、#1647
- `cnpg` プラグインの `status` コマンドで WAL アーカイブ情報を適切に表示する #1666
- `AdditionalPodAffinity` および `AdditionalPodAntiAffinity` が設定されていない場合でも、`nodeAffinity` が適用されるように保証#1663
- オンラインアップグレードフェーズ中にフェールオーバー遅延を導入する#1728 以前は、オンラインアップグレードプロセスが不必要にフェールオーバーロジックをトリガーする場合がありました。

バージョン 1.19.0

発売日 2023 年 2 月 14 日

重要なお知らせ

- PostgreSQL バージョン 10 は、EOL に達したため、サポートされません。バージョン 11 以降がサポートされています。できるだけ早く PostgreSQL 15 への移行を計画してください。 Importing Postgres databases を参照してください

PostgreSQL のメジャーオフラインアップグレードの詳細については、

特徴

- スタンバイからのバックアップ `prefer-standby` に設定すると実行されることを受け入れる `.spec.backup.target` オプションを導入し、最も調整されたレプリカから物理ベースのバックアップを取得します#1162
- 遅延フェイルオーバープライマリの異常が検出されると、フェイルオーバープロセスを遅延するための `failoverDelay` パラメーターを導入します #1366

機能強化

- Kubernetes の予測ボリュームのサポートを紹介します #1269
- PostgreSQL サーバープロセスをより詳細に制御するためのサポートカスタム環境変数を導入します #1275
- `kubectl` の `cnpg` プラグインに `backup` コマンドを導入して、クラスターの新しいベースバックアップを発行します#1348
- ユーザーが WAL ファイルを既存の Postgres クラスター上の専用ボリュームに移動できるようにすることにより、個別の WAL ボリューム機能のサポートを向上させます#1066
- `min_wal_size` 、`max_wal_size` 、`keep_wal_size` 、`wal_keep_segments` に相当する値、および専用ボリュームに保存できる WAL の最大数を含む Prometheus エクスポータの追加メトリックで WAL の可観測性を強化#1382
- `streaming_replica` ユーザーにデータベースコメントを追加します#1349
- GKE の Webhook でのファイアウォールの問題を文書化する #1364
- `recovery` の postgresql.conf に関するノートを追加#1211
- パッケージを使用してプラグインのインストールに関する手順を追加します#1357
- 各マイナーリリースでサポートされている Postgres バージョンを指定する#1355
- CloudNativePG の PVC グループの意味を明確にします#1344
- DigitalOcean S3 互換スペースの例を追加します #1289
- 新しいクラスター定義のデフォルトの PostgreSQL バージョンを 15.2 に更新します#1430
- アーキテクチャドキュメント#1432 で Kubernetes レイヤーを詳細にカバーしています

技術的な強化

- リリースブランチの毎日のエンドツーエンドスモークテストを追加しました#1235

修正

- streaming_replica ユーザーとしての CHECKPOINT の実行をスキップする #1408
- waitForWalArchiveWorking の接続エラーに対する耐性を高める #1399
- PVC ロールが常に一貫していることを確認します #1380
- pvcTemplate を使用するときに walStorage のサイズ変更を許可します #1315
- ExecCommand がタイムアウトに従っていることを確認します#1242
- Prometheus がインストールされていない場合は PodMonitor リコンサイルを回避する #1238
- 必要がない場合は PodMonitor の検索を回避します #1213

147.5.2 CloudNativePG 1.18 のリリースノート

CloudNativePG の 1.18 マイナーリリースのユーザーに表示される変更の履歴。

変更の完全なリストについては、 commits を参照してください。

GitHub のリリースブランチで。

バージョン 1.18.5

発売日 2023 年 6 月 12 日

> **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。

機能強化

- snapshot コマンドを cnpg プラグインに追加して、Kubernetes VolumeSnapshot 標準リソースを使用してスタンバイからクラスターの一貫したコールドバックアップを作成します#1960
- .spec.bootstrap.recovery.volumeSnapshot スタンザを介した CSI VolumeSnapshot リソースのセットからのリカバリの最初の実装#1960
- pg_failover_slots をマネージド拡張機能に追加します #2057
- ドキュメントとクイックスタートガイドの指示を更新した Grafana ダッシュボードを改善しました #1916
- import スタンザに schemaOnly オプションを導入して、1 つ以上の既存のデータベースから新しい Postgres クラスターをブートストラップするときにデータのエクスポートおよびインポートを回避します#2234

- インスタンスポッドのスケジュールを管理するための TopologySpreadConstraints のサポートを追加します#2202
- PgBouncer の `Pooler` に `PodMonitor` サポートを追加します#2034
- デフォルトの Kubernetes スケジューラーをオーバーライドするオプションを追加します#2013
- `Pooler` リソースのデプロイメント戦略の構成を許可します #1983
- デフォルトの PostgreSQL バージョンを 15.3 に更新します#2022
- デフォルトで PgBouncer 1.19 を使用する#2018

技術的な強化

- 更新された k8s 種類のテスト済みバージョン#2054
- 別のトランザクションを使用して、ロールの資格情報を調整します。このパッチの前は、オペレーターは、いずれかが失敗した場合、すべてのロールの同期を元に戻していました#2004
- クラスターの復元中にフェンシングが削除されることを確認します #1987
- ポッド削除時のロギングを改善します #2136

修正

- K3 でのセットアップを妨げる可能性がある k3s エンジンのバインドされていない変数を修正しました#2157
- メトリックで正しい PG バージョンをレポートする #2126
- ドキュメントで正しい walStorage キーを使用する #2140
- オペレーターがインスタンスに接続できない場合に調整を停止し、このような場合に明確な診断を提供します。これは、ネットワークの問題が CloudNativePG の通常のオペレーションを妨げるケースを明確にするのに役立ちます#2145、#2233、および#2242

バージョン 1.18.4

発売日 2023 年 4 月 27 日

重要: PostgreSQL 10 が 2022 年 11 月にサポート終了 EOL に達したため、CloudNativePG は PostgreSQL 10 のサポートを終了します。バージョン 11 以降がサポートされています。できるだけ早く PostgreSQL 15 への移行を計画してください。 Importing Postgres databases を参照してください

PostgreSQL のメジャーオフラインアップグレードの詳細については、

機能強化

- `kubectl` の `cnpg` プラグインの `report` コマンドの`--logs` オプションを改善して、可能な場合は以前のログも含めます #1811
- `-any` サービスがデフォルトで無効になりました #1755

セキュリティ

- ローカルファイルを介したオーバーライドを介して SeccompProfile のカスタマイズを有効にします #1827

修正

- サーバーが初めて起動される前に、initdb ブートストラップフェーズ中にユーザーが提供した PostgreSQL 構成を適用します#1858

バージョン 1.18.3

発売日 2023 年 3 月 20 日

機能強化

- debug クラスターのログレベルを initdb ジョブに拡張します#1503
- PgBouncer プーラーの IPv6 とカスタム pg_hba をサポート#1395
- 2 つの新しいメトリックとステータスの追加情報でバックアップの可観測性を強化します#1428
- インスタンスマネージャーからのドキュメント API 呼び出し #1641
- Helm を介して展開名を明確にする #1505
- psql コマンドを kubectl #1668 の cnpg プラグインに追加して、ユーザーはポッドデフォルトでプライマリとの psql セッションを開始できます。

技術的な強化

- 依存関係の追跡/更新に Renovate を採用する #1367、#1473
- オペレーターイメージにサポートされているすべてのアーキテクチャのバイナリを注入します#1513
- バックアップ名を使用してバックアップオブジェクトストアのリソースと一致する#1650 Barman 3.3 で導入された--name オプションを活用して、バックアップとオブジェクトストアの関連付けをより堅牢にします。

修正

- プローブのエラー処理でパニックを防止 #1716
- HTTP パッケージとコントローラのランタイムログが JSON 形式であることを確認します#1442
- 単一インスタンスのクラスターに WAL ストレージを追加します クラスター#1570
- バックアップコードをより堅牢にするためのさまざまな改善#1536、#1564、#1588、#1466、#1647
- cnpg プラグインの status コマンドで WAL アーカイブ情報を適切に表示する #1666
- AdditionalPodAffinity および AdditionalPodAntiAffinity が設定されていない場合でも、nodeAffinity が適用されるように保証#1663

バージョン 1.18.2

発売日 2023 年 2 月 14 日

機能強化

- Kubernetes の予測ボリュームのサポートを紹介します #1269
- PostgreSQL サーバープロセスをより詳細に制御するためのサポートカスタム環境変数を導入します #1275
- `kubectl` の `cnpg` プラグインに `backup` コマンドを導入して、クラスターの新しいベースバックアップを発行します#1348
- ユーザーが WAL ファイルを既存の Postgres クラスター上の専用ボリュームに移動できるようにすることにより、個別の WAL ボリューム機能のサポートを向上させます#1066
- `min_wal_size` 、`max_wal_size` 、`keep_wal_size` 、`wal_keep_segments` に相当する値、および専用ボリュームに保存できる WAL の最大数を含む Prometheus エクスポータの追加メトリックで WAL の可観測性を強化#1382
- `streaming_replica` ユーザーにデータベースコメントを追加します#1349
- GKE の Webhook でのファイアウォールの問題を文書化する #1364
- `recovery` の postgresql.conf に関するノートを追加#1211
- パッケージを使用してプラグインのインストールに関する手順を追加します#1357
- 各マイナーリリースでサポートされている Postgres バージョンを指定する#1355
- CloudNativePG の PVC グループの意味を明確にします#1344
- DigitalOcean S3 互換スペースの例を追加します #1289

技術的な強化

- リリースブランチの毎日のエンドツーエンドスモークテストを追加しました#1235

修正

- `streaming_replica` ユーザーとしての `CHECKPOINT` の実行をスキップする #1408
- `waitForWalArchiveWorking` の接続エラーに対する耐性を高める #1399
- PVC ロールが常に一貫していることを確認します #1380
- `pvcTemplate` を使用するときに `walStorage` のサイズ変更を許可します #1315
- `ExecCommand` がタイムアウトに従っていることを確認します#1242
- Prometheus がインストールされていない場合は `PodMonitor` リコンサイルを回避する #1238
- 必要がない場合は `PodMonitor` の検索を回避します #1213

バージョン 1.18.1

発売日 2022 年 12 月 21 日

重要なお知らせ

- CNPG クラスターを識別するラベルとしての `postgresql` の差し迫った非推奨に関するアラート。このラベルを使用したリモートケースでは、代わりに `cnpg.io/cluster` ラベルの使用を開始してください #1130
- Armando Ruocco @armru を、一貫したインパクトのある貢献により新しい CloudNativePG メンテナーとして表彰#1167
- ARMv7 サポートの削除 #1092
- 1.16 の最終パッチリリース 1.16.5。リリース 1.16 はサポート終了となります。

機能強化

- サービスアカウントのラベルと注釈をカスタマイズする例、IRSA を介した GKE または EKS の ID 管理を介した認証を簡単にするために使用できるサービスアカウントテンプレートを追加します#1105
- `nodeAffinity` サポートを追加します#1182-より豊富なスケジュールオプションが可能になります
- Istio との互換性の向上 Istio の quit エンドポイントのサポートを追加して、Istio サイドカーを使用したジョブが無期限に実行されないように#967
- JSON ログのフィールドのリマッピングを許可します `level` および `ts` フィールドが既存のロギングを妨げる可能性があるユースケースに役立ちます#843
- `fio` コマンドを kubectl-cnpg プラグインに追加します #1097
- kubectl-cnpg プラグインの rpm/deb パッケージを追加します#1008
- 新しいクラスター定義のデフォルトの PostgreSQL バージョンを 15.1 に更新します#908
- Documentation
- CNPG サンドボックスへの参照を削除します#1120-クイックスタートドキュメントでのモニタリングに関する指示を優先して、CNPG サンドボックスは非推奨になりました
- “Release upgrade”ディスカッションへのリンク#1148-リリース更新ディスカッションがデフォルトになりますリリースの発表とディスカッションのためのチャンネル
- GOVERNANCE.md のメンテナーの名誉ステータスを文書#1033 - メンテナーがコントリビューションを続行する準備ができていない場合に続行する方法を説明します
- プルリクエストの作成手順の改善#1132
- 緊急バックアップのトラブルシューティング指示# 1184)
- アーキテクチャドキュメント#1432 で Kubernetes レイヤーを詳細にカバーしています

修正

- ブートストラップの PGDATA 権限が適切に 750 に設定されていることを確認します#1164
- クラスターをスケールダウンするときに WAL を含む PVC が削除されるように保証#1135

- `--dry-run` を使用する場合、pgbench マニフェストで欠落している ApiVersion と Kind を修正します #1088
- シークレットとサービスが見つからない場合にのみ作成するようにします#1145
- 復元時に設定済みの pg-wal を尊重する #1216
- nodeToClusters マップからレプリカをフィルタリングする #1194

技術的な強化

- テストの概要に `ciclops` を使用する #1064 ciclops GitHub アクションに依存して、E2E スイートの概要を提供し、そのプロジェクトから改善を継承します
- バックポートプル要求ワークフローを追加します #965 - 注釈が付けられている場合、リリースブランチにパッチを自動的にバックポートします
- e2e テストスイートでオペレーターログレベルを構成可能にする #1094
- ラベルに基づいてテストの実行を有効にする #951
- Go バージョンを 1.18 から 1.19 に更新します#1166

バージョン 1.18.0

発売日 2022 年 11 月 10 日

特徴

- **高可用性のためのクラスター管理物理レプリケーションスロット**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#740
- **Postgres クラスターハイバーネーション** 新しいサブコマンド `kubectl cnpg hibernate on/off/status <cluster-name>` を使用して、プラグインを介してクラスターハイバーネーションを導入します。ハイバネーションは、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壊します#782

セキュリティ

- `SeccomProfile` をポッドとコンテナに追加します #888

機能強化

- `pvcTemplate` でサイズ要求がある場合、クラスター仕様でストレージサイズを省略できるようにしました#914
- `cnpg` プラグインの `status` コマンド
- レプリケーションスロット情報を追加します #873
- フェンスされたクラスターの表示を明確にします #886
- レプリカクラスターの表示を改善します #871
- ドキュメント

- モニタリングページを改善し、プロメテウスとグラファナを使用してローカルシステムで CloudNativePG の可観測性機能を評価する方法に関する手順を提供します#968
- カスタムコントローラのデザイン理由に関するページを追加します #918
- 最後までの更新- to エンドテストスイートページ #945
- cnpg プラグインの新しいサブコマンド
  - pgbench は、クラスターに対して pgbench を実行するジョブ定義を生成します#958
  - install は、オペレーターのインストールマニフェストを生成します#944
- PostgreSQL 15.0 を新しいデフォルトバージョンとして設定します #821

修正

- plpgsql ファンクションを使用したデータベースのインポート #974
- ポイントインタイムリカバリーを実行するときに最も近いバックアップを適切に見つけます#949
- ScheduledBackup 形式が Kubernetes CronJob 形式に従っていないことを明確にします #883
- 古い可能性がある Kubernetes ポッドの準備状況ではなく、インスタンスマネージャーからの Postgres 情報に基づいてフェイルオーバーロジックをベースにします#890
- 新しく作成されるクラスターごとにアーカイブする WAL があることを確認します。これがないため、バックアップが機能しない可能性があります#897
- ドキュメントの barmanObjectStore の YAML キー名を修正しました #877
- krew リリースを修正#866

147.5.3 CloudNativePG 1.17 のリリースノート

CloudNativePG の 1.17 マイナーリリースのユーザーに表示される変更の履歴。

変更の完全なリストについては、 commits を参照してください。

GitHub のリリースブランチで。

バージョン 1.17.5

発売日 2023 年 3 月 20 日

> **警告:** これは、1.17.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。

機能強化

- debug クラスターのログレベルを initdb ジョブに拡張します#1503

- PgBouncer プーラーの IPv6 とカスタム pg_hba をサポート#1395
- インスタンスマネージャーからのドキュメント API 呼び出し #1641
- Helm を介して展開名を明確にする #1505
- psql コマンドを kubectl #1668 の cnpg プラグインに追加して、ユーザーはポッドデフォルトでプライマリとの psql セッションを開始できます。

技術的な強化

- 依存関係の追跡/更新に Renovate を採用する #1367、#1473

修正

- プローブのエラー処理でパニックを防止 #1716
- cnpg プラグインの status コマンドで WAL アーカイブ情報を適切に表示する #1666

バージョン 1.17.4

発売日 2023 年 2 月 14 日

特徴

- Kubernetes の予測ボリュームのサポート #1269
- PostgreSQL サーバープロセスをより詳細に制御するためのカスタム環境変数をサポート#1275

機能強化

- kubectl の cnpg プラグインに backup コマンドを導入して、クラスターの新しいベースバックアップを発行します#1348
- streaming_replica ユーザーにデータベースコメントを追加します#1349
- GKE の Webhook でのファイアウォールの問題を文書化する #1364
- recovery の postgresql.conf に関するノートを追加#1211
- パッケージを使用してプラグインのインストールに関する手順を追加します#1357
- 各マイナーリリースでサポートされている Postgres バージョンを指定する#1355
- CloudNativePG の PVC グループの意味を明確にします#1344
- DigitalOcean S3 互換スペースの例を追加します #1289

技術的な強化

- リリースブランチの毎日のエンドツーエンドスモークテストを追加しました#1235

修正

- PVC ロールが常に一貫していることを確認します #1380
- pvcTemplate を使用するときに walStorage のサイズ変更を許可します #1315
- Prometheus がインストールされていない場合は PodMonitor リコンサイルを回避する #1238

- 必要がない場合は PodMonitor の検索を回避します #1213

バージョン 1.17.3

発売日 2022 年 12 月 21 日

重要なお知らせ

- Armando Ruocco @armru を、一貫したインパクトのある貢献により新しい CloudNativePG メンテナーとして表彰#1167
- ARMv7 サポートの削除 #1092
- 1.16 の最終パッチリリース 1.16.5。リリース 1.16 はサポート終了となります。

機能強化

- Istio との互換性の向上 Istio の quit エンドポイントのサポートを追加して、Istio サイドカーを使用したジョブが無期限に実行されないように#967
- kubectl-cnpg プラグインの rpm/deb パッケージを追加します#1008
- 新しいクラスター定義のデフォルトの PostgreSQL バージョンを 15.1 に更新します#908
- Documentation
- CNPG サンドボックスへの参照を削除します#1120-クイックスタートドキュメントでのモニタリングに関する指示を優先して、CNPG サンドボックスは非推奨になりました
- “Release upgrade”ディスカッションへのリンク#1148-リリース更新ディスカッションがデフォルトになりますリリースの発表とディスカッションのためのチャンネル
- GOVERNANCE.md のメンテナーの名誉ステータスを文書#1033 - メンテナーがコントリビューションを続行する準備ができていない場合に続行する方法を説明します
- プルリクエストの作成手順の改善#1132
- 緊急バックアップのトラブルシューティング指示# 1184)
- アーキテクチャドキュメント#1432 で Kubernetes レイヤーを詳細にカバーしています

修正

- ブートストラップの PGDATA 権限が適切に 750 に設定されていることを確認します#1164
- クラスターをスケールダウンするときに WAL を含む PVC が削除されるように保証#1135
- シークレットとサービスが見つからない場合にのみ作成するようにします#1145
- 復元時に設定済みの pg-wal を尊重する #1216
- nodeToClusters マップからレプリカをフィルタリングする #1194

技術的な強化

- テストの概要に ciclops を使用する #1064 ciclops GitHub アクションに依存して、E2E スイートの概要を提供し、そのプロジェクトから改善を継承します

- バックポートプル要求ワークフローを追加します #965 - 注釈が付けられている場合、リリースブランチにパッチを自動的にバックポートします
- e2e テストスイートでオペレーターログレベルを構成可能にする #1094
- ラベルに基づいてテストの実行を有効にする #951
- Go バージョンを 1.18 から 1.19 に更新します#1166

バージョン 1.17.2

発売日 2022 年 11 月 10 日

セキュリティ

- `SeccomProfile` をポッドとコンテナに追加します #888

機能強化

- `cnpg` プラグインの `status` コマンド
- フェンスされたクラスターの表示を明確にします #886
- レプリカクラスターの表示を改善します #871
- ドキュメント
- モニタリングページを改善し、プロメテウスとグラファナを使用してローカルシステムで CloudNativePG の可観測性機能を評価する方法に関する手順を提供します#968
- カスタムコントローラのデザイン理由に関するページを追加します #918
- PostgreSQL 15.0 を新しいデフォルトバージョンとして設定します #821

修正

- `plpgsql` ファンクションを使用したデータベースのインポート #974
- ポイントインタイムリカバリーを実行するときに最も近いバックアップを適切に見つけます#949
- `ScheduledBackup` 形式が Kubernetes `CronJob` 形式に従っていないことを明確にします #883
- 古い可能性がある Kubernetes ポッドの準備状況ではなく、インスタンスマネージャーからの Postgres 情報に基づいてフェイルオーバーロジックをベースにします#890
- 新しく作成されるクラスターごとにアーカイブする WAL があることを確認します。これがないため、バックアップが機能しない可能性があります#897
- ドキュメントの `barmanObjectStore` の YAML キー名を修正しました #877
- `krew` リリースを修正#866

バージョン 1.17.1

発売日 2022 年 10 月 6 日

機能強化

- 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#759
- 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。正常にアーカイブされました

セキュリティ

- `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#754

修正

- 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#737
- ネイティブ Prometheus エクスポーターでの `MAPPEDMETRIC` および `DURATION` メトリックタイプの変換を尊重します #765
- 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確認します#741

バージョン 1.17.0

リリース日 2022 年 9 月 6 日マイナーリリース

特徴

- **WAL ファイルの別のボリューム**書き込み先行ログ WAL とデータベースデータファイルを別のディスクに分離することのサポート。データディレクトリの I/O 負荷を軽減することにより、大量の書き込みシステムのパフォーマンスの向上につながる可能性があります。このオプションは、メインで必須の `storage` セクション#513 で定義された `PGDATA` とは別の、専用ボリュームで WAL ファイル `pg_wal` を分離するオプションの `walStorage` セクションの導入で制御されます。現在の制限 `walStorage` は、クラスターの作成時にのみ設定でき、クラスターが稼働しているときに追加または削除することはできません。

機能強化

- PgBouncer 接続プーラー実装での下位ネットワーク TCP 設定の構成を有効にする#584
- `cnpg.io/instanceName` および `cnpg.io/podRole` ラベルが常にポッドと PVC に存在することを確認します#632 および#680
- インスタンスの `role` ラベルを基になる PVC に伝播する #634
- インスタンスと関連するすべての PVC の削除に役立つ `kubectl cnpg destroy` コマンドの導入 #643

修正

- 新しいクラスターのプライマリをブートストラップするときに shared_preload_libraries を使用する #642
- 別のプロセスが進行中であるかどうかをアトミックにチェックすることにより、オペレーターの複数のインプレースアップグレードプロセスが同時に実行されないようにします#655
- 新しくアップロードされたインスタンスマネージャーを保存するためのハードコードされたファイル名の使用を避け、オペレーターのオンラインアップグレード中に発生する可能性のある競合状態を防止します。#660
- GetAllAccessibleDatabases 呼び出し時にパニックが発生しないようにします#641

147.5.4 CloudNativePG 1.16 のリリースノート

CloudNativePG の 1.16 マイナーリリースのユーザーに表示される変更の履歴。

変更の完全なリストについては、 commits を参照してください。

GitHub のリリースブランチで。

バージョン 1.16.5

発売日 2022 年 12 月 21 日

> **警告:** これは、1.16.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。

重要なお知らせ

- Armando Ruocco @armru を、一貫したインパクトのある貢献により新しい CloudNativePG メンテナーとして表彰#1167
- ARMv7 サポートの削除 #1092
- 1.16 の最終パッチリリース 1.16.5。リリース 1.16 はサポート終了となります。

機能強化

- kubectl-cnpg プラグインの rpm/deb パッケージを追加します#1008
- 新しいクラスター定義のデフォルトの PostgreSQL バージョンを 15.1 に更新します#908
- Documentation
- CNPG サンドボックスへの参照を削除します#1120-クイックスタートドキュメントでのモニタリングに関する指示を優先して、CNPG サンドボックスは非推奨になりました
- “Release upgrade”ディスカッションへのリンク#1148-リリース更新ディスカッションがデフォルトになりますリリースの発表とディスカッションのためのチャンネル

- GOVERNANCE.md のメンテナーの名誉ステータスを文書#1033 - メンテナーがコントリビューションを続行する準備ができていない場合に続行する方法を説明します
- プルリクエストの作成手順の改善#1132
- 緊急バックアップのトラブルシューティング指示# 1184)

修正

- ブートストラップの PGDATA 権限が適切に 750 に設定されていることを確認します#1164
- シークレットとサービスが見つからない場合にのみ作成するようにします#1145
- 復元時に設定済みの pg-wal を尊重する #1216
- nodeToClusters マップからレプリカをフィルタリングする #1194

技術的な強化

- テストの概要に `ciclops` を使用する #1064 ciclops GitHub アクションに依存して、E2E スイートの概要を提供し、そのプロジェクトから改善を継承します
- バックポートプル要求ワークフローを追加します #965 - 注釈が付けられている場合、リリースブランチにパッチを自動的にバックポートします
- e2e テストスイートでオペレーターログレベルを構成可能にする #1094
- ラベルに基づいてテストの実行を有効にする #951
- Go バージョンを 1.18 から 1.19 に更新します#1166

バージョン 1.16.4

発売日 2022 年 11 月 10 日

セキュリティ

- `SeccomProfile` をポッドとコンテナに追加します #888

機能強化

- `cnpg` プラグインの `status` コマンド
- フェンスされたクラスターの表示を明確にします #886
- レプリカクラスターの表示を改善します #871
- ドキュメント
- モニタリングページを改善し、プロメテウスとグラファナを使用してローカルシステムで CloudNativePG の可観測性機能を評価する方法に関する手順を提供します#968
- カスタムコントローラのデザイン理由に関するページを追加します #918

修正

- `plpgsql` ファンクションを使用したデータベースのインポート #974

- ポイントインタイムリカバリーを実行するときに最も近いバックアップを適切に見つけます#949
- ScheduledBackup 形式が Kubernetes CronJob 形式に従っていないことを明確にします #883
- 古い可能性がある Kubernetes ポッドの準備状況ではなく、インスタンスマネージャーからの Postgres 情報に基づいてフェイルオーバーロジックをベースにします#890
- 新しく作成されるクラスターごとにアーカイブする WAL があることを確認します。これがないため、バックアップが機能しない可能性があります#897
- ドキュメントの barmanObjectStore の YAML キー名を修正しました #877
- krew リリースを修正#866

バージョン 1.16.3

発売日 2022 年 10 月 6 日

機能強化

- 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#759
- 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。正常にアーカイブされました

セキュリティ

- 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#754

修正

- 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#737
- ネイティブ Prometheus エクスポーターでの MAPPEDMETRIC および DURATION メトリックタイプの変換を尊重します#765

バージョン 1.16.2

発売日 2022 年 9 月 6 日

機能強化

- PgBouncer 接続プーラー実装での下位ネットワーク TCP 設定の構成を有効にする#584
- cnpg.io/instanceName および cnpg.io/podRole ラベルが常にポッドと PVC に存在することを確認します#632 および#680
- インスタンスの role ラベルを基になる PVC に伝播する #634

修正

- 新しいクラスターのプライマリをブートストラップするときに shared_preload_libraries を使用する #642
- 別のプロセスが進行中であるかどうかをアトミックにチェックすることにより、オペレーターの複数のインプレースアップグレードプロセスが同時に実行されないようにします#655
- 新しくアップロードされたインスタンスマネージャーを保存するためのハードコードされたファイル名の使用を避け、オペレーターのオンラインアップグレード中に発生する可能性のある競合状態を防止します。#660
- GetAllAccessibleDatabases 呼び出し時にパニックが発生しないようにします#641

バージョン 1.16.1

発売日 2022 年 8 月 12 日

機能強化

- PostgreSQL の huge_pages オプションの構成を有効にする#456
- 昇格と降格中の、フェールオーバーまたはスイッチオーバー後の、昇格の要求と書き込みの実際の可用性の間で経過時間を印刷することにより、ログを強化します#371
- クラスターステータスに PostgreSQL クラスターのタイムラインを導入する#462
- instanceName および clusterName ラベルをジョブ、ポッド、および PVC に追加して、これらのリソースとの相互作用を向上させます#534
- PostGIS クラスターを作成する方法に関する説明を追加します #570

セキュリティ

- securityContext を Pooler 展開に明示的に割り当てます #485
- 読み取りタイムアウト値を内部 Web サーバーに追加して、Slowloris DDoS を防止します #437

修正

- 再起動 stopDelay とスイッチオーバー switchoverDelay は、以前に誤ってスワップされたため、正しい遅延を使用します。switchoverDelay が設定されず、デフォルト値 40000000 秒を使用する場合、無期限に再起動をブロックする可能性があるため、これは重要な修正です。#531
- クエリがエラーを返したときにメトリックコレクターがパニックを発生させないようにしました#396
- 未チェックのポインターを参照していた安全でないデバッグメッセージを削除し、場合によっては、ログレベルに関係なくセグメンテーションフォールトを発生させます#491
- クラスターにアノテーションがない場合、フェンシング時のパニックを防止#512
- TLS 証明書が変更されない場合は、CRD の更新を回避する #501
- CRD に証明書を挿入しながら競合を処理する #547
- データベースインポート

  - データベースインポートで pg_restore を実行しているときに postgres ユーザーを使用します #411
  - モノリスインポートケースで sslmode を明示的に設定してオリジン外部サーバーとの SSL 接続を制御する要件を文書化します#572
  - 妨げたバグを修正 connectionParameters で dbname が指定された場合のインポート作業からのインポート #569
- バックアップとリカバリー - Google Cloud でオブジェクトストアの資格情報を正しく渡す #454

マイナーな変更

- デフォルトのオペランドイメージを PostgreSQL 15.0 に設定します

バージョン 1.16.0

リリース日 2022 年 7 月 7 日マイナーリリース

特徴

- **PostgreSQL のオフラインデータのインポートとメジャーアップグレード** bootstrap.initdb.import セクションを導入して、PostgreSQL 論理バックアップコンセプトを使用して、新しい CloudNativePG クラスター内の既存の PostgreSQL インスタンス Kubernetes の外部でも、ネットワークを介してオブジェクトをインポートする方法を提供します pg_dump /pg_restore 同じ方法を使用して、新しいクラスターでメジャー PostgreSQL のアップグレードを実行できます。この機能では、2 種類のインポート microservice 新しいクラスターに 1 つのデータベースのみをインポートしますと monolith 既存のインスタンスから選択したデータベースとロールをインポートします。
- ラベルに基づいた同期レプリケーションのアンチアフィニティルールアベイラビリティーゾーンなど、プライマリが実行されているノードとは異なる特性を持つノードで同期レプリカが実行されていることを確認します

機能強化

- 1 つ以上のインスタンスがフェンスされている場合にフェイルオーバーを無効にする既存の制限を削除することにより、フェンシングを向上させます
- DROP EXTENSION IF EXISTS を不必要に実行する代わりに、カタログに拡張機能が存在するかどうかを確認することにより、自動拡張機能管理フレームワークを強化します
- スイッチオーバーとフェイルオーバー中のインスタンスマネージャーのログを改善します
- オブジェクトストアからリカバリするとき、または pg_basebackup を介してインスタンスのクローンを作成するときに、アプリケーションのデータベースの名前、その所有者、および関連シークレットの再定義を有効にしますこれはこれまで initdb ブートストラップでのみ可能でした
- バックアップとリカバリー
  - PostgreSQL 15
  - の将来のサポートには Barman > = 3.0.0 が必要です inheritFromAzureAD オプションを介して Barman クラウドバックアップの Azure AD ワークロード ID を有効にします b_tran_5

barmanObjectStore.s3Credentials.region を導入して、バックアップとリカバリーオブジェクトストアの両方に AWS のリージョン AWS_DEFAULT_REGION を定義します

- Kubernetes 1.24 のサポート

変更点

- デフォルトのオペランドイメージを PostgreSQL 15.0 に設定します
- バックアップと WAL アーカイブに独自の実装に依存するのではなく、Kubernetes API の条件を使用します

修正

- PostgreSQL サーバーの CSV ログパイプを開始する WithActiveInstance ファンクション内の初期化順序を修正し、クラスター初期化フェーズでの適切なロギングを保証します。これは、バックアップからのリカバリーのようなブートストラップ操作が失敗している場合に特に役立ちますこのパッチの前には、このようなログ標準出力チャンネルに送信されず、永久に失われました)
- ホットスタンバイの依存パラメーターが減少し、プライマリが既に再起動している場合、不必要なスイッチオーバーを回避します
- ALTER ROLE ステートメントでロール名を適切に引用符で囲む
- バックアップとリカバリー
  - 要求されたリカバリタイムスタンプとバックアップの開始とを比較していた PITR の最も近い Barman バックアップを検出するアルゴリズムを修正します b_tran_9 トランザクション ID、名前付けリストアポイント、または recoveryTarget セクションに backupID と呼ばれる新しいフィールドを提供することにより「即時」ターゲット
  - barman-cloud-wal-archive および barman-cloud-backup コマンドを呼び出す暗号化パラメーターを修正します
  - 現在のクラスター名と一致しないサーバー名を使用してバックアップオブジェクトストアからリカバリするときに barmanObjectStore.serverName オプションを無視するのをやめる
- cnpg プラグイン - ユーザーが指定したときにプラグインが-n パラメーターに準拠していることを確認します - status コマンドを修正して結果を並べ替え、出力のばらつきを削除します

147.5.5 CloudNativePG 1.15 のリリースノート

CloudNativePG の 1.15 マイナーリリースのユーザーに表示される変更の履歴。

変更の完全なリストについては、 commits を参照してください。

GitHub のリリースブランチで。

> **警告:** これは、1.15.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。

バージョン 1.15.5

発売日 2022 年 10 月 6 日

機能強化

- 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#759
- 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。正常にアーカイブされました

セキュリティ

- `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#754

修正

- クラスターの条件を `metav1.Conditions` 構造体と互換性のあるものにする#720
- 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#737
- ネイティブ Prometheus エクスポーターでの `MAPPEDMETRIC` および `DURATION` メトリックタイプの変換を尊重します #765

バージョン 1.15.4

発売日 2022 年 9 月 6 日

機能強化

- PgBouncer 接続プーラー実装での下位ネットワーク TCP 設定の構成を有効にする#584
- `cnpg.io/instanceName` および `cnpg.io/podRole` ラベルが常にポッドと PVC に存在することを確認します#632 および#680
- インスタンスの `role` ラベルを基になる PVC に伝播する #634

修正

- 別のプロセスが進行中であるかどうかをアトミックにチェックすることにより、オペレーターの複数のインプレースアップグレードプロセスが同時に実行されないようにします#655
- 新しくアップロードされたインスタンスマネージャーを保存するためのハードコードされたファイル名の使用を避け、オペレーターのオンラインアップグレード中に発生する可能性のある競合状態を防止します。#660
- `GetAllAccessibleDatabases` 呼び出し時にパニックが発生しないようにします#641

バージョン 1.15.3

発売日 2022 年 8 月 12 日

機能強化

- PostgreSQL の huge_pages オプションの構成を有効にする#456
- 昇格と降格中の、フェールオーバーまたはスイッチオーバー後の、昇格の要求と書き込みの実際の可用性の間で経過時間を印刷することにより、ログを強化します#371
- instanceName および clusterName ラベルをジョブ、ポッド、および PVC に追加して、これらのリソースとの相互作用を向上させます#534
- PostGIS クラスターを作成する方法に関する説明を追加します #570

セキュリティ

- securityContext を Pooler 展開に明示的に割り当てます #485
- 読み取りタイムアウト値を内部 Web サーバーに追加して、Slowloris DDoS を防止します #437

修正

- 再起動 stopDelay とスイッチオーバー switchoverDelay は、以前に誤ってスワップされたため、正しい遅延を使用します。switchoverDelay が設定されず、デフォルト値 40000000 秒を使用する場合、無期限に再起動をブロックする可能性があるため、これは重要な修正です。#531
- クエリがエラーを返したときにメトリックコレクターがパニックを発生させないようにしました#396
- 未チェックのポインターを参照していた安全でないデバッグメッセージを削除し、場合によっては、ログレベルに関係なくセグメンテーションフォールトを発生させます#491
- クラスターにアノテーションがない場合、フェンシング時のパニックを防止#512
- TLS 証明書が変更されない場合は、CRD の更新を回避する #501
- CRD に証明書を挿入しながら競合を処理する #547
- バックアップとリカバリー - Google Cloud でオブジェクトストアの資格情報を正しく渡す #454

マイナーな変更

- デフォルトのオペランドイメージを PostgreSQL 15.0 に設定します

バージョン 1.15.2

リリース日 2022 年 7 月 7 日パッチリリース

機能強化

- スイッチオーバーとフェイルオーバー中のインスタンスマネージャーのログを改善します
- バックアップとリカバリーでの PostgreSQL 15 の将来のサポートには Barman > = 3.0.0 が必要です

変更点

- デフォルトのオペランドイメージを PostgreSQL 15.0 に設定します

修正

- PostgreSQL サーバーの CSV ログパイプを開始する `WithActiveInstance` ファンクション内の初期化順序を修正し、クラスター初期化フェーズでの適切なロギングを保証します。これは、バックアップからのリカバリーのようなブートストラップ操作が失敗している場合に特に役立ちますこのパッチの前には、このようなログ標準出力チャンネルに送信されず、永久に失われました)
- ホットスタンバイの依存パラメーターが減少し、プライマリが既に再起動している場合、不必要なスイッチオーバーを回避します
- `ALTER ROLE` ステートメントでロール名を適切に引用符で囲む
- バックアップとリカバリー
  - 要求されたリカバリタイムスタンプとバックアップの開始とを比較していた PITR の最も近い Barman バックアップを検出するアルゴリズムを修正します b_tran_9 トランザクション ID、名前付けリストアポイント、または `recoveryTarget` セクションに `backupID` と呼ばれる新しいフィールドを提供することにより「即時」ターゲット
  - `barman-cloud-wal-archive` および `barman-cloud-backup` コマンドを呼び出す暗号化パラメーターを修正します
  - 現在のクラスター名と一致しないサーバー名を使用してバックアップオブジェクトストアからリカバリするときに `barmanObjectStore.serverName` オプションを無視するのをやめる
- cnpg プラグイン
  - プラグインが user
  - で指定されたときに-n パラメーターに準拠していることを確認します `status` コマンドを修正して結果をソートし、出力のばらつきを削除します

バージョン 1.15.1

リリース日 2022 年 5 月 27 日パッチリリース

マイナーな変更

- PostgreSQL の `archive_timeout` 設定の構成を有効にします。これは以前は固定パラメーターでしたデフォルトで 5 分に設定
- `scheduledBackup` リソースに `backupOwnerReference` と呼ばれる新しいフィールドを導入して、作成したバックアップリソースの所有権参照を設定します。使用可能な値は `none` デフォルト、`self` スケジュールされたバックアップオブジェクトが所有するオブジェクト、および `cluster` Postgres クラスターが所有オブジェクト)
- ネイティブ Prometheus エクスポーターに PostgreSQL 14 以降の `pg_stat_wal` メトリックの自動収集を導入します
- デフォルトのオペランドイメージを PostgreSQL 15.0 に設定します

修正

- postgres に関連する孤立したプロセスを強制終了してフェンシングを修正する
- WithActiveInstance ファンクション内の CSV ログパイプを有効にして、リカバリブートストラップジョブからログを収集し、トラブルシューティングフェーズに役立ちます
- 空ではないバックアップオブジェクトストアを使用した新しいクラスターのブートストラップを防止し、既存のバックアップを上書きするリスクを排除します
- recovery ブートストラップ方法では、既存のバックアップが上書きされないように、リカバリオブジェクトストアとバックアップオブジェクトストアが異なることを確認します
- バックアップの RBAC がまだ作成されていない場合、調整ループを再キューイングします
- バックアップに関する問題と、クラスター名プロパティの誤った仕様を修正します
- 複数のレプリカを使用した高可用性のオペレーターの展開の場合、オペレーターポッドが常に最新の証明書を持つように保証
- cnpg report operator コマンドを修正して、複数のレプリカを使用した高可用性のオペレーターの展開の場合を正しく処理する
- クラスターの inheritedMetadata ラベルとアノテーションの変更を、再起動を必要とせずにクラスターの関連リソースに適切に伝播します
- デフォルト値だけに依存するのではなく、オペレーターの展開で定義されたカスタム構成マップとシークレット名を正しく解析するように cnpg プラグインを修正
- mkdocs の minidocks/mkdocs イメージを使用して、ドキュメントのローカルビルドを修正

バージョン 1.15.0

発売日 2022 年 4 月 21 日

特徴

- **フェンシング** cnpg.io/fencedInstances アノテーションを介したクラスターまたは特定の PostgreSQL インスタンスのフェンシング機能の導入。空でない場合、クラスター内のスイッチオーバー/フェールオーバーを無効にします。フェンスされたインスタンスはシャットダウンされ、ポッドは検査と緊急事態に備えて準備ができていないと考えられて実行され続けます。
- **LDAP 認証** pg_hba.conf の LDAP Simple Bind および Search+Bind 構成オプションを Postgres クラスター仕様で宣言的に定義できるようにし、ldapbindpasswd などの機密オプションに Kubernetes シークレットのオプションで使用を有効にします
- primaryUpdateMethod オプションの導入、 switchover デフォルトと restart の値を受け入れ、教師なし primaryUpdateStrategy の場合に使用されます。このメソッドは、ローリング更新プロシージャ中にプライマリインスタンスに何が起こるかを制御します
- オペレーターと特定の Postgres クラスターの両方の診断とより効果的なトラブルシューティングを行うための kubectl cnp プラグインの新しい report コマンド
- バックアップオブジェクトストアにない Backup オブジェクトをプルーニング
- ポイントインタイムリカバリーブートストラップ方式でのターゲットタイムラインと LSN の仕様

- sessionToken オプションを介した AWS S3 での AWS_SESSION_TOKEN 認証トークンのサポート
- quay.io/enterprisedb/pgbouncer:1.17.0 に設定された Pooler ポッドの PgBouncer のデフォルトのイメージ名

修正

- 以前は、使用可能な最新のバックアップより前のターゲットが不可能であったため、targetTime を介したポイントインタイムリカバリーのベースバックアップ検出が正しく機能するようになりました。検出アルゴリズムは、開始点として最後のバックアップを選択することにより常に間違っていました。
- オペレーターが pg_controldata から収集した値に依存することにより、ホットスタンバイの影響を受けやすいパラメーターの復元力が向上しました
- 無効な証明書エラーをスローする代わりに、無効なエントリを適切に破棄することにより、中間証明書の処理が改善されました
- データベース内の Prometheus エクスポーターメトリック収集クエリーがロールバックの代わりにコミットされるようになりましたこれにより、ダウンストリームダッシュボードから可視されるロールバックトランザクションの数が変更される場合があります該当する場合

バージョン 1.15.0 は、CloudNativePG の最初のリリースです。以前は、このソフトウェアは EDB Cloud Native PostgreSQL 現在は EDB Postgres for Kubernetes と呼ばれていました。以前のリリースに関する情報をお探しの場合は、

> EDB documentation を参照してください。

第148章　Appendix A

148.1 付録 A-バックアップ用の共通オブジェクトストア

付録 *A*-バックアップ用の共通オブジェクトストア 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

- Amazon S3
- *Microsoft 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。

148.1.1 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 バックアップに関する限り、2 つの方法でバックアップを S3 バケットに保存する権限を定義できます。

- CloudNativePG 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.cnpg.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 を構成できます。

```
apiVersion: postgresql.cnpg.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 ワークロード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.cnpg.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.cnpg.io/v1
kind: Cluster
[...]
spec:
  backup:
    barmanObjectStore:
      destinationPath: "s3://[your-bucket-name]/[your-backup-folder]/"
      endpointURL: "https://sfo3.digitaloceanspaces.com"
      s3Credentials:
        [...]
```

重要: HTTPS 経由で MinIO を使用する場合など、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、 Barman が証明書を正しく検証できるように、CA バンドルを含むシークレットを参照するオプション *endpointCA* を設定する必要があります。

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 `cnpg.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 cnpg 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。

148.1.2 Azure Blob Storage

Azure Blob Storage は、Microsoft が提供する obect ストレージサービスです。

CloudNativePG 管理対象データベース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.cnpg.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。

```yaml
apiVersion: postgresql.cnpg.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:
          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。

148.1.3 Google Cloud Storage

現在、CloudNativePG オペレーターは、
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.cnpg.io/v1
kind: Cluster
[...]
spec:
  [...]
  backup:
    barmanObjectStore:
      destinationPath: "gs://<destination path here>"
      googleCredentials:
        gkeEnvironment: true

  serviceAccountTemplate:
    metadata:
      annotations:
        iam.gke.io/gcp-service-account:  [...].iam.gserviceaccount.com
        [...]
```

認証を使用する

instruction from Google をフォローする

認証に必要な情報をすべて含む JSON ファイルを取得します。

JSON ファイルのコンテンツは、次のコマンドで作成できる Secret を使用して提供する必要があります。

```
kubectl create secret generic backup-creds --from-file=gcsCredentials=gcs_credentials_
↪file.json
```

これにより、次のように yaml ファイルで使用される backup-creds という名前で Secret が作成されます。

```
apiVersion: postgresql.cnpg.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。

148.1.4 MinIO ゲートウェイ

オプションで、バックアップオブジェクトを S3 や GCS などの他のクラウドストレージソリューションに中継する共通インターフェイスとして MinIO ゲートウェイを使用できます。詳細については、
MinIO official documentation を参照してください。

具体的には、CloudNativePG クラスターは、以前に作成した資格情報とサービスを使用して、エンドポイントとしてローカル MinIO ゲートウェイを直接ポイントできます。

MinIO シークレットは、PostgreSQL クラスターと MinIO インスタンスの両方で使用されます。したがって、それらを同じ名前空間に作成する必要があります。

```
kubectl create secret generic minio-creds \
  --from-literal=MINIO_ACCESS_KEY=<minio access key here> \
  --from-literal=MINIO_SECRET_KEY=<minio secret key here>
```

注釈: この場合、クラウドオブジェクトストレージ資格情報は MinIO ゲートウェイによってのみ使用されます。

重要: PostgreSQL が MinIO ゲートウェイに到達できるようにオーダーには、MinIO ゲートウェイインスタンスにバインドされたポート `9000` に `ClusterIP` サービスを作成する必要があります。

例

```
apiVersion: v1
kind: Service
metadata:
  name: minio-gateway-service
spec:
  type: ClusterIP
  ports:
    - port: 9000
      targetPort: 9000
      protocol: TCP
  selector:
    app: minio
```

警告: このドキュメントの執筆時点では、公式の MinIO Operator

for Kubernetes は、ゲートウェイ機能をサポートしていません。そのため、代わりに `deployment` を使用します。

MinIO 展開は、クラウドストレージの資格情報を使用してオブジェクトをリモートバケットにアップロードし、バックアップファイルを別の場所に中継します。

次に、AWS S3 をクラウドオブジェクトストレージとして使用する例を示します。

```
apiVersion: apps/v1
kind: Deployment
[...]
spec:
  containers:
  - name: minio
    image: minio/minio:RELEASE.2020-06-03T22-13-49Z
    args:
    - gateway
    - s3
    env:
    # MinIO access key and secret key
```

(次のページに続く)

(前のページからの続き)

```
    - name: MINIO_ACCESS_KEY
      valueFrom:
        secretKeyRef:
          name: minio-creds
          key: MINIO_ACCESS_KEY
    - name: MINIO_SECRET_KEY
      valueFrom:
        secretKeyRef:
          name: minio-creds
          key: MINIO_SECRET_KEY
    # AWS credentials
    - name: AWS_ACCESS_KEY_ID
      valueFrom:
        secretKeyRef:
          name: aws-creds
          key: ACCESS_KEY_ID
    - name: AWS_SECRET_ACCESS_KEY
      valueFrom:
        secretKeyRef:
          name: aws-creds
          key: ACCESS_SECRET_KEY
#  Uncomment the below section if session token is required

#    - name: AWS_SESSION_TOKEN

#      valueFrom:

#        secretKeyRef:

#          name: aws-creds

#          key: ACCESS_SESSION_TOKEN

        ports:
        - containerPort: 9000
```

Cluster 定義の endpointURL として MinIO Gateway サービスを構成し、 BUCKET_NAME を置き換えるバケット名を選択します。

```
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
[...]
spec:
  backup:
    barmanObjectStore:
```

(次のページに続く)

(前のページからの続き)

```
    destinationPath: s3://BUCKET_NAME/
    endpointURL: http://minio-gateway-service:9000
    s3Credentials:
      accessKeyId:
        name: minio-creds
        key: MINIO_ACCESS_KEY
      secretAccessKey:
        name: minio-creds
        key: MINIO_SECRET_KEY
  [...]
```

バックアップを続行する前に、s3://BUCKET_NAME/ でアーカイブされた WAL ファイルの存在を確認します。