



EDB

Postgres® for the AI Generation

EDB CloudNativePG Global Cluster

リリース *1.1.1*

EnterpriseDB Corporation

2025 年 04 月 30 日

目次

第 1 章	Prerequisite and Release Note	3
1.1	EDB CloudNativePG Global Cluster	3
1.1.1	始める前に	3
1.1.2	サポートされている Kubernetes ディストリビューション	3
1.1.3	要件	3
1.1.4	API リファレンス	4
1.1.5	商標	4
1.2	EDB Postgres for Kubernetes Release notes	4
1.2.1	EDB Postgres Distributed for Kubernetes 1.1.1 release notes	4
1.2.2	EDB Postgres Distributed for Kubernetes 1.1.0 release notes	5
1.2.3	EDB Postgres Distributed for Kubernetes 1.0.1 release notes	6
1.2.4	EDB Postgres Distributed for Kubernetes 1.0.0 release notes	7
第 2 章	Getting Started	9
2.1	Before You Start	9
2.1.1	Kubernetes の用語	9
2.1.2	PGD 用語	10
2.1.3	クラウド用語	11
2.1.4	次に何をするか	11
2.2	Use cases	11
2.2.1	ケース 1Kubernetes 内のアプリケーション	11
2.2.2	ケース 2Kubernetes 外部のアプリケーション	12
2.3	Architecture	13
2.3.1	EDB Postgres Distributed との関係	13
2.3.2	Kubernetes の EDB Postgres との関係	13
2.3.3	Kubernetes 用に分散された EDB Postgres を使用した PGD の管理	14
2.3.4	Kubernetes での PGD の展開	15
2.4	Installation and upgrade	17
2.4.1	Kubernetes へのオペレーターのインストール	17
2.4.2	デプロイメントの詳細	20
2.4.3	PGD クラスターを展開する	20
2.4.4	アップグレード	23
2.5	Quick start	25
2.5.1	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	26
2.5.2	パート 2-Kubernetes 用に分散された EDB Postgres のインストール	26
2.5.3	パート 3-PGD クラスターを展開する	26
第 3 章	Configuration	31
3.1	Operator configuration	31
3.1.1	Kubernetes Operator 構成用に分散された EDB Postgres	31

3.1.2	Kubernetes Operator 構成の EDB Postgres	34
3.2	Examples of configuration	34
3.2.1	基本	34
3.2.2	バックアップとリストア	49
3.2.3	読み取りノードルーティング	64
3.2.4	LDAP	65
3.2.5	管理対象ロール	69
3.2.6	TDE	70
3.2.7	構成	72
第 4 章	Using	81
4.1	Managing EDB Postgres Distributed (PGD) databases	81
4.1.1	PGD CLI	81
4.1.2	データベースへのアクセス	82
4.2	Backup on object stores	84
4.2.1	共通オブジェクトストア	84
4.2.2	WAL アーカイブ	84
4.2.3	スケジュールされたバックアップ	85
4.2.4	リテンション ポリシー	87
4.2.5	圧縮アルゴリズム	88
4.2.6	バックアップオブジェクトのタグ付け	88
4.2.7	PGD ノードのオンデマンドバックアップ	88
4.3	Recovery	88
4.3.1	前提条件	89
4.3.2	オブジェクトストアからのリカバリー	89
4.3.3	オブジェクトストアからの PITR	90
4.3.4	backupID を指定したオブジェクトストアからのリカバリー	91
4.3.5	volumeSnapshot からのリカバリー	92
4.3.6	volumeSnapshot からの PITR	93
4.3.7	リカバリーターゲット	94
4.3.8	複数のリージョンでの PGD グループのリカバリと作成	94
4.4	Security	95
4.4.1	コード	95
4.4.2	コンテナ	95
4.4.3	クラスター	96
4.5	Connectivity	99
4.5.1	サービス	100
4.5.2	ドメイン名解決	102
4.5.3	TLS 構成	102
4.5.4	アプリケーションからの接続	103
4.6	Certificates	104
4.6.1	サーバー証明書の構成	104
4.6.2	クライアント証明書の構成	106
4.6.3	TLS モード	107
4.6.4	自己署名の例	107
4.7	Client TLS/SSL connections	109
4.8	Declarative pausing and resuming	109

4.9	EDB's private container registries	110
4.9.1	どのリポジトリを選択しますか？	111
4.9.2	トークンの取得方法	111
4.9.3	トライアル	112
4.9.4	オペランドイメージ	112
4.9.5	オペランド画像のカスタマイズ	114
4.10	Predefined labels	116
4.10.1	事前定義されたアノテーション	117
4.11	Managed configuration	117
4.12	Monitoring	118
4.12.1	モニタリングインスタンス	118
4.12.2	モニタリングオペレーター	118
4.12.3	Openshift でモニタ	119
4.13	PGDGroup parting	119
4.13.1	削除とファイナライザー	119
4.13.2	制限時間	119
4.13.3	ファイナライザーをスキップ	119
4.13.4	PGD グループのクリーンアップ	120
4.14	SQLMutations	120
4.14.1	SQL ミューテーションタイプ	121
4.15	Red Hat OpenShift	122
4.15.1	EDB プライベートレジストリへのアクセスの保証	122
4.15.2	オペレーターのインストール	123
4.15.3	PGD クラスターの作成	124
4.15.4	チャンネル	133
4.15.5	依存関係	133
4.16	Transparent data encryption (TDE)	134
4.17	LDAP authentication	136
4.18	API Reference	137
4.18.1	Resource Types	137
4.18.2	CertificateKeystores	137
4.18.3	CertificatePrivateKey	138
4.18.4	CertificateSpec	138
4.18.5	ConditionStatus	140
4.18.6	JKSKeystore	140
4.18.7	KeyUsage	141
4.18.8	LocalObjectReference	141
4.18.9	ObjectReference	141
4.18.10	PKCS12Keystore	142
4.18.11	PrivateKeyAlgorithm	142
4.18.12	PrivateKeyEncoding	143
4.18.13	PrivateKeyRotationPolicy	143
4.18.14	SecretKeySelector	143
4.18.15	X509Subject	143
4.18.16	ClusterImageCatalog	144
4.18.17	ImageCatalog	144
4.18.18	PGDGroup	145

4.18.19 PGDGroupCleanup	145
4.18.20 Backup	145
4.18.21 BackupStatus	146
4.18.22 CNPStatus	146
4.18.23 CatalogImage	147
4.18.24 CertManagerTemplate	148
4.18.25 ClientCertConfiguration	148
4.18.26 ClientPreProvisionedCertificates	148
4.18.27 ClusterStatus	149
4.18.28 CnpBaseConfiguration	149
4.18.29 CnpConfiguration	152
4.18.30 ConnectionString	152
4.18.31 ConnectivityConfiguration	153
4.18.32 ConnectivityStatus	153
4.18.33 DNSConfiguration	154
4.18.34 DiscoveryJobConfig	154
4.18.35 ImageCatalogRef	155
4.18.36 ImageCatalogSpec	155
4.18.37 ImageStatus	155
4.18.38 InheritedMetadata	156
4.18.39 JoinMethod	156
4.18.40 Metadata	156
4.18.41 NodeCertificateStatus	157
4.18.42 NodeKindName	157
4.18.43 NodeSummary	157
4.18.44 NodesExtensionsStatus	158
4.18.45 OTELConfiguration	158
4.18.46 OTELTLSConfiguration	158
4.18.47 OperatorPhase	159
4.18.48 OperatorPhaseCleanup	159
4.18.49 PGDGroupCleanupSpec	159
4.18.50 PGDGroupCleanupStatus	160
4.18.51 PGDGroupSpec	160
4.18.52 PGDGroupStatus	161
4.18.53 PGDNodeGroupEntry	162
4.18.54 PGDNodeGroupSettings	163
4.18.55 PGDProxyConfiguration	163
4.18.56 PGDProxyEntry	164
4.18.57 PGDProxySettings	165
4.18.58 PGDProxyStatus	165
4.18.59 PGDRaftStatus	166
4.18.60 PGDStatus	166
4.18.61 ParentGroupConfiguration	167
4.18.62 PauseStatus	167
4.18.63 PgdConfiguration	168
4.18.64 PhaseType	169
4.18.65 PreProvisionedCertificate	169

4.18.66	ReconciliationStage	169
4.18.67	RecoverabilityPointsByMethod	169
4.18.68	ReplicationCertificateStatus	170
4.18.69	Restore	170
4.18.70	RestoreStatus	171
4.18.71	RootDNSConfiguration	171
4.18.72	SQLMutation	171
4.18.73	SQLMutationType	172
4.18.74	SQLMutations	172
4.18.75	ScheduledBackupSpec	172
4.18.76	ScheduledBackupStatus	173
4.18.77	ServerCertConfiguration	174
4.18.78	ServiceTemplate	174
4.18.79	ServiceUpdateStrategy	174
4.18.80	TLSConfiguration	175
4.18.81	TLSMode	175
4.18.82	VolumeSnapshotRestoreStatus	175
4.18.83	VolumeSnapshotsConfiguration	176
4.19	Supported versions	176
4.19.1	命名スキーム	176
4.19.2	一般的なサポートステータス	176
4.19.3	詳細なサポートステータス	177
4.19.4	Openshift Platform でサポートされているバージョン	177
4.20	Known issues and limitations	178
4.20.1	Postgres メジャーバージョンのアップグレード	178
4.20.2	データの移行	178
4.20.3	PgBouncer との接続	178
4.20.4	バックアップ操作	178
4.20.5	EDB Postgres Distributed の既知の問題と制限	179
4.21	Image catalog	179
4.21.1	名前空間の例 ImageCatalog	180
4.21.2	ClusterImageCatalog リソースを使用したクラスター全体のカタログの例	180
4.22	Join method	181
4.22.1	初期グループ	181
4.22.2	ノード参加の参加方法を構成する	182
4.22.3	グループ参加の参加方法を構成する	183

Kubernetes の EDB Postgres Distributed PGD4K は、PGD プロキシによってルーティングされるトラフィックを使用して、Kubernetes で EDB Postgres 分散 PGD ワークロードを管理するように設計されたオペレーターです。

オペレーターが提供するメインのカスタムリソースは、PGDGroup と呼ばれます。

アーキテクチャは、異なる Kubernetes クラスターにまたがって展開することもできます。

第1章 Prerequisite and Release Note

1.1 EDB CloudNativePG Global Cluster

1.1.1 始める前に

EDB Postgres Distributed for Kubernetes は、Kubernetes 環境に EDB Postgres Distributed を展開する方法を提供します。したがって、[EDB Postgres Distributed documentation](#) を読むことをお勧めします。

Kubernetes 用 EDB Postgres Distributed での作業を開始するには、PGD ドキュメントの以下を読んでください。

- [Terminology](#)
- [PGD overview](#)
- [Choosing your architecture](#)
- [Choosing a Postgres distribution](#)

高度な使用法と最大限のカスタマイズのためには、[Architecture](#) で説明しているように、[EDB Postgres for Kubernetes documentation](#) に慣れることも重要です。

1.1.2 サポートされている Kubernetes ディストリビューション

Kubernetes の EDB Postgres Distributed は、次の場合に利用できます。

- Helm チャートを介した Kubernetes バージョン 1.23 以降
- Red Hat OpenShift 認定オペレーターを介した Red Hat OpenShift バージョン 4.10 以降のみ

1.1.3 要件

Kubernetes 用 EDB Postgres 分散では、分散 PGD クラスターをホスティングする Kubernetes / OpenShift クラスターが、以下に対応するようにお客様が準備したものである必要があります。

- PGD グローバルグループが分散されているすべての Kubernetes クラスターを含む公開キーインフラストラクチャ PKI。mTLS は、メッシュトポロジ内のすべてのノードを認証および承認し、暗号化された通信を保証するために必要です。
- PGD グローバルグループに関係するすべての Kubernetes クラスターのネットワーキングインフラストラクチャで、各ノードが相互に通信できるようにする

EDB Postgres Distributed for Kubernetes には、Cert Manager 1.10 以降も必要です。

参考:

詳細については、[Connectivity](#) を参照してください。

1.1.4 API リファレンス

EDB Postgres Distributed for Kubernetes が提供するリソースのリストについては、[API リファレンス](#) を参照してください。

1.1.5 商標

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されています。

1.2 EDB Postgres for Kubernetes Release notes

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

Version	Release date
<i>Bootstrap</i>	Mar 18 2025
<i>Bootstrap</i>	Dec 25 2024
<i>Bootstrap</i>	14 Oct 2024
<i>Bootstrap</i>	24 Apr 2024

1.2.1 EDB Postgres Distributed for Kubernetes 1.1.1 release notes

リリース日 2025 年 3 月 18 日

機能

- **新しい PGD バージョンのサポート** PGD バージョン 5.7 がサポートされるようになりました。
- **新しい PG4K オペレーターバージョンのサポート** PG4K オペレーター v1.25.1 がサポートされるようになりました。

Component	Description
PGD4K	ローカルグループへの PGD ノードの物理的参加のサポートを追加しました。
PGD4K	リモートグループへの PGD ノードの物理的参加のサポートを追加しました。
PGD4K	PGD オペレーターの ImageCatalog サポートを追加しました。
PGD4K	`writelead` ノードにパッチを適用する前に `writeLead` 転送が必要かどうかを判断する検出が改善されました。
PGD4K	'bdr' を PG 追加ライブラリに 1 回のみ追加できるようにした。
PGD4K	プロキシポッドの pgd-proxy および pgd-cli の共有構成ファイルを追加しました。

サポートされているバージョン

- Kubernetes 1.32、1.31、1.30、および 1.29
- PostgreSQL 17、16、15、14、13
- PG4K v1.22.8、v1.22.8、v1.25.0、v1.25.1

デフォルト画像

- PGD postgresql-pgd 17.4-5.7.0-1
- PGD プロキシ edb-pgd-proxy 5.7.0-1

1.2.2 EDB Postgres Distributed for Kubernetes 1.1.0 release notes

リリース日 2024 年 12 月 25 日

機能

Component	Description
PGD4K	新しい PG4K リリースのサポート バージョン 1.1.0 から、PGD4K オペレーターは、新しい PG4K 長期サポート LTS リリースバージョン 1.25 をサポートするようになりました。

[次のページに続く](#)

表 3 – 前のページからの続き

Component	Description
PGD4K	新しい PGD バージョンのサポート PGD バージョン 5.6.1 がサポートされるようになりました。
PGD4K	新しいオペレーターアーキテクチャのサポート linux/arm64 および linux/ppc64le アーキテクチャのサポートを追加します。
PGD4K	cnpg-i プラグインの構成のサポート。
PGD4K	PG4K クラスタで reconcilePodSpec が使用される場合、不必要な書き込みリード転送を回避します。
PGD4K	スケールダウン書き込みリーダー最後書き込みリーダーは、スケールダウンする最後の PGD ノードになります。
PGD4K	復元プロセスの強化 復元プロセスをコンポーネントに分割し、さまざまな種類の復元およびブートストラップ操作に個別のジョブを使用することにより、改善されました。
PGD4K	Webhook 検証の強化 restore.serverNames の有効性が Webhook を介してチェックされるようになりました。
PGD4K	管理対象クラスタのフェーズに関する情報の PGDGroup ステータスの一部として clusterStatus が含まれるようになりました。

1.2.3 EDB Postgres Distributed for Kubernetes 1.0.1 release notes

リリース日 2024 年 10 月 14 日

機能

Component	Description
PGD4K	オペレーターベースイメージが UBI9 にアップグレードされた
PGD4K	PGDGroup での LDAP 認証構成のサポート
PGD4K	PGDGroup でのテーブルスペース構成のサポート
PGD4K	PGDGroup での予測ボリュームテンプレート構成のサポート
PGD4K	プロキシサービスの作成による pgd の読み取りノードルーティングのサポート
PGD4K	VolumeSnapshot を使用した PGDGroup のバックアップと復元をサポート
PGD4K	さまざまなバックアップ方法に対応する PGDGroup の複数のスケジューラーをサポート
PGD4K	書き込みリードノードに対してのみ実行される新しいミューテーションタイプをサポートします。
PGD4K	PGDGroup 構成が変更されるときに不必要な書き込みリーダー転送を回避する
PGD4K	一時停止された PGDGroup のノードセクターと許容変更の処理
PGD4K	検出ジョブでの nodeSelector と tolerance を再利用し

次のページに続く

表 4 – 前のページからの続き

Component	Description
PGD4K	参加する前に適用されるサポートアプリケーションデータベース構成
PGD4K	トポロジ SpreadConstraints 構成のサポート

1.2.4 EDB Postgres Distributed for Kubernetes 1.0.0 release notes

リリース日 2024 年 4 月 24 日

これは、EDB Postgres 分散クラスターを展開および管理する Kubernetes オペレーターである Kubernetes の EDB Postgres Distributed の最初のメジャー安定リリースです。

Kubernetes 1.0.0 用に分散された EDB Postgres のハイライト

オペレーターは、API グループ `pgd.k8s.enterprisedb.io` に PGDGroup カスタムリソースを実装します。このリソースを使用して、Kubernetes 内に EDB Postgres 分散クラスターを作成および管理できます。次の機能を使用します。

- バージョン 5 以降での EDB Postgres Distributed クラスターの展開。
- Postgres Distributed の自己修復機能に加えて、障害が発生した PGD ノードの復旧や再起動など。
- アプリケーションが各 PGD グループの書き込みリーダーに接続できるようにするサービスを定義します。

注釈: EDB Postgres Distributed for Kubernetes オペレーターは、[EDB Postgres for Kubernetes PG4K](#) を活用し、そのプロジェクトの機能の多くを継承します。Kubernetes バージョン 1.0.0 の EDB Postgres Distributed は、特に PG4K のリリース 1.22 に基づいています。詳細については、[PG4K release notes](#) を参照してください。

機能

Component	Description
PGD4K	Kubernetes 内のバージョン 5 以降を使用した EDB Postgres Distributed クラスターの展開
PGD4K	障害が発生した PGD ノードのリカバリや再起動などの自己修復機能
PGD4K	アプリケーションを各 PGD グループの書き込みリーダーに接続できる定義されたサービス
PGD4K	Raft サブグループの実装
PGD4K	TLS 接続とクライアント証明書認証

次のページに続く

表 5 – 前のページからの続き

Component	Description
PGD4K	S3 互換オブジェクトストアへの継続的なバックアップ

第2章 Getting Started

2.1 Before You Start

開始する前に、Kubernetes と PGD に固有のいくつかの用語に精通することが重要です。

2.1.1 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、ポッド内で使用するパスワード、アクセスキー、またはトークンなどの少量の機密データを保存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 Kubernetes ソフトウェア拡張機能です。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。

kubect1

kubect1 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。

Kubernetes の EDB Postgres Distributed 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。詳細については、[Supported releases](#) を参照してください。

2.1.2 PGD 用語

詳細については、PGD ドキュメントの [Terminology](#) を参照してください。

Data node

PGD データベースインスタンス。

Failover

高可用性データベースクラスターの障害を認識し、アプリケーションを別のアクティブなデータベースに接続するアクションを実行する自動プロセス。

Switchover

アプリケーションとクラスター内のアクティブなデータベースノード間の接続の計画的な変更。通常はメンテナンスのために行われます。

Write leader

常時オンアーキテクチャでは、ノードはアプリケーションの正しい接続エンドポイントとして選択されます。このノードは書き込みリーダーと呼ばれます。書き込みリーダーは、データノードのクォーラムのコンセンサスによって選択されます。

2.1.3 クラウド用語

リージョンクラウド内のリージョンは、アベイラビリティ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 **アベイラビリティゾーン***ゾーン*としても知られるは、リソースを展開できるリージョンのエリアです。通常、アベイラビリティゾーンは、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。

2.1.4 次に何をするか

用語に慣れたので、選択したクラウド環境にオペレーターを展開する前に *test EDB Postgres Distributed for Kubernetes on your laptop using a local cluster* できます。

2.2 Use cases

Kubernetes の EDB Postgres Distributed 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 クラスターにあるアプリケーションと連携するように設計されました。

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、アプリケーションを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。

以下は、基本的な考慮事項の概要です。 *EDB Postgres for Kubernetes documentation* を参照してください
詳細については。

2.2.1 ケース 1Kubernetes 内のアプリケーション

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

アプリケーションは通常ステートレスであり、標準の展開として管理され、複数のレプリカがさまざまな 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 HTTPS を介してプロバイダーのロードバランサー機能を介してエンドユーザーに外部公開されます。

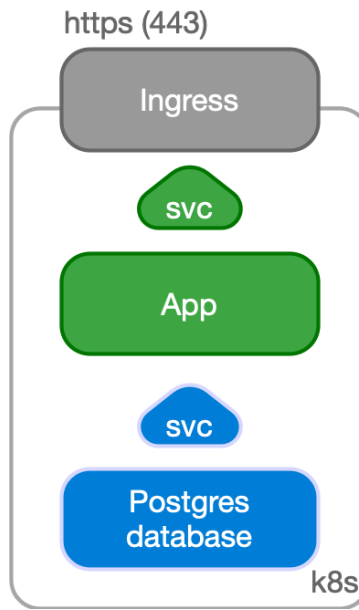


図 1: Application and Database inside Kubernetes

2.2.2 ケース 2Kubernetes 外部のアプリケーション

もう 1 つの考えられるユースケースは、Kubernetes 内で PGD 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、PGD 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、PGD への TLS 接続のメリットを得ることができます。

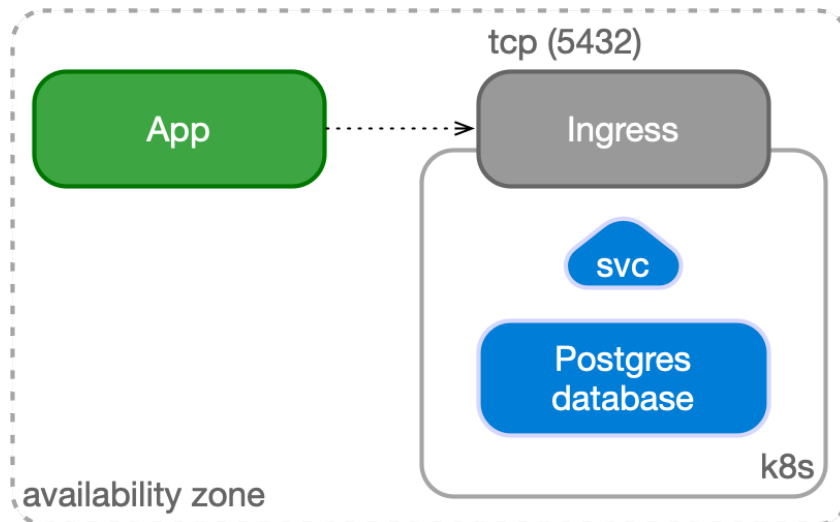


図 2: Application outside Kubernetes

2.3 Architecture

Kubernetes で EDB Postgres 分散を展開するときは、これらの主なアーキテクチャの側面を考慮してください。

Kubernetes 用に分散された EDB Postgres は [Kubernetes 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る EDB Postgres 分散クラスターを展開および管理するように設計されています。

2.3.1 EDB Postgres Distributed との関係

EDB Postgres Distributed (PGD)

は、高いパフォーマンスと可用性を実現するように設計された Postgres のマルチマスター実装です。PGD は通常、[Trusted Postgres Architect \(TPA\)](#) を使用した展開が必要です。このツールは、[Ansible](#) を使用して PGD クラスターをプロビジョニングおよび展開するツールです。

EDB Postgres Distributed for Kubernetes は、コンテナと Kubernetes を活用して、PGD クラスターを展開する別の方法を提供します。利点は、結果のアーキテクチャが次のことです。

- 自己修復性と堅牢性があります。
- 宣言的構成を介して管理されます。
- 広大で成長を続ける Kubernetes エコシステムを利用します。

2.3.2 Kubernetes の EDB Postgres との関係

PGD クラスターは、1 つ以上の `_PGD グループ_` で構成され、それぞれが 1 つ以上の `_PGD ノード_` があります。PGD ノードは Postgres データベースです。EDB Postgres Distributed for Kubernetes は、EDB Postgres for Kubernetes で定義された `Cluster` リソース、特に単一のインスタンスつまりレプリカなしのクラスターを使用して各 PGD ノードを内部管理します。

`.spec.cnf` の各 `Cluster` によって作成された単一の PostgreSQL インスタンスを構成できます。

PGD グループ仕様の

EDB Postgres Distributed for Kubernetes では、EDB Postgres for Kubernetes と同様に、基になるデータベース実装がデータレプリケーションを担当します。ただし、フェイルオーバーとスイッチオーバーの動作が異なり、Raft 選挙と新しい書き込みリーダーの指名が必要になることに注意することが重要です。Kubernetes の EDB Postgres は、データノードの展開と修復のみを処理します。

2.3.3 Kubernetes 用に分散された EDB Postgres を使用した PGD の管理

EDB Postgres Distributed for Kubernetes オペレーターは、PGD クラスターの完全なライフサイクルを管理できます。そのため、PGD ノード単一インスタンス **Clusters** として表されるに加えて、PGD に関連付けられている他のオブジェクトを管理する必要があります。

PGD は、分散コンセンサスのための Raft アルゴリズムに依存して、ノードのメタデータ、特に `_write leader_` に関する契約を管理します。データノード間のコンセンサスは、新しいグローバルシーケンスの生成や分散 DDL の実行などの操作にも必要です。

これらの考慮事項により、データベースノードの上の PGD に追加のアクターが強制されます。

EDB Postgres Distributed for Kubernetes は、以下を管理します。

- データノード。ノードはデータベースであり、EDB Postgres for Kubernetes によって管理され、単一のインスタンスで **Cluster** を作成します。
- **Witness nodes**

は、データレプリケーションに参加しない基本的なデータベースインスタンスです。彼らの機能は、偶数のデータノードを持つグループ、またはネットワーク分割後のコンセンサスが可能であることを保証することです。Witness ノードは、単一インスタンスの **Cluster** リソースを使用して管理されます。

- **PGD proxies**

書き込みリーダーの情報をを使用して Postgres プロキシとして機能します。PGD プロキシは、書き込みを現在の書き込みリーダーにルーティングするために Raft からの情報を必要とします。

プロキシとルーティング

PGD グループは、PGD ノードのフルメッシュ接続を想定しています。各ノードは、適切な接続文字列 `libpq` スタイルの DSN を使用して他のすべてのノードに接続する必要があります。書き込み操作はすべてのノードに送信される必要はありません。PGD は、1 つのノードにコミットされた後、データの複製を担当します。

パフォーマンスを向上させるために、書き込み操作のほとんどを単一のノード `_write leader_` に送信することをお勧めします。Raft は、どのノードが書き込みリーダーであるかを識別し、PGD ノードに関するメタデータを保持するために使用されます。PGD プロキシは、書き込みを書き込みリーダーに透過的にルーティングし、スイッチオーバーまたはフェイルオーバーが発生した場合に新しい書き込みリーダーにすばやくピボットするために使用されます。

`_Raft subgroups_` を構成することが可能であり、それぞれが別個の書き込みリーダーを維持できます。EDB Postgres Distributed for Kubernetes では、PGD プロキシを含む PGD グループは Raft サブグループを構成します。

PGD プロキシでは 2 種類のルーティングが使用できます。

- グローバルルーティングは、トップレベルの Raft グループを使用し、1 つのグローバル書き込みリーダーを維持します。
- ローカルルーティングは、サブグループを使用して個別の書き込みリーダーを維持します。ローカルルーティングは、書き込みの地理的な分離を実現するために使用されることがよくあります。

EDB Postgres Distributed for Kubernetes では、デフォルトでローカルルーティングが使用され、グローバルルーティングを選択する構成オプションを使用できます。

Raft を使用したルーティングの詳細については、PGD ドキュメントの [Proxies, Raft, and Raft subgroups](#) を参照してください。

PGD アーキテクチャと高可用性

EDB は、PGD の分散マルチマスター機能を有効に活用し、高可用性を提供するために、いくつかの推奨アーキテクチャを提案しています。

Always On アーキテクチャは、1 つの場所にある 1 つのグループ、または 2 つの別の場所にある 2 つのグループから構築されます。詳細については、PGD ドキュメントの [Choosing your architecture](#) を参照してください。

2.3.4 Kubernetes での PGD の展開

EDB Postgres Distributed for Kubernetes は、Kubernetes を活用して PGD クラスターを展開および管理します。そのため、PGD を Kubernetes エコシステムに変換するには、いくつかの適応が必要です。

イメージとオペランド

3 つの Postgres ディストリビューションのいずれかを実行するように PGD を構成できます。[PGD documentation](#) を参照してください

各ディストリビューションの特徴を理解するため。

Kubernetes で機能するには、Postgres ディストリビューションごとにコンテナが提供されます。これらは `_operands_` です。さらに、オペレーターイメージはこれらと同じリポジトリに保持されます。

EDB private image registries を参照

画像へのアクセスの詳細については、

Kubernetes アーキテクチャ

[PG4K document on Kubernetes architecture](#) のポイントの一部

はここに再現されています。詳細については、PG4K ドキュメントを参照してください。

Kubernetes は、別の物理的な場所にまたがる可能性をネイティブに提供します。これらの物理的な場所は、データセンター、障害ゾーン、またはより多くの場合、`_アベイラビリティゾーン_`としても知られます。これらは、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ます。

分散システムであるため、`_Kubernetes クラスター_`の推奨される最小数は 3 です。この最小値により、コントロールプレーンは単一ゾーンの障害に対する回復力があります。これは、各データセンターがいつでもアクティブで、ワークロードを同時に実行できることを意味します。

Kubernetes の EDB Postgres Distributed を **単一の Kubernetes クラスター** にインストールできます
 または **複数の Kubernetes クラスター** 全体にわたって。

単一の Kubernetes クラスター

マルチアベイラビリティゾーンの Kubernetes アーキテクチャは、クラウドプロバイダーが管理する Kubernetes サービスの一般的なものです。このようなアーキテクチャにより、EDB Postgres Distributed for Kubernetes と EDB Postgres for Kubernetes オペレーターは、すべてのゾーンがアクティブであることを考慮して、アベイラビリティゾーンにわたってワークロードとノードをスケジュールできます。

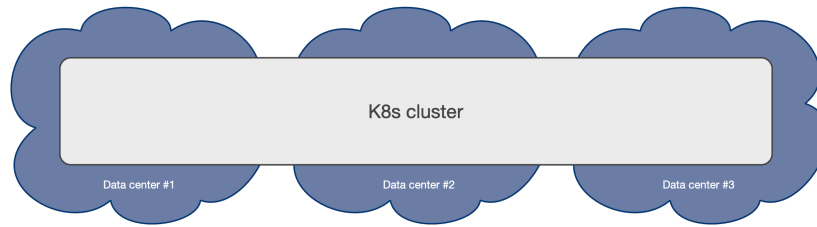


図 3: Kubernetes cluster spanning over 3 independent data centers

PGD クラスターは、単一の Kubernetes クラスターに展開でき、Kubernetes 可用性ゾーンを利用して、Always On 推奨アーキテクチャを含む高可用性アーキテクチャを有効にすることができます。

PGD ドキュメントの [Choosing your architecture](#) に示されている `_Always On Single Location_` アーキテクチャは、3つのアベイラビリティゾーンを持つ単一の Kubernetes クラスターで実現できます。

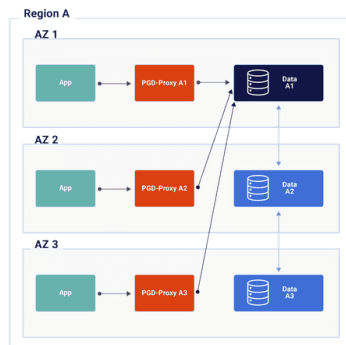


図 4: Always On Single Region

EDB Postgres Distributed for Kubernetes オペレーターは、EDB Postgres for Kubernetes の場合と同様に、アフィニティ、トレランス、ノードセクターを使用して、ポッドのスケジューリングつまりどのポッドがどのデータセンターに行くかを制御できます。個々のスケジューリング制御は、ノードだけでなくプロキシでも利用できます。

詳細については、EDB Postgres for Kubernetes ドキュメントの [Kubernetes documentation on scheduling](#) および [Scheduling](#) を参照してください。

複数の Kubernetes クラスター

PGD クラスターは、相互に確実に通信できる複数の Kubernetes クラスターに展開することもできます。

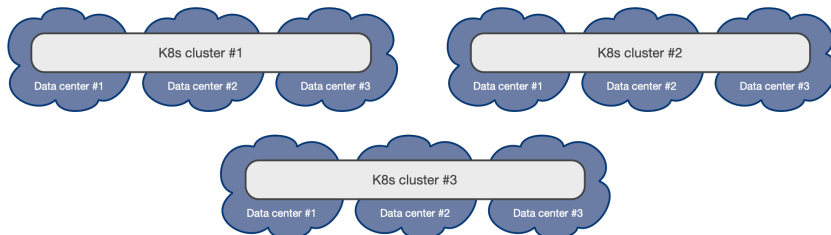


図 5: Multiple Kubernetes clusters

Always On multi-location PGD architectures

接続要件を満たす複数の Kubernetes クラスターで実現できます。

詳細については、[Connectivity](#) を参照してください。

2.4 Installation and upgrade

参考:

Red Hat OpenShift Container Platform に Cloud Native PostgreSQL をインストールする方法については、[Red Hat OpenShift](#) での 1.0.1 へのアップグレードを参照してください。

セクション。

2.4.1 Kubernetes へのオペレーターのインストール

EDB サブスクリプショントークンの取得

重要: Kubernetes の EDB Postgres Distributed をインストールするには、EDB サブスクリプショントークンを取得する必要があります。トークンは、EDB プライベートソフトウェアリポジトリへのアクセスを付与します。

Kubernetes 用 EDB Postgres Distributed をインストールするには、EDB Repos 2.0 トークンが必要であり、EDB プライベートソフトウェアリポジトリへのアクセスを取得します。

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

アカウントプロフィールページには、Repos 2.0 Token のラベルの横にトークンが表示されます。デフォルトでは、トークンは隠蔽されています。「Show」ボタン目のアイコンをクリックして、表示します。

トークンにより、2つのリポジトリ `standard` または `enterprise` のいずれかにアクセスすることができます。

- **standard** - 演算子と EDB Postgres 拡張オペランドイメージが含まれます。
- **enterprise** - EDB Postgres Advanced および EDB Postgres Extended の演算子、およびオペランドイメージが含まれています。

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

```
EDB_SUBSCRIPTION_PLAN=enterprise
```

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

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

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

これで、インストールを続行できます。

Helm チャートを使用する

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

オペレーターマニフェストを直接使用する

マニフェストを直接使用して、EDB Postgres Distributed for Kubernetes オペレーターを展開できます。このマニフェストは、EDB Postgres Distributed for Kubernetes operator と最新にサポートされている EDB Postgres for Kubernetes operator の両方を同じ名前空間にインストールします。マニフェストを使用してオペレーターを展開するには、以下の手順に従います。

cert-manager のインストール

EDB Postgres Distributed for Kubernetes には、Cert Manager 1.10 以降が必要です。[installation guide](#) に従うか、次のコマンドを使用して cert-manager を展開できます。

```
kubectl apply -f \
  https://github.com/cert-manager/cert-manager/releases/download/v1.16.2/cert-manager.
  ↪yaml
```

EDB プルシークレットのインストール

Kubernetes 用 EDB Postgres 分散をインストールする前に、EDB コンテナレジストリのプルシークレットを作成する必要があります。

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

```
kubectl create namespace pgd-operator-system
```

プルシークレットを作成するには、次のコマンドを実行します。

```
kubectl create secret -n pgd-operator-system docker-registry edb-pull-secret \
  --docker-server=docker.enterprisedb.com \
  --docker-username=k8s_${EDB_SUBSCRIPTION_PLAN}_pgd \
  --docker-password=${EDB_SUBSCRIPTION_TOKEN}
```

オペレーターマニフェストのインストール

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

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

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

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

```
kubectl apply --server-side -f \
  https://get.enterprisedb.io/pg4k-pgd/pg4k-pgd-${EDB_SUBSCRIPTION_PLAN}-1.1.1.yaml
```

次のコマンドでオペレーターの展開を確認できます。

```
kubectl get deployment -n pgd-operator-system pgd-operator-controller-manager
```

注釈: EDB Postgres Distributed for Kubernetes は、EDB Postgres for Kubernetes によって定義された `Cluster` リソースを使用して各 PGD ノードを内部管理するため、EDB Postgres for Kubernetes オペレーターも依存関係としてインストールする必要があります。上記で使用されるマニフェストには、EDB Postgres for Kubernetes オペレーターの十分にテストされたバージョンが含まれており、EDB Postgres Distributed for Kubernetes オペレーターと同じ名前空間にインストールされます。

2.4.2 デプロイメントの詳細

Kubernetes では、オペレーターはデフォルトで Kubernetes Deployment として `pgd-operator-system` 名前空間にインストールされます。この展開の名前は、インストール方法によって異なります。マニフェストを介してインストールされる場合、デフォルトでは、`pgd-operator-controller-manager` という名前が付けられます。Helm を介してインストールされる場合、デフォルトでは、展開名は `helm` リリース名にサフィックス-`edb-postgres-distributed-for-kubernetes` つまり `<name>-edb-postgres-distributed-for-kubernetes` を追加したものから取得されます。

注釈: Helm を使用すると、`"values.yaml"` file の `fullnameOverride` フィールドを介してデプロイメントの名前をカスタマイズできます。

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

```
$ kubectl get deployments -n pgd-operator-system
```

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

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

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。EDB Postgres Distributed for Kubernetes オペレーターのデフォルト構成は、単一のレプリカの展開であり、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い場合、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。さらに、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と同じノードでオペレーターが実行されないようにできます。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も含まれる場合があります。

参考:

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

2.4.3 PGD クラスターを展開する

PGD クラスターの展開を開始する前に、証明書発行者を作成してください。Helm チャートはこれを行うように求めますが、見逃した場合に備えて、たとえば次のことを実行できます。

```
kubectl apply -f \
  https://raw.githubusercontent.com/EnterpriseDB/edb-postgres-for-kubernetes-charts/
  ↪main/hack/samples/issuer-selfsigned.yaml
```

オペレーターと自己署名証明書発行者が展開されたら、PGD クラスターの作成を開始できます。例については、[Quick start](#) を参照してください。

デフォルトオペランドイメージ

デフォルトでは、各オペレーターリリースは、PGD および PGD プロキシイメージのデフォルトバージョンとフレーバーをバインドします。pgdgroup YAML ファイルの `spec.pgd.imageName` および `spec.pgdproxy.imageName` フィールドでイメージ名が指定されていない場合、デフォルトのイメージが使用されます。

デフォルトイメージは、`pgd-operator-controller-manager-config` オペレーター構成マップを使用して上書きできます。詳細については、[Kubernetes Operator 構成用に分散された EDB Postgres](#) のセクションを参照してください。

PGD グループのステータスを確認することにより、PGD クラスターが使用しているイメージを見つけることができます。

```
kubectl get pgdgroup <pgdgroup name> -o yaml | yq ".status.image"
```

オペランドイメージの指定

次に、明示的なイメージ名を使用した PGD クラスターの例を示します。

```
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: group-example-customized
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  imageName: docker.enterprisedb.com/k8s_enterprise_pgd/edb-postgres-extended-pgd:17.
→4-5.7.0-1
  imagePullSecrets:
    - name: registry-pullsecret
  pgd:
    parentGroup:
      name: world
      create: true
  cnp:
    storage:
      size: 1Gi
  pgdProxy:
    imageName: docker.enterprisedb.com/k8s_enterprise_pgd/edb-pgd-proxy:5.7.1-1
```

ImageCatalog を使用してオペランドイメージを指定する

バージョン 1.1.1 のリリース以降、PGD4K-PGD オペレーターは、ImageCatalog を使用してオペランドイメージを指定することをサポートしています。

サブスクリプションプランと、各 PostgreSQL フレーバーの PGD バージョンに基づいて、さまざまな ImageCatalogs を利用できます。ImageCatalog に含まれる画像は ubi-9 ベースであることに注意してください。

- EDB Postgres Advanced PGD < <https://get.enterprisedb.io/pgd-k8s-image-catalogs/epas-k8s> - ><EDB_SUBSCRIPTION_PLAN>-pgd-<PGD_VERSION>.yaml
- EDB Postgres 拡張 PGD < <https://get.enterprisedb.io/pgd-k8s-image-catalogs/pgextended-k8s> - ><EDB_SUBSCRIPTION_PLAN>-pgd-<PGD_VERSION>.yaml
- Postgres コミュニティ PGD <https://get.enterprisedb.io/pgd-k8s-image-catalogs/postgresql-k8s> - <EDB_SUBSCRIPTION_PLAN>-pgd-<PGD_VERSION>.yaml

PGD クラスター名前空間に Imagecatalog を作成し、YAML ファイルで参照できます。

EDB Postgres Extended PGD 5.7 ImageCatalog を作成してみましょう。

```
kubectl create -f \
    https://get.enterprisedb.io/pgd-k8s-image-catalogs/epas-k8s-enterprise-pgd-5.
→7.yaml
```

次の例は、EDB Postgres 拡張 PGD 5.7 ImageCatalog を使用して、PGD オペランドイメージに PostgreSQL メジャーバージョン 17 を指定する方法を示しています。PGD プロキシイメージは、ImageCatalog から自動的に取得されます。

```
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: group-example-catalog
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  imageCatalogRef:
    apiGroup: pgd.k8s.enterprisedb.io
    kind: ImageCatalog
    major: 17
    name: epas-enterprise-pgd-5.7
...
```

2.4.4 アップグレード

重要: *EDB Postgres Distributed for Kubernetes 1.0.0 release notes* をよくお読みください

バージョンによっては追加の手順が必要になる場合があるため、アップグレードを実行する前に。

EDB Postgres Distributed for Kubernetes PGD4K オペレーターは、EDB Postgres for Kubernetes PG4K オペレーターに依存してクラスターを管理します。EDB PGD4K オペレーターをアップグレードするには、PG4K オペレーターもサポートされているバージョンにアップグレードする必要があります。EDB PG4K オペレーターは [Long Term Support \(LTS\)](#) で維持することをお勧めします

これは、PGD4K オペレーターと互換性のあるテストされたバージョンであるため。

Kubernetes オペレーターの EDB Postgres DistributedPGD をアップグレードするには、次の手順を実行します。

1. 依存関係としての EDB Postgres for Kubernetes オペレーターをアップグレードする
2. PGD4K 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

EDB Postgres for Kubernetes オペレーターをアップグレードすると、各 PGD ノードのインスタスマネージャーもアップグレードされることに注意してください。詳細については、*EDB Postgres for Kubernetes upgrades documentation* を参照してください。

リリースノートに別段の記載がない限り、これらの手順は、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ことにより実行されます。

バージョン間の互換性

EDB Postgres Distributed for Kubernetes PGD4K は、セマンティックバージョンニングに従います。同じ API バージョン内のオペレーターのすべてのリリースは、以前の API バージョンと互換性があります。現在の API バージョンは v1 で、オペレーターのバージョン 1.xy に対応します。

PGD4K オペレーターの minor バージョンは、PG4K LTS リリースの変更を追跡しています。例

- PGD4K オペレーター 1.0.x は、PG4K LTS 1.22 に対して完全にテストされています。
- PGD4K オペレーター 1.1.x は、PG4K LTS 1.25 に対して完全にテストされています。

PGD4K オペレーターリリースのサポート範囲は、追跡している PG4K LTS リリースと同じです。

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。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ます。このため、**ユーザーが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。**

EDB Postgres Distributed for Kubernetes 1.0.0 release notes ページには、EDB Postgres Distributed for Kubernetes 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だけでアップグレードが完了します。

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の PGD4K と PG4K の両方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ず、オペレーター自身自身にのみ影響します。

Red Hat OpenShift での 1.0.1 へのアップグレード

Openshift プラットフォームでは、バージョン 1.0.1 から、EDB Postgres Distributed for Kubernetes オペレーターは、PG4K オペレーターの LTS リリースを参照する必要があります。たとえば、PGD4K v1.0.1 は、特に PG4K LTS バージョン 1.22.x を参照します 1.22 LTS ブランチのパッチリリースは有効です。

PGD4K オペレーターをアップグレードするには、最初に PG4K オペレーターがサポートされているバージョンにアップグレードされていることを確認します。その場合にのみ、PGD4K オペレーターをアップグレードできます。

重要: PGD4K v1.0.0 は、PG4K オペレーターの LTS リリースを参照する必要があるため、任意の PG4K バージョンでインストールされている可能性があります。インストールされている PG4K バージョンが 1.22 より前の場合、サブスクリプションチャンネルを変更することにより PG4K バージョン 1.22 にアップグレードできます。詳細については、[upgrading the operator](#) を参照してください。ただし、インストールされた PG4K バージョンが 1.22 より大きい場合、stable-1.22 チャンネルからオペレーターを再インストールして、PGD4K を v1.0.1 にアップグレードする必要があります。

マニフェストのサーバーサイドの適用

Kubernetes 1.29 および次のバージョンとの互換性を保証するために、Kubernetes の EDB Postgres Distributed は **Server-side apply** の使用を義務付けました

オペレーターマニフェストを展開するとき。

このインストール方法を採用することは、新しい展開に問題はありませんが、`--server-side` オプションを使用して既存のオペレーターマニフェストを更新すると、以下の例のようなエラーが発生する場合があります。

```
Apply failed with 1 conflict: conflict with "kubectl-client-side-apply" using...
```

このようなエラーが発生した場合は、`--force-conflicts` オプションを明示的に指定して競合解決を強制することにより解決できます。

```
kubectl apply --server-side --force-conflicts -f <OPERATOR_MANIFEST>
```

それ以降、`kube-apiserver` は CRD の認識マネージャーとして自動的に認識され、この問題に関するさらなる手動介入の必要はありません。

オペランドのアップグレード

PGD クラスターは、オペランドイメージのマイナーバージョンのインプレースアップグレードをサポートしていますが、アップグレード中に PostgreSQL サービスは一時的に使用できなくなります。

現在の PGD とプロキシバージョンの確認

アップグレードする前に、次のコマンドを使用して、現在の PGD および PGD プロキシバージョンを確認できます。

```
kubectl get pgdgroup <pgdgroup name> -o yaml | yq ".status.image"
```

アップグレードプロシージャ

- デフォルトまたはカスタマイズされたイメージ名の使用オペランドを新しいマイナーバージョンにアップグレードするには、PGD グループ YAML ファイルの `spec.imageName` および `spec.pgproxy.imageName` セクションの `imageName` を新しい `imageName` に置き換えます。各ノードのイメージは順次アップグレードされ、それに応じて再起動されます。
- イメージカタログの使用 PGD クラスターが `ImageCatalog` を使用してイメージバージョンを管理している場合、参照される `ImageCatalog` で指定されたイメージバージョンを単にアップグレードします。PGD クラスターは、新しいイメージバージョンを自動的に適用します。

2.5 Quick start

`Kind` で構築された単一のローカル Kubernetes クラスターで Kubernetes 用 EDB EDB Postgres EDB Postgres DistributedPGD クラスターをテストできます。

警告: これらの指示は、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。

このクイックスタートでは、ローカルの Kubernetes インストールで EDB Postgres 分散クラスターを起動して、実験できる方法を示します。

重要: Kubernetes クラスターに接続するには、マシンに `kubectl` がインストールされていることを確認します。 `installing kubectl` <<https://kubernetes.io/docs/tasks/tools/install-kubectl/>>_ に関する Kubernetes ドキュメントを参照してください。

2.5.1 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

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ある Kind をインストールします。Kind は Kubernetes IN Docker の略です。既に Kubernetes クラスターにアクセスしている場合、パート 2 に進んでください。

[Kind Quick Start](#) の指示に従って、Kind を環境にインストールします。次に、Kubernetes クラスターを作成します。

```
kind create cluster --name pgd
```

2.5.2 パート 2-Kubernetes 用に分散された EDB Postgres 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たら、Kubernetes の EDB Postgres Distributed をインストールできます。

詳細は、[Installation and upgrade](#) を参照してください。

2.5.3 パート 3-PGD クラスターを展開する

Kubernetes の他のデプロイメントと同様に、PGD クラスターを展開するには、PGD クラスターを構成する目的の PGDGroup リソースを定義する構成ファイルを適用する必要があります。

一部のサンプルファイルは、EDB Postgres Distributed for Kubernetes リポジトリに含まれています。マニフェストには、2つのデータグループと3つのリージョンにまたがるグローバル監視ノードを含むPGD クラスターの定義が含まれています。各データグループは、2つのデータノードと1つのローカル監視ノードで構成されています。

```
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: region-a
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  pgd:
    parentGroup:
      name: world
      create: true
    discovery:
      - host: region-a-group
      - host: region-b-group
      - host: region-c-group
  cnp:
```

(次のページに続く)

(前のページからの続き)

```

storage:
  size: 1Gi
connectivity:
  dns:
    additional:
      - domain: alternate.domain
      - domain: my.domain
    hostSuffix: -dc1
  tls:
    mode: verify-ca
    clientCert:
      caCertSecret: client-ca-key-pair
      certManager:
        spec:
          issuerRef:
            name: client-ca-issuer
            kind: Issuer
            group: cert-manager.io
    serverCert:
      caCertSecret: server-ca-key-pair
      certManager:
        spec:
          issuerRef:
            name: server-ca-issuer
            kind: Issuer
            group: cert-manager.io
-- --
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: region-b
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  pgd:
    parentGroup:
      name: world
    discovery:
      - host: region-a-group
      - host: region-b-group
      - host: region-c-group
  cnp:
    storage:
      size: 1Gi

```

(次のページに続く)

(前のページからの続き)

```

connectivity:
  tls:
    mode: verify-ca
    clientCert:
      caCertSecret: client-ca-key-pair
      certManager:
        spec:
          issuerRef:
            name: client-ca-issuer
            kind: Issuer
            group: cert-manager.io
    serverCert:
      caCertSecret: server-ca-key-pair
      certManager:
        spec:
          issuerRef:
            name: server-ca-issuer
            kind: Issuer
            group: cert-manager.io
  ---
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: region-c
spec:
  instances: 0
  proxyInstances: 0
  witnessInstances: 1
  pgd:
    parentGroup:
      name: world
    discovery:
      - host: region-a-group
      - host: region-b-group
      - host: region-c-group
  cnp:
    storage:
      size: 1Gi
  connectivity:
    tls:
      mode: verify-ca
      clientCert:
        caCertSecret: client-ca-key-pair
        certManager:
          spec:

```

(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
  serverCert:
    caCertSecret: server-ca-key-pair
  certManager:
    spec:
      issuerRef:
        name: server-ca-issuer
        kind: Issuer
        group: cert-manager.io

```

最初に保存して実行することにより、`flexible-3-regions` 例を展開できます。

```
kubectl apply -f flexible_3regions.yaml
```

`get pods` コマンドを使用して、ポッドが作成されていることを確認できます。

```
kubectl get pods
```

ポッドは PGD ノードの一部として作成されています。[Architecture](#) で説明しているように、これらは Kubernetes クラスターの EDB Postgres の上に実装されています。

次に、クラスターをリストすることができます。これは、PGD ノードを表示します。

```
$ kubectl get clusters
```

NAME	AGE	INSTANCES	READY	STATUS	PRIMARY
region-a-1	2m50s	1	1	Cluster in healthy state	region-a-1-1
region-a-2	118s	1	1	Cluster in healthy state	region-a-2-1
region-a-3	91s	1	1	Cluster in healthy state	region-a-3-1
...					
...					

最終的に、PGD ノードは、PGD クラスターを構成する PGD グループの一部として作成されます。

```
$ kubectl get pgdgroups
```

NAME	DATA INSTANCES	WITNESS INSTANCES	PHASE	AGE
region-a	2	1	PGDGroup - Healthy	4m50s
region-b	2	1	PGDGroup - Healthy	4m50s
region-c	0	1	PGDGroup - Healthy	4m50s

region-c グループが監視ノードにすぎないことに注意してください。

第3章 Configuration

3.1 Operator configuration

3.1.1 Kubernetes Operator 構成用に分散された EDB Postgres

EDB Postgres Distributed for Kubernetes PGD4K のオペレーターは、標準の展開マニフェストからインストールされ、構成パラダイムに関する規約に従います。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。

- PGD または PGD プロキシの別のデフォルトイメージを定義
- 演算子とオペランドイメージの追加のプルシークレットを定義

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 `pgd-operator-controller-manager-config` を持つ `ConfigMap /Secret` を介してカスタマイズできます。

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

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

使用可能なオプション

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

Name	Description
<code>PGD_IMAGE_NAME</code>	マニフェストのデフォルトのものを上書きする PGD オペランドイメージ名。
<code>PGD_PROXY_IMAGE_NAME</code>	マニフェストのデフォルトのものを上書きする PGD プロキシオペランドイメージ名。
<code>PULL_SECRET_NAME</code>	オペレーターの名前空間で定義され、イメージのダウンロードに使用される追加のプルシークレットの名前

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 PGD グループ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<group-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。

オペレーター構成マップの定義

次の例では、追加のイメージプルシークレットを定義することにより、オペレーターの動作をカスタマイズします。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: pgd-operator-controller-manager-config
  namespace: pgd-operator-system
data:
  PULL_SECRET_NAME: additional-pull-secret
```

オペレーターシークレットの定義

次の例では、デフォルトのオペランドイメージ名を定義することにより、オペレーターの動作をカスタマイズします。

```
apiVersion: v1
kind: Secret
metadata:
  name: pgd-operator-controller-manager-config
  namespace: pgd-operator-system
type: Opaque
stringData:
  PGD_IMAGE_NAME: docker.enterprisedb.com/k8s_enterprise_pgd/postgresql-pgd:17.4-5.7.
  → 0-1
  PGD_PROXY_IMAGE_NAME: docker.enterprisedb.com/k8s_enterprise_pgd/edb-pgd-proxy:5.7.
  → 0-1
```

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 pgd-operator-system \
  pgd-operator-controller-manager
```

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

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

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 `PGD groups` リソースにのみ適用されます。

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一覧表示する 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` . 要求に一覧表示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 pgd-operator-system pgd-operator-controller-manager
```

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

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

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

3.1.2 Kubernetes Operator 構成の EDB Postgres

EDB Postgres Distributed for Kubernetes PGD4K オペレーターは、EDB Postgres for Kubernetes PG4K オペレーターによって定義された Cluster リソースを使用して、各 PGD ノードを管理します。したがって、PG4K オペレーターの構成は引き続き適用できます。[operator configuration](#) を参照してください。

詳細については、

3.2 Examples of configuration

重要: 使用可能な例は、デモンストレーションと実験のみを目的としています。

これらの例は、Kubernetes 環境で EDB Postgres 分散クラスターをセットアップするための構成ファイルです。

3.2.1 基本

柔軟な 3 つのリージョン

:

```
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: region-a
spec:
  instances: 2
```

(次のページに続く)

(前のページからの続き)

```

proxyInstances: 2
witnessInstances: 1
pgd:
  parentGroup:
    name: world
    create: true
  discovery:
    - host: region-a-group
    - host: region-b-group
    - host: region-c-group
cnp:
  storage:
    size: 1Gi
connectivity:
  dns:
    additional:
      - domain: alternate.domain
      - domain: my.domain
      hostSuffix: -dc1
  tls:
    mode: verify-ca
    clientCert:
      caCertSecret: client-ca-key-pair
      certManager:
        spec:
          issuerRef:
            name: client-ca-issuer
            kind: Issuer
            group: cert-manager.io
    serverCert:
      caCertSecret: server-ca-key-pair
      certManager:
        spec:
          issuerRef:
            name: server-ca-issuer
            kind: Issuer
            group: cert-manager.io
--
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: region-b
spec:
  instances: 2
  proxyInstances: 2

```

(次のページに続く)

(前のページからの続き)

```

witnessInstances: 1
pgd:
  parentGroup:
    name: world
  discovery:
    - host: region-a-group
    - host: region-b-group
    - host: region-c-group
cnp:
  storage:
    size: 1Gi
connectivity:
  tls:
    mode: verify-ca
    clientCert:
      caCertSecret: client-ca-key-pair
      certManager:
        spec:
          issuerRef:
            name: client-ca-issuer
            kind: Issuer
            group: cert-manager.io
    serverCert:
      caCertSecret: server-ca-key-pair
      certManager:
        spec:
          issuerRef:
            name: server-ca-issuer
            kind: Issuer
            group: cert-manager.io
- - -
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: region-c
spec:
  instances: 0
  proxyInstances: 0
  witnessInstances: 1
  pgd:
    parentGroup:
      name: world
    discovery:
      - host: region-a-group
      - host: region-b-group

```

(次のページに続く)

(前のページからの続き)

```

- host: region-c-group
cnp:
  storage:
    size: 1Gi
connectivity:
  tls:
    mode: verify-ca
    clientCert:
      caCertSecret: client-ca-key-pair
      certManager:
        spec:
          issuerRef:
            name: client-ca-issuer
            kind: Issuer
            group: cert-manager.io
    serverCert:
      caCertSecret: server-ca-key-pair
      certManager:
        spec:
          issuerRef:
            name: server-ca-issuer
            kind: Issuer
            group: cert-manager.io

```

は、2つのデータグループと3つのリージョンにまたがるグローバル監視ノードを含む PGD クラスターを提供します。各データグループは2つのデータノードとローカル監視ノードで構成されます。

物理的結合を使用した柔軟な3リージョン

:

```

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: region-a
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  pgd:
    parentGroup:
      name: world
      create: true
  cnp:
    joinMethod: physical
    storage:

```

(次のページに続く)

(前のページからの続き)

```

    size: 1Gi
connectivity:
  dns:
    additional:
      - domain: alternate.domain
      - domain: my.domain
      hostSuffix: -dc1
  tls:
    mode: verify-ca
    clientCert:
      caCertSecret: client-ca-key-pair
      certManager:
        spec:
          issuerRef:
            name: client-ca-issuer
            kind: Issuer
            group: cert-manager.io
    serverCert:
      caCertSecret: server-ca-key-pair
      certManager:
        spec:
          issuerRef:
            name: server-ca-issuer
            kind: Issuer
            group: cert-manager.io
--
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: region-b
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  pgd:
    parentGroup:
      name: world
    # The first node in region-b will join to one of the node defined in discovery.
    ↪list using physical join
    groupJoinMethod: physical
    # list the data node of region a to physical Join
    discovery:
      - host: region-a-3-node
      - host: region-a-1-node
      - host: region-a-2-node

```

(次のページに続く)

(前のページからの続き)

```

cnp:
  joinMethod: physical
  storage:
    size: 1Gi
connectivity:
  tls:
    mode: verify-ca
    clientCert:
      caCertSecret: client-ca-key-pair
      certManager:
        spec:
          issuerRef:
            name: client-ca-issuer
            kind: Issuer
            group: cert-manager.io
    serverCert:
      caCertSecret: server-ca-key-pair
      certManager:
        spec:
          issuerRef:
            name: server-ca-issuer
            kind: Issuer
            group: cert-manager.io
-- --
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: region-c
spec:
  instances: 0
  proxyInstances: 0
  witnessInstances: 1
  pgd:
    parentGroup:
      name: world
    discovery:
      - host: region-a-group
      - host: region-b-group
      - host: region-c-group
  cnp:
    storage:
      size: 1Gi
  connectivity:
    tls:
      mode: verify-ca

```

(次のページに続く)

(前のページからの続き)

```

clientCert:
  caCertSecret: client-ca-key-pair
  certManager:
    spec:
      issuerRef:
        name: client-ca-issuer
        kind: Issuer
        group: cert-manager.io
serverCert:
  caCertSecret: server-ca-key-pair
  certManager:
    spec:
      issuerRef:
        name: server-ca-issuer
        kind: Issuer
        group: cert-manager.io

```

は、2つのデータグループと1つのグローバル監視ノードを含むPGD クラスターを提供します。Flexible 3 regions のような。違いは、このサンプルのグループ結合が物理的結合を使用していることです。グループ region-a は、最初に起動するグループ `spec.pgd.parentGroup.create=true`、region-a のノードは物理的結合を使用して互いに結合します。グループリージョン-b はその `spec.pgd.groupJoinMethod=physical` を定義します。これは、リージョン-b の最初のノードが起動してリモートノードに物理的に参加することを意味します、参加ターゲットはグループリージョン-b の `spec.pgd.discovery` セクションで定義されます。リージョン-b の残りのノードはに参加しますそれぞれが `spec.cnp.joinMethod=physical` として物理的結合を使用します。region-c は、常に論理結合を使用する監視グループです。

事前にプロビジョニングされたクライアントシークレットを使用した柔軟な 3 リージョン

:

```

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: region-a
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  pgd:
    parentGroup:
      name: world
      create: true
    discovery:
      - host: region-a-group
      - host: region-b-group
      - host: region-c-group

```

(次のページに続く)

(前のページからの続き)

```

cnp:
  storage:
    size: 1Gi
connectivity:
  dns:
    additional:
      - domain: my.domain
        hostSuffix: -dc1
  tls:
    mode: verify-ca
    clientCert:
      # if the pre-provisioned secrets is set, the caCertSecret will accept a CA
      ↪secrets, which contains ca.crt or ca.key optionally
      caCertSecret: client-ca-secrets
    preProvisioned:
      streamingReplica:
        secretRef: replication-secrets-region-a
    serverCert:
      caCertSecret: server-ca-key-pair
      certManager:
        spec:
          issuerRef:
            name: server-ca-issuer
            kind: Issuer
            group: cert-manager.io
--
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: region-b
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  pgd:
    parentGroup:
      name: world
    discovery:
      - host: region-a-group
      - host: region-b-group
      - host: region-c-group
  cnp:
    storage:
      size: 1Gi
    connectivity:

```

(次のページに続く)

(前のページからの続き)

```

dns:
  additional:
    - domain: my.domain
      hostSuffix: -dc1
tls:
  mode: verify-ca
  clientCert:
    caCertSecret: client-ca-secrets
  preProvisioned:
    streamingReplica:
      secretRef: replication-secrets-region-b
  serverCert:
    caCertSecret: server-ca-key-pair
    certManager:
      spec:
        issuerRef:
          name: server-ca-issuer
          kind: Issuer
          group: cert-manager.io
-- --
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: region-c
spec:
  instances: 0
  proxyInstances: 0
  witnessInstances: 1
  pgd:
    parentGroup:
      name: world
    discovery:
      - host: region-a-group
      - host: region-b-group
      - host: region-c-group
  cnp:
    storage:
      size: 1Gi
  connectivity:
    dns:
      additional:
        - domain: my.domain
          hostSuffix: -dc1
    tls:
      mode: verify-ca

```

(次のページに続く)

(前のページからの続き)

```

clientCert:
  caCertSecret: client-ca-secrets
  preProvisioned:
    streamingReplica:
      secretRef: replication-secrets-region-c
serverCert:
  caCertSecret: server-ca-key-pair
  certManager:
    spec:
      issuerRef:
        name: server-ca-issuer
        kind: Issuer
        group: cert-manager.io
-- --

### This is the pre-provisioned client certificate which is signed by certificate in
↳ client-ca-secrets

apiVersion: v1
data:
  tls.crt:
↳ LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSB0tLS0tCk1JSUNYekNDQWdTZ0F3SUJBZ01SQUw0N2V5eXEydzNEN0gwM2RPaE9iaFF3Q2dZ
  tls.key:
↳ LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSU1FcFFJQkFBS0NBUEVBNGRHejZsYktsUUFXa0JlK2pwRjgrM0t5RFBC
kind: Secret
metadata:
  name: replication-secrets-region-a
type: Opaque
-- --

### This is the pre-provisioned client certificate which is signed by certificate in
↳ client-ca-secrets

apiVersion: v1
data:
  tls.crt:
↳ LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSB0tLS0tCk1JSUNYekNDQWdTZ0F3SUJBZ01SQUw0N2V5eXEydzNEN0gwM2RPaE9iaFF3Q2dZ
  tls.key:
↳ LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSU1FcFFJQkFBS0NBUEVBNGRHejZsYktsUUFXa0JlK2pwRjgrM0t5RFBC
kind: Secret
metadata:
  name: replication-secrets-region-b
type: kubernetes.io/tls
-- --

```

(次のページに続く)

(前のページからの続き)

```

### This is the pre-provisioned client certificate which is signed by certificate in
→client-ca-secrets

apiVersion: v1
data:
  tls.crt:
    ↳LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSB0tLS0tCk1JSUNYekNDQWdTZ0F3SUJBZ01SQUw0N2V5eXEydzNEN0gwM2RPaE9iaFF3Q2dZ
  tls.key:
    ↳LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSU1FcFFJQkFBS0NBUEVBNGRHejZsYktsUUFXa0JlK2pwRjgrM0t5RFBC
kind: Secret
metadata:
  name: replication-secrets-region-c
type: kubernetes.io/tls
- - -

### This is the client CA certificate, which will be used to sign other client
→certificate

apiVersion: v1
data:
  ca.crt:
    ↳LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSB0tLS0tCk1JSUJoakNDQVN1Z0F3SUJBZ01SQU9ZYmNlcmxSVgzcU55bi8wWkNSem93Q2dZ
kind: Secret
metadata:
  name: client-ca-secrets
type: Opaque

```

は、2つのデータグループと1つのグローバル監視ノードを含むPGD クラスターを提供します。Flexible 3 regions のような。各PGD グループでは、サーバー TLS 証明書はオペレーターと cert-manager によって管理されます。クライアントレプリケーション証明書は事前にプロビジョニングされています。このサンプルをセットアップする手順は次のとおりです。

1. ターゲット名前空間で自己署名発行者と証明書を確立します。事前にプロビジョニングされたクライアントレプリケーション証明書を使用しているため、`issuer-selfsigned.yaml` によって作成されたクライアントシークレット、証明書、および発行者を削除できます。

```

kubectl -n <namespace> apply -f issuer-selfsigned.yaml
kubectl -n <namespace> delete secrets/client-ca-key-pair \
  certificate/client-ca \
  issuer/client-ca-issuer

```

2. PGD グループを作成します。事前にプロビジョニングされたクライアント証明書とその CA 証明書は、`flexible_3regions_previsioned_secrets.yaml` ファイルにも含まれています。

```

kubectl -n <namespace> apply -f flexible_3regions_previsioned_secrets.yaml

```

異なる名前空間の柔軟な 3 つのリージョン

:

```

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: region-a
  namespace: region-a
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  pgd:
    parentGroup:
      name: world
      create: true
    discovery:
      # need list the full discovery name for the group
      - host: region-a-group.region-a.svc.cluster.local
      - host: region-b-group.region-b.svc.cluster.local
      - host: region-c-group.region-c.svc.cluster.local
  cnp:
    storage:
      size: 1Gi
  connectivity:
    dns:
      # we need to configure the domain for the group so it could be resolved
      domain: region-a.svc.cluster.local
      additional:
        - domain: my.domain
          hostSuffix: -dc1
    tls:
      mode: verify-ca
      clientCert:
        caCertSecret: client-ca-key-pair
        certManager:
          spec:
            issuerRef:
              name: client-ca-issuer
              kind: Issuer
              group: cert-manager.io
      serverCert:
        caCertSecret: server-ca-key-pair
        certManager:
          spec:
            issuerRef:

```

(次のページに続く)

(前のページからの続き)

```

        name: server-ca-issuer
        kind: Issuer
        group: cert-manager.io
-- --
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: region-b
  namespace: region-b
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  pgd:
    parentGroup:
      name: world
    discovery:
      # need list the full discovery name for the group
      - host: region-a-group.region-a.svc.cluster.local
      - host: region-b-group.region-b.svc.cluster.local
      - host: region-c-group.region-c.svc.cluster.local
  cnp:
    storage:
      size: 1Gi
  connectivity:
    dns:
      # we need to configure the domain for the group so it could be resolved
      domain: region-b.svc.cluster.local
      additional:
        - domain: my.domain
        hostSuffix: -dc1
    tls:
      mode: verify-ca
      clientCert:
        caCertSecret: client-ca-key-pair
        certManager:
          spec:
            issuerRef:
              name: client-ca-issuer
              kind: Issuer
              group: cert-manager.io
      serverCert:
        caCertSecret: server-ca-key-pair
        certManager:
          spec:

```

(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: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: region-c
  namespace: region-c
spec:
  instances: 0
  proxyInstances: 0
  witnessInstances: 1
  pgd:
    parentGroup:
      name: world
    discovery:
      # need list the full discovery name for the group
      - host: region-a-group.region-a.svc.cluster.local
      - host: region-b-group.region-b.svc.cluster.local
      - host: region-c-group.region-c.svc.cluster.local
  cnp:
    storage:
      size: 1Gi
  connectivity:
    dns:
      # we need to configure the domain for the group so it could be resolved
      domain: region-c.svc.cluster.local
      additional:
        - domain: my.domain
          hostSuffix: -dc1
    tls:
      mode: verify-ca
      clientCert:
        caCertSecret: client-ca-key-pair
        certManager:
          spec:
            issuerRef:
              name: client-ca-issuer
              kind: Issuer
              group: cert-manager.io
      serverCert:
        caCertSecret: server-ca-key-pair
        certManager:

```

(次のページに続く)

(前のページからの続き)

```
spec:
  issuerRef:
    name: server-ca-issuer
    kind: Issuer
    group: cert-manager.io
```

Flexible 3 regions 例に基づいて構築されたサンプルを提供します。各リージョンには、異なる名前空間が割り当てられます。discovery、connectivity セクションが名前空間に合わせて変更されます。このサンプルをセットアップするには、ECDSA アルゴリズムで生成されたプライベートキーを使用して、各名前空間に client-ca-key-pair および server-ca-key-pair シークレットを事前に作成する必要があります。これにより、cert-manager によって生成されたサーバー TLS 証明書とクライアントレプリケーション証明書が同じプライベートキーから署名されることが保証されます。

- 名前空間 region-a、region-b、および region-c を作成します。
- 次を使用して CA シークレットを作成します

```
## Those two secrets are used to store the ECDSA private key for the client and
↪server CA pairs used by certificate
```

```
## in self-signed issuer. The client-ca-key-pair and server-ca-key-pair is finally
↪used in sample flexible_3regions_3ns.yaml
```

```
## This is to make sure in different namespaces, we have same CA certificate
```

```
apiVersion: v1
```

```
data:
```

```
tls.key:
```

```
↪LS0tLS1CRUdJTiBFQyBQUklWQVRFIEtFWS0tLS0tCk1lY0NBUEVFSUNOZW1KbHp1K3VPbFR1UHltWHZLWHN0aFlUSkRRSkpIQzJs
```

```
kind: Secret
```

```
metadata:
```

```
name: client-ca-key-pair
```

```
type: Opaque
```

```
--
```

```
apiVersion: v1
```

```
data:
```

```
tls.key:
```

```
↪LS0tLS1CRUdJTiBFQyBQUklWQVRFIEtFWS0tLS0tCk1lY0NBUEVFSUZqVXc5b2hQNV12blo0RkRML1ZQV092ODhRUGpRb2xHYXFC
```

```
kind: Secret
```

```
metadata:
```

```
name: server-ca-key-pair
```

```
type: Opaque
```

証明書の各名前空間で。

```
kubectl -n region-a apply -f issuer-ecdsa-key.yaml
kubectl -n region-b apply -f issuer-ecdsa-key.yaml
kubectl -n region-c apply -f issuer-ecdsa-key.yaml
```

- 各名前空間に自己署名発行者と対応する証明書を作成します。CA シークレット `client-ca-key-pair` および `server-ca-key-pair` は、`ca.crt` で更新されます。

```
kubectl -n region-a apply -f issuer-selfsigned.yaml
kubectl -n region-b apply -f issuer-selfsigned.yaml
kubectl -n region-c apply -f issuer-selfsigned.yaml
```

- PGD グループを作成する

```
kubectl apply -f flexible_3regions_3ns.yaml
```

分割されたグループのクリーンアップ

:

```
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroupCleanup
metadata:
  name: region-b-cleanup
spec:
  executor: region-a
  target: region-b
```

は、PGDGroupCleanup を使用して `region-b` のメタデータをクリーンアップするサンプルを提供します。クリーンアップは `region-a` から実行されます。`region-b` に属するすべてのノードは、この CR を実行する前に `PARTED` ステータスになる必要があります。

3.2.2 バックアップとリストア

2つのスケジューラーを含むサンプルグループ

```
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: group-example-with-2schedulers
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  pgd:
    parentGroup:
      name: world
```

(次のページに続く)

(前のページからの続き)

```

    create: true
  cnp:
    storage:
      storageClass: csi-hostpath-sc
      size: 1Gi
    connectivity:
      dns:
        additional:
          - domain: alternate.domain
          - domain: my.domain
          hostSuffix: -dc1
      tls:
        mode: verify-ca
        clientCert:
          caCertSecret: client-ca-key-pair
          certManager:
            spec:
              issuerRef:
                name: client-ca-issuer
                kind: Issuer
                group: cert-manager.io
          serverCert:
            caCertSecret: server-ca-key-pair
            certManager:
              spec:
                issuerRef:
                  name: server-ca-issuer
                  kind: Issuer
                  group: cert-manager.io
        backup:
          schedulers:
            # volumeSnapshot scheduler backup
            - method: volumeSnapshot
              immediate: true
              schedule: "0 0 8 ** *"
            # barmanObjectStore scheduler backup
            - method: barmanObjectStore
              immediate: true
              schedule: "0 0 9 ** *"
          configuration:
            # backup configuration for volumeSnapshot
            volumeSnapshot:
              className: csi-hostpath-snapclass
            # backup configuration for barmanObjectStore
            barmanObjectStore:

```

(次のページに続く)

(前のページからの続き)

```

destinationPath: s3://backups/
endpointURL: https://minio-service.minio:9000
s3Credentials:
  accessKeyId:
    name: minio-secrets
    key: ACCESS_KEY_ID
  secretAccessKey:
    name: minio-secrets
    key: ACCESS_SECRET_KEY
wal:
  compression: gzip
  maxParallel: 8
data:
  jobs: 4
  immediateCheckpoint: true

```

は、2つのスケジュールされたバックアップが構成された PGD グループサンプルを提供します。1つは volumeSnapshot を使用し、もう1つは barmanObjectStore を使用します。

``barmanObjectStore`` を使用したバックアップとリストアを含むサンプルグループ

:

```

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: group-example-with-barman-backup
spec:
  instances: 3
  proxyInstances: 2
  witnessInstances: 0
  pgd:
    parentGroup:
      name: world
      create: true
  cnp:
    storage:
      size: 1Gi
    walStorage:
      size: 2Gi
  connectivity:
    dns:
      additional:
        - domain: alternate.domain
        - domain: my.domain
      hostSuffix: -dc1

```

(次のページに続く)

```

tls:
  mode: verify-ca
  clientCert:
    caCertSecret: client-ca-key-pair
    certManager:
      spec:
        issuerRef:
          name: client-ca-issuer
          kind: Issuer
          group: cert-manager.io
  serverCert:
    caCertSecret: server-ca-key-pair
    certManager:
      spec:
        issuerRef:
          name: server-ca-issuer
          kind: Issuer
          group: cert-manager.io
backup:
  schedulers:
    - method: barmanObjectStore
      immediate: true
      schedule: "0 0 9 ** *"
  configuration:
    barmanObjectStore:
      destinationPath: s3://backups/
      endpointURL: http://minio-service.minio:9000
      s3Credentials:
        accessKeyId:
          name: minio-secrets
          key: ACCESS_KEY_ID
        secretAccessKey:
          name: minio-secrets
          key: ACCESS_SECRET_KEY
    wal:
      compression: gzip
      maxParallel: 8
    data:
      jobs: 4
      immediateCheckpoint: true

```

3 ノード PGD グループのバックアップサンプルを提供します。このセットアップには、スケジュールされたバックアップと barmanObjectStore への継続的な WAL アーカイブが含まれます。

:

```

# group-example-with-barman-restore-a is configured to full restore from

# backups in barmanObjectStore, will restore from one of servername

# group-example-with-barman-bak-1, group-example-with-barman-bak-2 or group-example-
→with-barman-bak-3

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
name: group-example-with-barman-restore-a
spec:
instances: 3
proxyInstances: 2
witnessInstances: 0
pgd:
parentGroup:
name: world
create: true
cnp:
storage:
size: 1Gi
walStorage:
size: 2Gi
connectivity:
dns:
additional: - domain: alternate.domain - domain: my.domain
hostSuffix: -dc1
tls:
mode: verify-ca
clientCert:
caCertSecret: client-ca-key-pair
certManager:
spec:
issuerRef:
name: client-ca-issuer
kind: Issuer
group: cert-manager.io
serverCert:
caCertSecret: server-ca-key-pair
certManager:
spec:
issuerRef:
name: server-ca-issuer
kind: Issuer

```

(次のページに続く)

(前のページからの続き)

```

group: cert-manager.io
restore:

# List all the serverNames, so pgdgroup will find a more suitable server to restore

serverNames:
  - group-example-with-barman-backup-1
  - group-example-with-barman-backup-2
  - group-example-with-barman-backup-3
barmanObjectStore:
  destinationPath: s3://backups/
  endpointURL: http://minio-service.minio:9000
  s3Credentials:
    accessKeyId:
      name: minio-secrets
      key: ACCESS_KEY_ID
    secretAccessKey:
      name: minio-secrets
      key: ACCESS_SECRET_KEY
  wal:
    compression: gzip
    maxParallel: 8
  data:
    jobs: 4
    immediateCheckpoint: true

# The restored pgdgroup also configured with backup, so that it can be used for
→future restore

# the backup here use a different serverName

backup:
  schedulers: - method: barmanObjectStore
  immediate: true
  schedule: "0 0 9 \ * * *"
  configuration:
    barmanObjectStore:
      destinationPath: s3://backups/
      endpointURL: http://minio-service.minio:9000
      s3Credentials:
        accessKeyId:
          name: minio-secrets
          key: ACCESS_KEY_ID
        secretAccessKey:
          name: minio-secrets

```

(次のページに続く)

(前のページからの続き)

```

key: ACCESS_SECRET_KEY
wal:
compression: gzip
maxParallel: 8
data:
jobs: 4
immediateCheckpoint: true

-- --

# group-example-with-barman-restore-b is a three-node data group, it is created from
# scratch and configured to join group-example-with-barman-restore-a and group-
→example-with-barman-restore-c

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
name: group-example-with-barman-restore-b
spec:
instances: 3
proxyInstances: 2
witnessInstances: 0
pgd:
parentGroup:
name: world
discovery: - host: group-example-with-barman-restore-a-group - host: group-example-
→with-barman-restore-b-group - host: group-example-with-barman-restore-c-group
cnf:
storage:
size: 1Gi
walStorage:
size: 2Gi
connectivity:
dns:
additional: - domain: alternate.domain - domain: my.domain
hostSuffix: -dc1
tls:
mode: verify-ca
clientCert:
caCertSecret: client-ca-key-pair
certManager:
spec:
issuerRef:
name: client-ca-issuer

```

(次のページに続く)

(前のページからの続き)

```

kind: Issuer
group: cert-manager.io
serverCert:
caCertSecret: server-ca-key-pair
certManager:
spec:
issuerRef:
name: server-ca-issuer
kind: Issuer
group: cert-manager.io

# The restored pgdgroup also configured with backup, so that it can be used for
→future restore

# the backup here use a different serverName

backup:
schedulers: - method: barmanObjectStore
immediate: true
schedule: "0 0 9 \ * * *"
configuration:
barmanObjectStore:
destinationPath: s3://backups/
endpointURL: http://minio-service.minio:9000
s3Credentials:
accessKeyId:
name: minio-secrets
key: ACCESS_KEY_ID
secretAccessKey:
name: minio-secrets
key: ACCESS_SECRET_KEY
wal:
compression: gzip
maxParallel: 8
data:
jobs: 4
immediateCheckpoint: true

- -

# group-example-with-barman-restore-c is a witness group, it will not have any
→instances

# it is configured to join group-example-with-barman-restore-a or group-example-with-
→barman-restore-b

```

(次のページに続く)

(前のページからの続き)

```

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
name: group-example-with-barman-restore-c
spec:
instances: 0
proxyInstances: 0
witnessInstances: 1
pgd:
parentGroup:
name: world
discovery: - host: group-example-with-barman-restore-a-group - host: group-example-
→with-barman-restore-b-group - host: group-example-with-barman-restore-c-group
cnf:
storage:
size: 1Gi
walStorage:
size: 2Gi
connectivity:
dns:
additional: - domain: alternate.domain - domain: my.domain
hostSuffix: -dc1
tls:
mode: verify-ca
clientCert:
caCertSecret: client-ca-key-pair
certManager:
spec:
issuerRef:
name: client-ca-issuer
kind: Issuer
group: cert-manager.io
serverCert:
caCertSecret: server-ca-key-pair
certManager:
spec:
issuerRef:
name: server-ca-issuer
kind: Issuer
group: cert-manager.io

```

は、3つのリージョン PGD グループの復元サンプルを提供します。最初のリージョンでは、グループ `group-example-with-barman-restore-a` がバックアップから復元され、親グループ `world` がこのグループに作成されます。2番目と3番目のリージョンでは、グループ `group-example-with-barman-restore-b`

および `group-example-with-barman-restore-c` がスクラッチから作成され、それぞれ復元されたグループに参加します。

``volumeSnapshot`` を使用したバックアップと復元を行うサンプルグループ

```
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
name: group-example-with-vs-backup
spec:
instances: 2
proxyInstances: 2
witnessInstances: 1
pgd:
parentGroup:
name: world
create: true
cnp:
storage:
storageClass: csi-hostpath-sc
size: 1Gi
walStorage:
storageClass: csi-hostpath-sc
size: 2Gi
tablespaces: - name: ts1
storage:
storageClass: csi-hostpath-sc
size: 1Gi - name: ts2
storage:
storageClass: csi-hostpath-sc
size: 1Gi
connectivity:
dns:
additional: - domain: alternate.domain - domain: my.domain
hostSuffix: -dc1
tls:
mode: verify-ca
clientCert:
caCertSecret: client-ca-key-pair
certManager:
spec:
issuerRef:
name: client-ca-issuer
kind: Issuer
group: cert-manager.io
serverCert:
caCertSecret: server-ca-key-pair
```

(次のページに続く)

(前のページからの続き)

```

certManager:
spec:
issuerRef:
name: server-ca-issuer
kind: Issuer
group: cert-manager.io
backup:
schedulers: - method: volumeSnapshot
immediate: true
schedule: "0 0 8 \ * * *"
configuration:
volumeSnapshot:
className: csi-hostpath-snapclass
barmanObjectStore:
destinationPath: s3://backups/
endpointURL: http://minio-service.minio:9000
s3Credentials:
accessKeyId:
name: minio-secrets
key: ACCESS_KEY_ID
secretAccessKey:
name: minio-secrets
key: ACCESS_SECRET_KEY
wal:
compression: gzip
maxParallel: 8
data:
jobs: 4
immediateCheckpoint: true

```

は、スケジュールされたボリュームスナップショットバックアップと barmanObjectStore への継続的な WAL アーカイブで構成された 3 ノード PGD グループを指定します。

:

```

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
name: group-example-with-vs-restore
spec:
instances: 2
proxyInstances: 2
witnessInstances: 1
pgd:
parentGroup:

```

(次のページに続く)

(前のページからの続き)

```
name: world
create: true
cnp:
storage:
storageClass: csi-hostpath-sc
size: 1Gi
walStorage:
storageClass: csi-hostpath-sc
size: 2Gi
tablespaces: - name: ts1
storage:
storageClass: csi-hostpath-sc
size: 1Gi - name: ts2
storage:
storageClass: csi-hostpath-sc
size: 1Gi
connectivity:
dns:
additional: - domain: alternate.domain - domain: my.domain
hostSuffix: -dc1
tls:
mode: verify-ca
clientCert:
caCertSecret: client-ca-key-pair
certManager:
spec:
issuerRef:
name: client-ca-issuer
kind: Issuer
group: cert-manager.io
serverCert:
caCertSecret: server-ca-key-pair
certManager:
spec:
issuerRef:
name: server-ca-issuer
kind: Issuer
group: cert-manager.io
restore:
volumeSnapshots:
selector:
matchLabels: ## we select all the backups from the group-example-with-vs-backup group
"k8s.pgd.enterprisedb.io/group": group-example-with-vs-backup
barmanObjectStore:
destinationPath: s3://backups/
```

(次のページに続く)

(前のページからの続き)

```

endpointURL: http://minio-service.minio:9000
s3Credentials:
  accessKeyId:
  name: minio-secrets
  key: ACCESS_KEY_ID
  secretAccessKey:
  name: minio-secrets
  key: ACCESS_SECRET_KEY
wal:
  compression: gzip
  maxParallel: 8
  data:
  jobs: 4
  immediateCheckpoint: true

## This is the backup configuration for the restored pgdgroup

backup:
  configuration:
  volumeSnapshot:
  className: csi-hostpath-snapclass
  barmanObjectStore:
  destinationPath: s3://backups/
  endpointURL: http://minio-service.minio:9000
  s3Credentials:
  accessKeyId:
  name: minio-secrets
  key: ACCESS_KEY_ID
  secretAccessKey:
  name: minio-secrets
  key: ACCESS_SECRET_KEY
  wal:
  compression: gzip
  maxParallel: 8
  data:
  jobs: 4
  immediateCheckpoint: true

```

: ボリュームスナップショットバックアップからの完全な復元を定義します。

:

```

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:

```

(次のページに続く)

(前のページからの続き)

```
name: group-example-with-vs-pitr
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  pgd:
    parentGroup:
      name: world
      create: true
  cnp:
    storage:
      storageClass: csi-hostpath-sc
      size: 1Gi
    walStorage:
      storageClass: csi-hostpath-sc
      size: 2Gi
    tablespaces:
      - name: ts1
        storage:
          storageClass: csi-hostpath-sc
          size: 1Gi
      - name: ts2
        storage:
          storageClass: csi-hostpath-sc
          size: 1Gi
    connectivity:
      dns:
        additional:
          - domain: alternate.domain
          - domain: my.domain
        hostSuffix: -dc1
      tls:
        mode: verify-ca
        clientCert:
          caCertSecret: client-ca-key-pair
          certManager:
            spec:
              issuerRef:
                name: client-ca-issuer
                kind: Issuer
                group: cert-manager.io
        serverCert:
          caCertSecret: server-ca-key-pair
          certManager:
            spec:
```

(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
restore:
  recoveryTarget:
    targetTime: "2024-09-10 00:20:12.213757"
  volumeSnapshots:
    selector:
      matchLabels:
        ## we select all the backups from the group-example-with-vs-backup group
        "k8s.pgd.enterprisedb.io/group": group-example-with-vs-backup
  barmanObjectStore:
    destinationPath: s3://backups/
    endpointURL: http://minio-service.minio:9000
    s3Credentials:
      accessKeyId:
        name: minio-secrets
        key: ACCESS_KEY_ID
      secretAccessKey:
        name: minio-secrets
        key: ACCESS_SECRET_KEY
  wal:
    compression: gzip
    maxParallel: 8
  data:
    jobs: 4
    immediateCheckpoint: true
## This is the backup configuration for the restored pgdgroup
backup:
  configuration:
    volumeSnapshot:
      className: csi-hostpath-snapclass
  barmanObjectStore:
    destinationPath: s3://backups/
    endpointURL: http://minio-service.minio:9000
    s3Credentials:
      accessKeyId:
        name: minio-secrets
        key: ACCESS_KEY_ID
      secretAccessKey:
        name: minio-secrets
        key: ACCESS_SECRET_KEY
  wal:
    compression: gzip

```

(次のページに続く)

(前のページからの続き)

```

    maxParallel: 8
  data:
    jobs: 4
    immediateCheckpoint: true

```

は、ボリュームスナップショットバックアップからのリストアと、それに続くポイントインタイムリカバリーを定義します。

注釈: 上記の `volumeSnapshot` サンプルでは、`csi-hostpath-sc` ストレージクラスを使用しています。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してください。詳細については、[Backup on volume snapshots](#) を参照してください。

3.2.3 読み取りノードルーティング

読み取りノードルーティングが有効になったサンプルグループ

```

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: group-example-with-readnode-routing
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  pgd:
    proxySettings:
      enableReadNodeRouting: true
  parentGroup:
    name: world
    create: true
  cnp:
    storage:
      size: 1Gi
    connectivity:
      proxyReadServiceTemplate:
        metadata:
          annotations:
            annotationName: "annotationValue"
  dns:
    additional: - domain: alternate.domain - domain: my.domain
  hostSuffix: -dc1
  tls:

```

(次のページに続く)

(前のページからの続き)

```

mode: verify-ca
clientCert:
caCertSecret: client-ca-key-pair
certManager:
spec:
issuerRef:
name: client-ca-issuer
kind: Issuer
group: cert-manager.io
serverCert:
caCertSecret: server-ca-key-pair
certManager:
spec:
issuerRef:
name: server-ca-issuer
kind: Issuer
group: cert-manager.io

```

は、読み取りノードルーティングを有効にしたサンプルを提供します。

3.2.4 LDAP

サンプルグループはバインドと検索で `ldap` を使用します

```

apiVersion: v1
data:
password: YWRtaW4=
kind: Secret
metadata:
name: ldap-bind-password
type: Opaque
--
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
name: group-example-with-ldap-bind-search
spec:
instances: 2
proxyInstances: 2
witnessInstances: 1
pgd:
parentGroup:

```

(次のページに続く)

(前のページからの続き)

```
name: world
create: true
cnp:
postgresql:
ldap:
server: ldap-service.ldap.svc.cluster.local
bindSearchAuth:
baseDN: "dc=example,dc=org"
bindDN: "cn=admin,dc=example,dc=org"
searchAttribute: "uid"
bindPassword:
name: ldap-bind-password
key: password
storage:
size: 1Gi
connectivity:
dns:
additional: - domain: alternate.domain - domain: my.domain
hostSuffix: -dc1
tls:
mode: verify-ca
clientCert:
caCertSecret: client-ca-key-pair
certManager:
spec:
issuerRef:
name: client-ca-issuer
kind: Issuer
group: cert-manager.io
serverCert:
caCertSecret: server-ca-key-pair
certManager:
spec:
issuerRef:
name: server-ca-issuer
kind: Issuer
group: cert-manager.io
backup:
schedulers: - method: barmanObjectStore
immediate: true
schedule: "0 0 9 \ * * *"
configuration:
barmanObjectStore:
destinationPath: s3://backups/
endpointURL: https://minio-service.minio:9000
```

(次のページに続く)

(前のページからの続き)

```
s3Credentials:
accessKeyId:
name: minio-secrets
key: ACCESS_KEY_ID
secretAccessKey:
name: minio-secrets
key: ACCESS_SECRET_KEY
wal:
compression: gzip
maxParallel: 8
data:
jobs: 4
immediateCheckpoint: true
```

は、認証に bind および search を使用するように構成された PGD グループの LDAP サンプルを提供します。

サンプルグループは単純なバインドで LDAP を使用します

:

```
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
name: group-example-with-ldap-simple-bind
spec:
instances: 2
proxyInstances: 2
witnessInstances: 1
pgd:
parentGroup:
name: world
create: true
cnp:
postgresql:
ldap:
server: ldap-service.ldap.svc.cluster.local
bindAsAuth:
prefix: "uid="
suffix: ",dc=example,dc=org"
storage:
size: 1Gi
connectivity:
dns:
additional: - domain: alternate.domain - domain: my.domain
hostSuffix: -dc1
```

(次のページに続く)

(前のページからの続き)

```
tls:
mode: verify-ca
clientCert:
caCertSecret: client-ca-key-pair
certManager:
spec:
issuerRef:
name: client-ca-issuer
kind: Issuer
group: cert-manager.io
serverCert:
caCertSecret: server-ca-key-pair
certManager:
spec:
issuerRef:
name: server-ca-issuer
kind: Issuer
group: cert-manager.io
backup:
schedulers: - method: barmanObjectStore
immediate: true
schedule: "0 0 9 \ * * *"
configuration:
barmanObjectStore:
destinationPath: s3://backups/
endpointURL: https://minio-service.minio:9000
s3Credentials:
accessKeyId:
name: minio-secrets
key: ACCESS_KEY_ID
secretAccessKey:
name: minio-secrets
key: ACCESS_SECRET_KEY
wal:
compression: gzip
maxParallel: 8
data:
jobs: 4
immediateCheckpoint: true
```

は、認証にシンプルバインドを使用する PGD グループの LDAP サンプルを提供します。

3.2.5 管理対象ロール

管理対象ロールを含むサンプルグループ

:

```

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
name: group-example-with-managed
spec:
instances: 2
proxyInstances: 2
witnessInstances: 1
pgd:
parentGroup:
name: world
create: true
cnp:
storage:
size: 1Gi
managed:
roles: - name: foo
comment: Foo
ensure: present
connectionLimit: 100
login: true
superuser: true
createdb: true
createrole: true
replication: true
inherit: true - name: bar
ensure: present
inRoles: - foo
login: true
services:
disabledDefaultServices: - ro - r
connectivity:
dns:
additional: - domain: alternate.domain - domain: my.domain
hostSuffix: -dc1
tls:
mode: verify-ca
clientCert:
caCertSecret: client-ca-key-pair
certManager:

```

(次のページに続く)

(前のページからの続き)

```
spec:
  issuerRef:
    name: client-ca-issuer
    kind: Issuer
    group: cert-manager.io
  serverCert:
    caCertSecret: server-ca-key-pair
  certManager:
    spec:
      issuerRef:
        name: server-ca-issuer
        kind: Issuer
        group: cert-manager.io
```

は、管理対象ロールと管理対象サービスを含む PGD グループサンプルを提供します。

3.2.6 TDE

tde を使用したサンプルグループ

:

```
-- --

apiVersion: v1
kind: Secret
metadata:
  name: tde-key
data:
  key: bG9zcG9sbGl0b3NkaWN1bnBpb3Bpb3Bpb2N1YW5kb3RpZW51bmhhbWJyZW51YW5kb3RpZW51bmZyaW8=

-- --

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: group-example-with-tde
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  imageName: docker.enterprisedb.com/k8s_enterprise_pgd/edb-postgres-advanced-pgd:16.4.
  ↪1-5.5.1
  pgdProxy:
```

(次のページに続く)

(前のページからの続き)

```

imageName: docker.enterprisedb.com/k8s_enterprise_pgd/edb-pgd-proxy:5.5.0
pgd:
parentGroup:
name: world
create: true
cnp:
postgresql:
epas:
tde:
enabled: true
secretKeyRef:
name: tde-key
key: key
storage:
size: 1Gi
connectivity:
dns:
additional: - domain: alternate.domain - domain: my.domain
hostSuffix: -dc1
tls:
mode: verify-ca
clientCert:
caCertSecret: client-ca-key-pair
certManager:
spec:
issuerRef:
name: client-ca-issuer
kind: Issuer
group: cert-manager.io
serverCert:
caCertSecret: server-ca-key-pair
certManager:
spec:
issuerRef:
name: server-ca-issuer
kind: Issuer
group: cert-manager.io

```

は、TDE が有効になった PGD グループサンプルを提供します。TDE では PostgreSQL が特定のフレーバーとバージョンである必要があるため、YAML ファイルには、PGD と PGD プロキシイメージの両方の構成が含まれています。

3.2.7 構成

Mutation を含むサンプルグループ

```

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: group-example-with-mutations
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  pgd:
    parentGroup:
      name: world
      create: true
      mutations: - isApplied: - |
        WITH node_route_config AS (
        SELECT node_name, route_dsn
        FROM bdr.node_routing_config_summary
        )
        SELECT NOT EXISTS(SELECT * FROM node_route_config WHERE regexp_count(route_dsn,
        ↳application_name=) >= 1);
      exec: - |
        WITH node_route_config AS (
        SELECT node_name,
        regexp_replace(route_dsn, concat( application_name=, node_name), ) as route_dsn
        FROM bdr.node_routing_config_summary
        )
        SELECT bdr.alter_node_option(node_name, route_dsn, route_dsn)
        FROM node_route_config;
    cnp:
      storage:
        size: 1Gi
      connectivity:
      dns:
        additional: - domain: alternate.domain - domain: my.domain
      hostSuffix: -dc1
      tls:
        mode: verify-ca
      clientCert:
        caCertSecret: client-ca-key-pair
      certManager:
      spec:
        issuerRef:
          name: client-ca-issuer

```

(次のページに続く)

(前のページからの続き)

```

kind: Issuer
group: cert-manager.io
serverCert:
caCertSecret: server-ca-key-pair
certManager:
spec:
issuerRef:
name: server-ca-issuer
kind: Issuer
group: cert-manager.io

```

: always 変異が構成された PGD グループサンプルを提供します。

サービステンプレートが構成されたサンプルグループ

```

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
name: group-example-service-template
spec:
instances: 3
proxyInstances: 2
pgd:
parentGroup:
create: true
name: world
proxySettings:
enableReadNodeRouting: true
inheritedMetadata:
labels:
region: zeta
annotations:
region.domain.com: region-domain
cnp:
storage:
size: 1Gi
connectivity:
dns:
additional: - domain: alternate.domain - domain: my.domain
hostSuffix: -dc1
tls:
mode: verify-ca
clientCert:
caCertSecret: client-ca-key-pair
certManager:

```

(次のページに続く)

(前のページからの続き)

```

spec:
  issuerRef:
    name: client-ca-issuer
    kind: Issuer
    group: cert-manager.io
  serverCert:
    caCertSecret: server-ca-key-pair
  certManager:
    spec:
      issuerRef:
        name: server-ca-issuer
        kind: Issuer
        group: cert-manager.io
      groupServiceTemplate:
        metadata:
          labels:
            application: test-app
          annotations:
            test.domain.com: domain-value
        nodeServiceTemplate:
          metadata:
            labels:
              application: test-app
            annotations:
              test.domain.com: domain-value
        proxyReadServiceTemplate:
          metadata:
            annotations:
              annotationName: "annotationValue"
        proxyServiceTemplate:
          metadata:
            labels:
              application: test-app
            annotations:
              test.domain.com: domain-value

```

groupServiceTemplate、nodeServiceTemplate proxyServiceTemplate、および proxyReadServiceTemplate を構成したサンプルを提供します。

オペランドイメージがカスタマイズされたサンプルグループ

```

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: group-example-customized

```

(次のページに続く)

(前のページからの続き)

```

spec:
instances: 2
proxyInstances: 2
witnessInstances: 1
imageName: docker.enterprisedb.com/k8s_enterprise_pgd/edb-postgres-extended-pgd:17.4-
→5.7.0-1
imagePullSecrets:

- name: registry-pullsecret
  pgd:
    parentGroup:
      name: world
      create: true
    cnp:
      storage:
        size: 1Gi
    pgdProxy:
      imageName: docker.enterprisedb.com/k8s_enterprise_pgd/edb-pgd-proxy:5.7.0-1
      connectivity:
        dns:
          additional: - domain: alternate.domain - domain: my.domain
          hostSuffix: -dc1
          tls:
            mode: verify-ca
            clientCert:
              caCertSecret: client-ca-key-pair
              certManager:
                spec:
                  issuerRef:
                    name: client-ca-issuer
                    kind: Issuer
                    group: cert-manager.io
              serverCert:
                caCertSecret: server-ca-key-pair
                certManager:
                  spec:
                    issuerRef:
                      name: server-ca-issuer
                      kind: Issuer
                      group: cert-manager.io

```

: pgd およびプロキシイメージ名がカスタマイズされた PGD グループサンプルを提供します。

ImageCatalog を使用してカスタマイズされたオペランド画像を含むサンプルグループ

```

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: ImageCatalog
metadata:
name: postgresql-pgd-catalog
spec:
pgdImages:

- image: docker.enterprisedb.com/k8s_standard_pgd/postgresql-pgd:15.10-5.6.1-1
  major: 15

- image: docker.enterprisedb.com/k8s_standard_pgd/postgresql-pgd:16.6-5.6.1-1
  major: 16
  proxyImage: docker.enterprisedb.com/k8s_standard_pgd/edb-pgd-proxy:5.6.0-ubi9

-- --

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
name: group-example-catalog
spec:
instances: 2
proxyInstances: 2
witnessInstances: 1
imageCatalogRef:
apiGroup: pgd.k8s.enterprisedb.io
kind: ImageCatalog
major: 16
name: postgresql-pgd-catalog
imagePullSecrets:

- name: registry-pullsecret
  pgd:
    parentGroup:
      name: world
      create: true
    cnp:
      storage:
        size: 1Gi
      connectivity:
        dns:
          additional: - domain: alternate.domain - domain: my.domain
      hostSuffix: -dc1

```

(次のページに続く)

(前のページからの続き)

```

tls:
  mode: verify-ca
  clientCert:
    caCertSecret: client-ca-key-pair
  certManager:
    spec:
      issuerRef:
        name: client-ca-issuer
        kind: Issuer
      group: cert-manager.io
  serverCert:
    caCertSecret: server-ca-key-pair
  certManager:
    spec:
      issuerRef:
        name: server-ca-issuer
        kind: Issuer
      group: cert-manager.io

```

: ImageCatalog を使用してカスタマイズされた pgd とプロキシイメージを使用した PGD グループサンプルを提供します。

物理的結合を使用してスケールアップしたサンプルグループ

:

```

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: group-example-with-physical-local-join
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  pgd:
    parentGroup:
      name: world
      create: true
    cnp:
      joinMethod: physical
    storage:
      size: 1Gi
    connectivity:
  dns:
    additional: - domain: alternate.domain - domain: my.domain
  hostSuffix: -dc1

```

(次のページに続く)

(前のページからの続き)

```

tls:
mode: verify-ca
clientCert:
caCertSecret: client-ca-key-pair
certManager:
spec:
issuerRef:
name: client-ca-issuer
kind: Issuer
group: cert-manager.io
serverCert:
caCertSecret: server-ca-key-pair
certManager:
spec:
issuerRef:
name: server-ca-issuer
kind: Issuer
group: cert-manager.io

```

は、物理的結合が構成された PGD グループサンプルを提供します。

使用可能なオプションのリストについては、[API リファレンス](#) を参照してください。

注釈: 上記の PGD グループサンプルでは、自己署名発行者を使用した cert-manager が必要です。使用できません

```

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:

```

(次のページに続く)

(前のページからの続き)

```
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: 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
```

をクリックして、PGD グループを設定する前に自己署名発行者を作成します。

第4章 Using

4.1 Managing EDB Postgres Distributed (PGD) databases

[architecture document](#) で説明しているように、EDB Postgres Distributed for Kubernetes は、PGD データベースを展開するために作成されたオペレーターです。Kubernetes エコシステムを活用することで、自己修復と宣言的制御を提供できます。オペレーターは、バックアップおよび復元操作も担当します。[バックアップとリストア](#) を参照してください。

ただし、PGD クラスターの操作と制御の多くは、オペレーターによって管理されていません。EDB Postgres Distributed for Kubernetes によって作成されたポッドには、[PGD CLI](#) がインストールされた状態で提供されます。このツールを使用して、スイッチオーバーを実行できます。

4.1.1 PGD CLI

警告: PGD CLI を使用してリソースを作成および削除しないでください。たとえば、`create-proxy` および `delete-proxy` コマンドは避けてください。リソースのプロビジョニングはオペレーターの制御下にあり、手動での作成と削除はサポートされていません。

例として、`switchover` コマンドを実行します。

プロキシポッドから PGD CLI を使用することをお勧めします。それらを見つけるには、クラスターのポッドリストを取得します。

```
kubectl get pods -n my-namespace
```

NAME	READY	STATUS	RESTARTS	AGE
location-a-1-1	1/1	Running	0	2h
location-a-2-1	1/1	Running	0	2h
location-a-3-1	1/1	Running	0	2h
location-a-proxy-0	1/1	Running	0	2h
location-a-proxy-1	1/1	Running	0	2h

プロキシノードの名前には `proxy` が含まれます。1 つを選択して、コマンドプロンプトを取得します。

```
kubectl exec -n my-namespace -ti location-a-proxy-0 -- bash
```

これで、プロキシポッドとの `bash` セッションが開きました。 `pgd` コマンドは使用できます。

pgd

Available Commands:

```

check-health      Checks the health of the EDB Postgres Distributed cluster.
<- snipped ->
switchover        Switches over to new write leader.
<- snipped ->

```

スイッチオーバーに必要な情報を取得して、簡単に移動できます。

pgd switchover --help

```

$ pgd switchover --group-name group_a --node-name bdr-a1
switchover is complete

```

pgd show-groups -f \$PGD_CONFIG_FILE

Group	Group ID	Type	Parent Group	Location	Raft	Routing	Write Leader
world	3239291720	global			true	true	location-a-2
location-a	2135079751	data	world		true	true	location-a-1

pgd show-nodes -f \$PGD_CONFIG_FILE

Node	Node ID	Group	Type	Current State	Target State	Status	Seq ID
location-a-1	3165289849	location-a	data	ACTIVE	ACTIVE	Up	1
location-a-2	3266498453	location-a	data	ACTIVE	ACTIVE	Up	2
location-a-3	1403922770	location-a	data	ACTIVE	ACTIVE	Up	3

4.1.2 データベースへのアクセス

Use cases は、Kubernetes クラスター内でのデータベースの使用と外部からのデータベースの使用に関するディスカッションです。*Connectivity* では、アプリケーションからデータベースへのアクセスに関連するサービスに関するディスカッションを見つけることができます。

システムを実装する方法にかかわらず、アプリケーションはプロキシサービスを使用して接続し、PGD と EDB Postgres Distributed for Kubernetes オペレーターによって追加された自己修復機能の強化のメリットを享受する必要があります。

重要: EDB Postgres EDB Postgres Distributed ドキュメントで説明されている `bdrdb` データベースとは対照的に、Kubernetes のデフォルトの EDB Postgres に従って、データノードは `app` と呼ばれるデータベースで作成され、*app* という名前のユーザーが所有します。これらの値は、マニフェストの `cnp` セクションで構成できます。参考については、EDB Postgres for Kubernetes ドキュメントの [Bootstrap](#) を参照してください。

さい。

ただし、管理タスクのために psql CLI を使用して PGD データノードにアクセスすることが必要になる場合があります。

PGD クラスターと `kubectl exec` のポッドリストを取得して、データノードに入れることができます。

```
kubectl exec -n my-namespace -ti location-a-1-1 -- psql
```

psql のよく知られた領域では、デフォルトで作成されるデータベースの名前が `app` であることに注意してください。前述の警告を参照してください。

```
postgres=# \c app
You are now connected to database "app" as user "postgres".
app=# \x
Expanded display is on.
app=# select * from bdr.node_summary;
- [ RECORD 1 ]-----
node_name          | location-a-1
node_group_name    | location-a
interface_connstr  | host=location-a-1-node user=streaming_replica sslmode=verify-
→ca port=5432 sslkey=/controller/certificates/streaming_replica.key sslcert=/
→controller/certificates/streaming_replica.crt sslrootcert=/controller/certificates/
→server-ca.crt application_name=location-a-1 dbname=app
peer_state_name    | ACTIVE
peer_target_state_name | ACTIVE

<- snipped ->
```

アプリケーションには、非特権ロールデフォルトで `app` を使用します。

ユーザーの資格情報が必要です。これは Kubernetes シークレットに保存されています。

```
kubectl get secrets
```

NAME	TYPE	DATA	AGE
<- snipped ->			
location-a-app	kubernetes.io/basic-auth	2	2h

このシークレットには、Postgres DSN に必要な base64 でエンコードされたユーザー名とパスワードが含まれています。

```
kubectl get secrets location-a-app -o yaml
```

```
apiVersion: v1
data:
  password: <base64-encoded-password>
```

(次のページに続く)

(前のページからの続き)

```

username: <base64-encoded-username>
kind: Secret
metadata:
  creationTimestamp: <timestamp>
  labels:

<- snippet ->

```

4.2 Backup on object stores

Kubernetes の EDB Postgres 分散は、オブジェクトストアでの物理バックアップと WAL アーカイブを介して PGD クラスターのオンライン/ホットバックアップをサポートします。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 PITR が利用できることを意味します。

4.2.1 共通オブジェクトストア

AWS S3、Microsoft Azure Blob Storage、Google Cloud Storage、MinIO Gateway、または S3 互換プロバイダーなど、複数のオブジェクトストアがサポートされています。EDB Postgres Distributed for Kubernetes が Kubernetes の EDB Postgres に依存してオブジェクトストアとの接続を構成することを考えると、[EDB Postgres for Kubernetes common object stores for backups](#) を参照してください。

詳細については、ドキュメントを参照してください。

重要: EDB Postgres for Kubernetes ドキュメントのクラウドプロバイダー構成セクションは、`spec.backup.barmanObjectStore` で入手できます。EDB Postgres Distributed for Kubernetes の例では、オブジェクトストアセクションは別のパス `spec.backup.configuration.barmanObjectStore` にあります。

4.2.2 WAL アーカイブ

WAL アーカイブは、WAL ファイルをオブジェクトストレージに送信するプロセスであり、オンライン/ホットバックアップまたは PITR を実行することが重要です。Kubernetes の EDB Postgres 分散では、各 PGD ノードはオブジェクトストアに WAL ファイルを個別にアーカイブするように設定されます。

WAL アーカイブは PGD Group `spec.backup.configuration.barmanObjectStore` スタンザで定義され、宛先パスとクラウド資格情報が設定されるとすぐに有効になります。WAL ファイルはアップロードされる前に圧縮し、暗号化できます。パラレル WAL アーカイブを有効にすることもできます。

```

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
[...]

```

(次のページに続く)

(前のページからの続き)

```
spec:
  backup:
    configuration:
      barmanObjectStore:
        [...]
      wal:
        compression: gzip
        encryption: AES256
        maxParallel: 8
```

詳細については、[EDB Postgres for Kubernetes WAL archiving](#) ドキュメントを参照してください。

4.2.3 スケジュールされたバックアップ

スケジュールされたバックアップは、Kubernetes の EDB Postgres Distributed 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。PGD グループ `spec.backup.configuration.barmanObjectStore` および `spec.backup.schedulers[].schedule` スタンザが構成されている場合、オペレーターは `Scheduled Backup` リソースを作成する選択したバックアップノードとして PGD データノードの 1 つを選択します。

`.spec.backup.schedulers[].method` フィールドでは、スケジュールされたバックアップ方法を定義できます。2 つのバックアップ方法がサポートされています。

- `volumeSnapshot`
- `barmanObjectStore` デフォルト

複数のスケジューラーを定義できますが、各メソッドは 1 つのスケジューラーのみが使用できます。つまり、2 つのスケジューラーが同じメソッドを使用することはできません。

オブジェクトストアのバックアップの場合、デフォルトの `barmanObjectStore` 方法では、スタンザ `spec.backup.configuration.barmanObjectStore` を使用して、バックアップと WAL アーカイブの両方のオブジェクトストア情報を定義します。詳細については、EDB Postgres for Kubernetes ドキュメントの [Backup on object stores](#) を参照してください。

`volumeSnapshot` バックアップを実行するには、`volumeSnapshot` 方法を選択できます。スタンザ `spec.backup.configuration.barmanObjectStore.volumeSnapshot` を使用して、`volumeSnapshot` 構成を定義します。詳細については、EDB Postgres for Kubernetes ドキュメントの [Backup on volume snapshots](#) を参照してください。

この例は、バックアップに `volumeSnapshot` メソッドを使用する方法を示しています。WAL アーカイブは引き続き Barman オブジェクトストアで行われます。

```
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
[...]
spec:
  backup:
```

(次のページに続く)

(前のページからの続き)

```

configuration:
  volumeSnapshot:
    className: csi-hostpath-snapclass
  barmanObjectStore:
    destinationPath: "<destination path here>"
    s3Credentials:
      accessKeyId:
        name: backup-storage-creds
        key: ID
      secretAccessKey:
        name: backup-storage-creds
        key: KEY
    wal:
      compression: gzip
      encryption: AES256
      maxParallel: 8
  schedulers:
  - method: volumeSnapshot
    schedule: "0 0 0 ** *"
    backupOwnerReference: self
    suspend: false
    immediate: true

```

これら2つのバックアップ方法の比較については、EDB Postgres for Kubernetes ドキュメントの *Object stores or volume snapshots* を参照してください。

`.spec.backup.schedulers[].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 cron スケジュールを定義できます。

```

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
[...]
spec:
  backup:
    schedulers:
    - method: barmanObjectStore
      schedule: "0 0 0 ** *"
      backupOwnerReference: self
      suspend: false
      immediate: true

```

必要に応じて、`.spec.backup.schedulers[].suspend` を `true` に設定することにより、スケジュールされたバックアップを一時停止できます。この設定により、新しいバックアップが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実行したい場合は、`.spec.backup.schedulers[].immediate` を `true` に設定します。

`.spec.backupOwnerReference` は、作成したバックアップリソースで使用する `ownerReference` を示します。オプションは次のとおりです。

- **none** — 作成されたバックアップオブジェクトの所有者参照を設定しません。
- **self** — `ScheduledBackup` 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。

警告: `.spec.backup.cron` フィールドは非推奨です。代わりに `.spec.backup.schedulers` を使用してください。 `.spec.backup.cron` は引き続き使用できますが、 `.spec.backup.schedulers` と同時に使用することはできません。

注釈: EDB Postgres for Kubernetes `ScheduledBackup` オブジェクトには、バックアップするクラスターを指定する `cluster` オプションが含まれています。このオプションは現在 EDB Postgres Distributed for Kubernetes ではサポートされておらず、指定されても無視されます。

選択されたバックアップノードが削除される場合、オペレーターは新しいバックアップノードを透過的に選択し、それに応じて `ScheduledBackup` リソースを調整します。

4.2.4 リテンション ポリシー

Kubernetes の EDB Postgres Distributed は、リカバリウィンドウに基づいて保持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。このプロセスは、未使用の WAL ファイルと、削除がスケジュールされているバックアップに関連付けられている WAL の削除も行います。

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: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
[...]
spec:
  backup:
    configuration:
      retentionPolicy: "30d"
```

詳細については、EDB Postgres for Kubernetes ドキュメントの [EDB Postgres for Kubernetes retention policies](#) を参照してください。

重要: 現在、保持ポリシーは、選択された `Backup Node` バックアップと WAL ファイルにのみ適用されます。他の各 PGD ノードも自分の WAL を独立してアーカイブすることを考えると、オブジェクトストレージデータ保持ポリシーを活用するなど、これらの WAL ファイルのライフサイクルを管理するのはあなた

の責任です。また、すべての PGD ノードディレクトリにオブジェクトストレージデータ保持ポリシーを設定している場合、オペレーターが管理する保持ポリシーと重複または干渉していないことを確認します。

4.2.5 圧縮アルゴリズム

バックアップと WAL ファイルはデフォルトで非圧縮です。ただし、複数の圧縮アルゴリズムがサポートされています。詳細については、[EDB Postgres for Kubernetes compression algorithms](#) ドキュメントを参照してください。

4.2.6 バックアップオブジェクトのタグ付け

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。詳細については、[tagging of backup objects](#) に関する EDB Postgres for Kubernetes ドキュメントを参照してください。

4.2.7 PGD ノードのオンデマンドバックアップ

PGD ノードは、Kubernetes Cluster オブジェクトの単一インスタンス EDB Postgres として表されます。そのため、必要に応じて、Kubernetes Backup リソースの EDB Postgres を作成することにより、特定の PGD ノードのオンデマンドバックアップを要求できます。これを行うには、EDB Postgres for Kubernetes ドキュメントの [EDB Postgres for Kubernetes on-demand backups](#) を参照してください。

ヒント: `kubectl get cluster -l k8s.pgd.enterprisedb.io/group=my-pgd-group -n my-namespace` を実行して、PGD グループを構成する Kubernetes クラスターの EDB Postgres のリストを取得できます。

4.3 Recovery

Kubernetes の EDB Postgres Distributed では、PGD ノードの使用可能な物理バックアップから開始して、新しい PGD グループをブートストラップする方法として、リカバリを利用できます。リカバリは、既存の PGD グループでインプレースで実行できません。

Kubernetes の EDB Postgres 分散は、ポイントインタイムリカバリー PITR もサポートしており、カタログで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、任意のポイント・インタイムに PGD グループを復元できます。PITR には WAL アーカイブがあることが必須です。

4.3.1 前提条件

バックアップから復元する前に

- 復旧したクラスターの PostgreSQL 構成 `.spec.cnp.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。
- 新しく作成した名前空間で復旧する場合、復旧した PGD グループを展開する前に、最初に cert-manager CA 発行者を設定します。

詳細については、EDB Postgres for Kubernetes ドキュメントの *EDB Postgres for Kubernetes recovery - Additional considerations* を参照してください。

4.3.2 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レージに保存されている PGD ノードバックアップから復旧できます。

たとえば、バックアップが使用可能な 3 つのインスタンスを持つ `pgdgroup-example` という名前の PGD グループがあると、オブジェクトストレージには各ノードのディレクトリが含まれます。

`pgdgroup-example-1`、`pgdgroup-example-2`、`pgdgroup-example-3`

この例では、オブジェクトストアからの完全なリカバリーを定義します。オペレーターは、定義された `serverNames` から最新のバックアップを透過的に選択し、最後の使用可能な WAL まで再生します。

```
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: pgdgroup-restore
spec:
  [...]
  restore:
    serverNames:
      - pgdgroup-backup-1
      - pgdgroup-backup-2
      - pgdgroup-backup-3
    barmanObjectStore:
      destinationPath: "<destination path here>"
      s3Credentials:
        accessKeyId:
          name: backup-storage-creds
          key: ID
        secretAccessKey:
          name: backup-storage-creds
          key: KEY
      wal:
        compression: gzip
```

(次のページに続く)

(前のページからの続き)

```

encryption: AES256
maxParallel: 8

```

重要: ソースクラスターに従って、WAL セクションを正しく構成してください。この例では、*pgdgroup-example* PGD グループは `compression` および `encryption` を使用しているため、*restore* によって作成されている PGD グループでも適切なパラメーターを設定してください。

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調整してください。必要なときに違いを生みます。

4.3.3 オブジェクトストアからの 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 を再生するのではなく、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。

この例では、リカバリーの時間ベースターゲットを定義します。

```

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: pgdgroup-restore
spec:
  [...]
  restore:
    recoveryTarget:
      targetTime: "2023-08-11 11:14:21.000000+02"
    serverNames:
      - pgdgroup-backup-1
      - pgdgroup-backup-2
      - pgdgroup-backup-3
    barmanObjectStore:
      destinationPath: "<destination path here>"
      s3Credentials:
        accessKeyId:
          name: backup-storage-creds
          key: ID
        secretAccessKey:
          name: backup-storage-creds

```

(次のページに続く)

(前のページからの続き)

```

    key: KEY
wal:
  compression: gzip
  encryption: AES256
  maxParallel: 8

```

重要: PITR では、**リカバリーターゲット** で説明するオプションを使用して、*targetTime* リカバリーターゲットを指定する必要があります。 *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、指定された ``serverNames`` 間で時系列で最後の使用可能なバックアップを選択します。

4.3.4 backupID を指定したオブジェクトストアからのリカバリー

`.spec.restore.recoveryTarget.backupID` オプションを使用すると、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ます。デフォルトでは、この値は空です。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ーのベースとして使用します。値は Barman バックアップ ID の形式である必要があります。

この例では、pgdgroup-backup-1 PGD ノードの特定のバックアップ ID から新しい PGD グループを復旧します。

```

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: pgdgroup-restore
spec:
  [...]
  restore:
    recoveryTarget:
      backupID: 20230824T133000
      serverNames:
        - pgdgroup-backup-1
    barmanObjectStore:
      destinationPath: "<destination path here>"
      s3Credentials:
        accessKeyId:
          name: backup-storage-creds
          key: ID
        secretAccessKey:
          name: backup-storage-creds
          key: KEY
      wal:

```

(次のページに続く)

(前のページからの続き)

```
compression: gzip
encryption: AES256
maxParallel: 8
```

重要: *backupID* を指定する場合、*serverNames* オプションで関連する PGD ノードのみをリストし、他のものはリストしないようにしてください。

注釈: 特定の ``backupID`` の定義は、次のリカバリターゲット *targetName*、*targetXID*、および ``targetImmediate`` のいずれかを使用する場合に特に必要です。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に問題がない場合を除き、*backupID* を指定することが重要です。

4.3.5 volumeSnapshot からのリカバリー

volumeSnapshot バックアップから PGD グループを復旧することもできます。スタンザ `spec.restore.volumeSnapshots` は、volumeSnapshots リストア候補の基準を定義するために使用されます。オペレーターは、候補の中から最新の volumeSnapshot を透過的に選択します。

オペレーターには、volumeSnapshot に次のアノテーション/ラベルが必要です。これらの注釈/ラベルは、オペレーターが volumeSnapshot を撮影した場合、自動的に追加されます。

注釈

- `k8s.enterprisedb.io/backupEndTime` は、最新のスナップショットを比較および選択するために使用されます。
- `k8s.enterprisedb.io/pvcRole` は、volumeSnapshot の pvcRole を表します。サポートされているロールには、PG_WAL および PG_DATA が含まれます。

ラベル

- `k8s.enterprisedb.io/cluster` は、volumeSnapshot が取得されるノードを示します。WAL 再生のオブジェクトストアで *serverName* を取得するために重要です。
- `k8s.enterprisedb.io/backupName` は、volumeSnapshot のバックアップ名です。バックアップでより多くのボリュームが定義されている場合、volumeSnapshot をグループ化するために使用されます。
- `k8s.enterprisedb.io/tablespaceName` は、volumeSnapshot ロールが PG_TABLESPACE の場合、volumeSnapshot のテーブルスペース名を表します。

この例は、volumeSnapshots からの完全なリカバリーを示しています。volumeSnapshot リカバリーの後、完全なリカバリーの WAL 再生はターゲットサーバー `pgdgroup-backup-vs-1` になります。

```
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
```

(次のページに続く)

(前のページからの続き)

```

name: pgdgroup-restore
spec:
  [...]
  restore:
    volumeSnapshots:
      selector:
        matchLabels:
          "k8s.enterprisedb.io/cluster": pgdgroup-backup-vs-1
          "k8s.pgdn.enterprisedb.io/group": pgdgroup-backup-vs
    barmanObjectStore:
      destinationPath: "<destination path here>"
      s3Credentials:
        accessKeyId:
          name: backup-storage-creds
          key: ID
        secretAccessKey:
          name: backup-storage-creds
          key: KEY
      wal:
        compression: gzip
        encryption: AES256
        maxParallel: 8

```

詳細については、EDB Postgres for Kubernetes ドキュメントの [Recovery from volumeSnapshot objects](#) を参照してください。

4.3.6 volumeSnapshot からの PITR

volumeSnapshot 復旧中の特定の瞬間に、先行書き込みログ WAL の再生を停止するように PostgreSQL に指示できます。これは、オブジェクトストアから復旧する場合と同じ機能です。

この例は、ボリュームスナップショットを使用したリカバリーの時間ベースのターゲットの設定を示しています。

```

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: pgdgroup-restore
spec:
  [...]
  restore:
    recoveryTarget:
      targetTime: "2023-08-11 11:14:21.000000+02"
    volumeSnapshots:

```

(次のページに続く)

```

selector:
  matchLabels:
    "k8s.enterprisedb.io/cluster": pgdgroup-backup-vs-1
    "k8s.pgd.enterprisedb.io/group": pgdgroup-backup-vs
barmanObjectStore:
  destinationPath: "<destination path here>"
  s3Credentials:
    accessKeyId:
      name: backup-storage-creds
      key: ID
    secretAccessKey:
      name: backup-storage-creds
      key: KEY
wal:
  compression: gzip
  encryption: AES256
  maxParallel: 8

```

4.3.7 リカバリーターゲット

PITR を超えて、使用できる他の回復ターゲット基準があります。使用可能なすべてのリカバリターゲットの詳細については、EDB Postgres for Kubernetes ドキュメントの *EDB Postgres for Kubernetes recovery targets* を参照してください。

4.3.8 複数のリージョンでの PGD グループのリカバリと作成

バックアップから復旧し、リージョンを超えて参加した複数の PGD グループを作成するには、バックアップから最初のグループ `spec.pgd.parentGroup.create` が `true` に設定されている場合のみを復元できます。他のグループを最初から設定し、最初のグループに参加する必要があります。これは、`bdr.join_node_group` を使用する場合の推奨事項に従います。

新しく参加するデータベースは、BDR 拡張機能を除き空であることをお勧めします。

詳細については、PGD documentation を参照してください。

4.4 Security

EDB Postgres Distributed for Kubernetes のセキュリティは、コード、コンテナ、クラスターの 3 つのレイヤーで分析されます。

警告: ここで説明するセキュリティ対策に加えて、Kubernetes クラスターで通常の InfoSec 業務を実行する必要があります。 [Overview of Cloud Native Security](#) を理解する

Kubernetes ドキュメント

参考:

[The 4C's Security Model in Kubernetes](#) を参照

Kubernetes の EDB Postgres Distributed のセキュリティに対して EDB が採用するアプローチのより良い理解とコンテキストに関するブログ記事。

4.4.1 コード

Kubernetes 用 EDB Postgres 分散のソースコードは、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ています。EDB は、[GolangCI-Lint](#) と呼ばれる Go の人気のオープンソースリンターを CI / CD パイプラインで直接使用します。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 つが [Golang Security Checker](#) または `gosec` です。`gosec` は、コードに隠された既知の脆弱性、脅威、および弱点を検出することを目的としたルールセットと比較して、ソースの抽象構文ツリーをスキャンするリンターです。これらのスレッドには、ハードコーディングされた資格情報、整数オーバーフロー、SQL インジェクションなどが含まれます。

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、Kubernetes 用に配布された EDB Postgres 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。

4.4.2 コンテナ

EDB Postgres Distributed for Kubernetes の一部であるすべてのコンテナイメージは、すべてのコミットに続いて CI / CD パイプラインを介してビルドされます。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 バージョンのイメージが含まれます。パイプラインでは、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
- [Clair](#) 基になるオペレーティングシステムと、それらが実行するライブラリとアプリケーションの両方で見つかった脆弱性

重要: すべてのオペランドイメージは、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必要な場合、パイプラインによって 1 日 1 回再ビルドされ、EDB 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ました。

- [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 EDB Postgres Distributed for Kubernetes](#) を参照してください

EDB Postgres Distributed for Kubernetes のコンテナレベルでセキュリティに対して EDB が採用する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。

4.4.3 クラスター

クラスターレベルのセキュリティでは、コントロールプレーンとノードの両方を形成するすべての Kubernetes コンポーネント、および PostgreSQL を含むクラスターで実行されるアプリケーションを考慮します。

ロールベースのアクセス制御 RBAC

オペレーターは、pgd-operator-controller-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、このアカウントはデフォルトで pgd-operator-system 名前空間にインストールされます。クラスターロールは、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、および動詞のセットを定義する pgd-operator-controller-manager クラスターロールの間をバインドします。

RedHat OpenShift は、[OperatorLifecycleManager \(OLM\)](#) を介してオペレーター RBAC エンティティを直接管理します。OLM では、最小特権の原則を実装して、必要な場合にのみ権限を付与できます。

重要: これらの権限は、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。PGDGroup および `PGDGroupCleanup` リソースとのみ対話するオペレーターのユーザーは、これらに直接アクセスできません。

以下は、いくつかの例と、最も重要な理由です。Kubernetes の EDB Postgres Distributed は、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です。

jobs オペレーターは、ジョブを処理して、さまざまな PGDGroup フェーズを管理する必要があります。

poddisruptionbudgets オペレーターはポッド中断バジェットを使用して、メンテナンス操作中に十分な PGD ノードがアクティブに保たれるようにします。

Pods オペレーターは、PGD ノードを Cluster リソースとして管理する必要があります。

secrets 証明書とパスワードをデータノードに提供しない限り、オペレーターは、ランダムに生成されたパスワードと TLS 証明書を自己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 オペレーターはサービスアカウントを作成して、PGDGroup リカバリジョブが存在するオブジェクトストアからバックアップオブジェクトを取得できるようにする必要があります。

services オペレーターは、アプリケーションから PGD クラスタ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。

statefulsets オペレーターは PGD プロキシを管理する必要があります。

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。

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 pgd-operator-manager-role` を実行できます。

EDB Postgres Distributed for Kubernetes は、EDB Postgres for Kubernetes で定義された Cluster リソースを使用して PGD ノードを内部管理します。[EDB Postgres for Kubernetes documentation](#) を参照してください

Kubernetes オペレーターサービスアカウントの EDB Postgres が使用する権限のリストは。

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ー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、ポッド内から API サーバーにアクセスするための推奨される方法です。詳細は、[Kubernetes documentation](#) を参照してください。

[EDB Postgres for Kubernetes documentation](#) を参照してください

インスタンスマネージャーの詳細については、

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。

EDB Postgres Distributed for Kubernetes 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 権限は必要ありません。Kubernetes プラットフォームまたは管理者が適切な権限を割り当てる必要があります。PostgreSQL コンテナは、読み取り専用のルートファイルシステム、つまり書き込み可能なレイヤーなしで実行されます。

オペレーターは、必要なセキュリティコンテキストを明示的に設定します。

Red Hat OpenShift では、クラウドネイティブ PostgreSQL は、最も制限の厳しい **restricted** セキュリティコンテキスト制約で実行されます。目標は、ポッドの実行を名前空間に割り当てられた UID と SELinux コンテキストに制限することです。

参考:

OpenShift のセキュリティコンテキスト制約 SCC の詳細については、[Managing SCC in OpenShift](#) を参照してください。

記事。

公開ポート

EDB Postgres Distributed for Kubernetes は、表に示すように、オペレーター、インスタスマネージャー、およびオペランドレベルでポートを公開します。

System	Port number	Exposing	Name	Certificates	Authentication
operator	9443	webhook server	<i>webhook-server</i>	TLS	Yes
operator	8080	metrics	<i>metrics</i>	no TLS	No
instance manager	9187	metrics	<i>metrics</i>	no TLS	No
instance manager	8000	status	<i>status</i>	no TLS	No
operand	5432	PostgreSQL instance	<i>postgresql</i>	optional TLS	Yes

PGD

Kubernetes の EDB Postgres Distributed の現在の実装は、postgres スーパーユーザーとデータベース所有者のパスワードを作成します。

パスワードの暗号化に関する限り、Kubernetes の EDB Postgres Distributed 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。以前のバージョンでは、`md5` に設定されています。

重要: [Connection DSNs and SSL](#) を参照

詳細については、PGD ドキュメントを参照してください。

EDB Postgres Distributed for Kubernetes は、`postgres` ロールを使用して PGD ノードを維持します。このため、仕様の `cnp` セクションで `enableSuperuserAccess` から `false` を設定することにより、`postgres` ユーザーアクセスを無効にすることはできません。

ストレージ

EDB Postgres Distributed for Kubernetes は、保存時の暗号化を基になるストレージクラスに委任します。運用環境でのデータ保護のために、保存時の暗号化をサポートするストレージクラスを選択することを強くお勧めします。

4.5 Connectivity

PGD クラスター内の安全なネットワーク通信に関する情報には次のものが含まれます。

- [サービス](#)
- 完全修飾ドメイン名 FQDN を使用した [ドメイン名解決](#)
- [TLS 構成](#)
- [アプリケーションからの接続](#)

4.5.1 サービス

PGD クラスター内のリソースは、Kubernetes サービスを介してアクセスできます。すべての PGD グループは、それらのいくつかを管理します。つまり

- ノードごとに 1 つのサービス、内部通信 `_node service_` に使用されます。
- グループ内の任意のノードに到達する `_group service_`。主に EDB Postgres Distributed for Kubernetes がクラスター内の新しいグループを検出するために使用します。
- アプリケーションが PGD プロキシを使用して透過的にグループの書き込みリーダーに到達できるようにする `_proxy service_`。
- PGD プロキシを透過的に使用して、アプリケーションがグループの `_read nodes_` に到達できるようにする `_proxy-r service_`。このサービスはデフォルトで無効になっており、`.spec.proxySettings.enableReadNodeRouting` 設定によって制御されます。

注釈: `_read nodes_` という用語は、書き込みリーダーとして指定されていないノードを指します。PGD グループでは、各ノードは書き込みを処理できます。書き込みリーダーは、`proxy service` を介して書き込みを処理するように指定された選択されたノードです。一方、`proxy-r service` が到達する `_read nodes_` も書き込み可能ですが、`write SQL` コマンドが `'proxy-r service'` に送信されないことを確認するのはユーザーの責任です。詳細については、[Read-only routing with PGD Proxy](#) を参照してください。

これらのサービスを使用する例については、[アプリケーションからの接続](#)を参照してください。

各サービスは、マニフェストの `.spec.connectivity` セクションのカスタマイズ可能なテンプレートから生成されます。

すべてのサービスは、すべての Kubernetes クラスターのすべての PGD ノードから FQDN を使用して到達可能である必要があります。[ドメイン名解決](#) を参照してください。

EDB Postgres Distributed for Kubernetes は、次のレベルでサービスを簡単にカスタマイズできる可用性を提供するサービステンプレートフレームワークを提供します。

- ノードサービステンプレート 各 PGD ノードは、`.spec.connectivity.nodeServiceTemplate` セクションで構成できるサービスを使用して到達可能です。
- グループサービステンプレート 各 PGD グループには、グループ全体の単一のエントリポイントであり、`.spec.connectivity.groupServiceTemplate` セクションで構成できるグループサービスがあります。
- プロキシサービステンプレート 各 PGD グループには、PGD プロキシを介してグループ書き込みリーダーに到達するプロキシサービスがあり、`.spec.connectivity.proxyServiceTemplate` セクションで構成できます。これは、アプリケーションのエントリポイントサービスです。
- プロキシ読み取りサービステンプレート 各 PGD グループには、PGD プロキシを介してグループ読み取りノードに到達するプロキシサービスがあり、`.spec.proxySettings.enableReadNodeRouting` で有効にでき、`.spec.connectivity.proxyReadServiceTemplate` セクションで構成できます。これは、アプリケーションのエントリポイントサービスです。

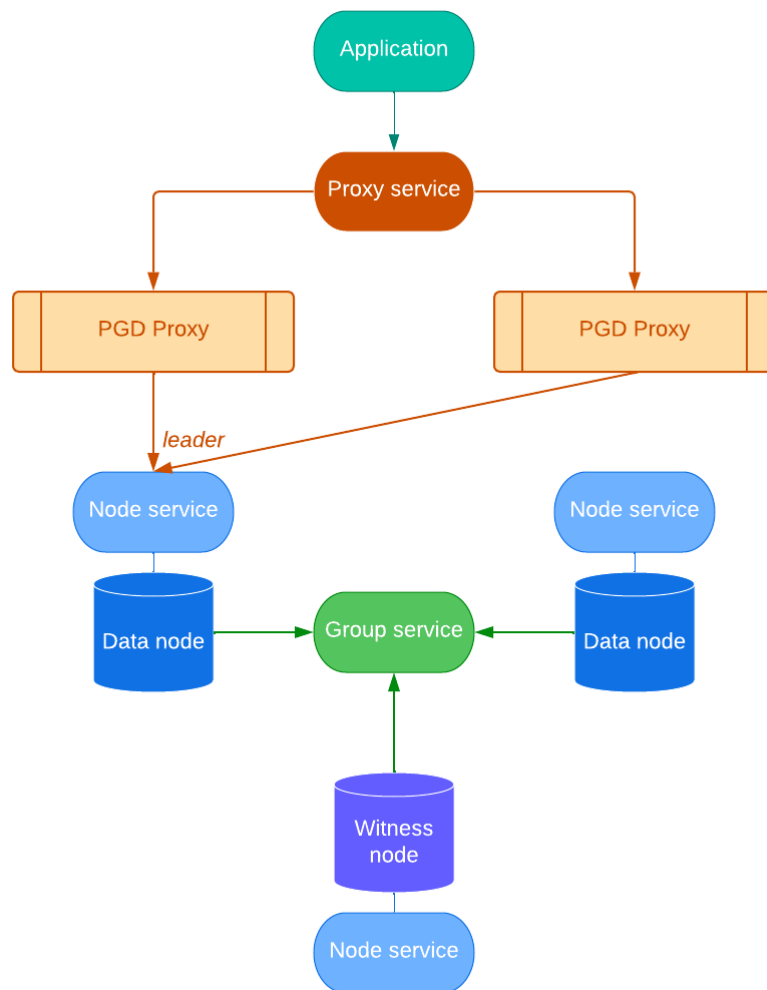


図 1: Basic architecture of an EDB Postgres Distributed for Kubernetes PGD group

テンプレートを使用して、LoadBalancer サービスを作成したり、任意のアノテーションとラベルをサービスに追加して、Kubernetes システムで使用可能な他のコンポーネントと統合したり、外部 DNS 名を作成したり、生成されたロードバランサーを調整したりできます。

次に、LoadBalancer プロキシサービスを作成するプロキシサービステンプレートの例を示します。

```
spec:
  connectivity:
    ...
  proxyServiceTemplate:
    metadata:
      annotations:
        ...
    spec:
      loadBalancerSourceRanges:
        - 0.0.0.0/0
      ports:
        - name: postgres
          port: 5432
          protocol: TCP
          targetPort: 5432
      type: LoadBalancer
      updateStrategy: patch
```

4.5.2 ドメイン名解決

Kubernetes 用 EDB Postgres Distributed は、PGD グループ内のすべてのリソースに FQDN があることを保証し、慣例により、PGD グループ名をそれらのすべてのプレフィックスとして使用します。

その結果、PGD グループのドメイン名を定義することが想定されます。これは、`.spec.connectivity.dns` セクションを介して行うことができます。このセクションは、2つのフィールドでリソースの FQDN を生成する方法を制御します。

- `domain` —使用する PGD グループ内のすべてのオブジェクトのドメイン名必須。
- `hostSuffix` —PGD グループの各サービスに追加するサフィックスオプション。

4.5.3 TLS 構成

EDB Postgres Distributed for Kubernetes では、PGD クラスター内のリソースが安全な接続を介して通信する必要があります。PostgreSQL の [SSL connections](#) のネイティブサポートに依存しています

セキュリティを向上させるために TLS プロトコルを使用してクライアント/サーバー通信を暗号化します。

現在、EDB Postgres Distributed for Kubernetes では、`cert-manager` をインストールする必要があります。Cert-manager は、Kubernetes 環境の標準として広く認識されているため、動的証明書をプロビジョニングするツールとして選択されました。

`spec.connectivity.tls` セクションでは、ノード間の通信がどのように発生するかを説明します。TLS の構成の詳細については、[certificates section](#) を参照してください。

4.5.4 アプリケーションからの接続

同じ Kubernetes クラスター内またはクラスターの外部から実行されているアプリケーションから PGD グループに接続するのは、簡単な手順です。どちらの場合も、`app` ユーザーとして PGD グループのプロキシサービスに接続します。プロキシサービスは、接続しているプロキシサービスに応じて、PGD グループの書き込みリーダーまたは読み取りノードに接続をルーティングする LoadBalancer サービスです。

クラスター内からの接続

クラスター内から接続する場合、プロキシサービス名を使用して PGD グループに接続できます。プロキシサービス名は、PGD グループ名と `-proxy`、および PGDGroup カスタムリソースの `spec.connectivity.dns` セクションで定義されたオプションのホストサフィックスで構成されます。

たとえば、PGD グループ名が `my-group` で、ホストサフィックスが `.my-domain.com` の場合、プロキシサービス名は `my-group-proxy.my-domain.com` になります。

接続する前に、アプリユーザーのシークレットからアプリユーザーのパスワードを取得する必要があります。アプリユーザー名は `spec.pgd.ownerName` によって定義され、アプリユーザーシークレットは `spec.pgd.ownerCredentialsSecret` によって定義されます

次のコマンドを使用して、シークレットからユーザー名とパスワードを取得できます。

```
kubectl get secret <app user secret> -o jsonpath={.data.username} | base64 --decode
kubectl get secret <app user secret> -o jsonpath={.data.password} | base64 --decode
```

これにより、PGD グループへの接続文字列のすべての部分が揃います。

```
postgresql://<app-user>:<app-password>@<proxy-service-name>:5432/<database>
```

または、`psql` 呼び出しの場合

```
psql -U <app-user> -h <proxy-service-name> <database>
```

`app-user` および `app-password` は、シークレットから取得した値であり、`database` は、`spec.pgd.databaseName` で定義された、接続するデータベースの名前です。デフォルトは、アプリユーザーの `app` です。

Kubernetes クラスターの外部からの接続

Kubernetes クラスターの外部から接続する場合、一般的には、[Ingress](#) リソースまたは [load balancer](#) が必要です。環境での動作の詳細については、クラウドプロバイダーまたはローカルインストールを確認してください。

イングレスとロードバランサーでは、接続を PGD プロキシに転送するためにポッドセクターが必要です。それらを構成するときは、次のラベルを使用することをお勧めします。

- `k8s.pgd.enterprisedb.io/group` — PGD グループ名を設定します。
- `k8s.pgd.enterprisedb.io/workloadType` — `pgd-proxy` に設定します。

ローカル開発に Kind または他のソリューションを使用している場合、外部から PGD グループにアクセスする最も簡単な方法は、プロキシサービスへのポート転送を使用することです。次のコマンドを使用して、ローカルマシンのポート 5432 をプロキシサービスに転送できます。

```
kubect1 port-forward svc/my-group.my-domain.com 5432:5432
```

`my-group.my-domain.com` は、前の例のプロキシサービス名です。

4.6 Certificates

Kubernetes の EDB Postgres Distributed 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。PGD クラスターをセットアップするには、各 PGD ノードに以下が必要です。

- サーバー証明書の構成。
- クライアント証明書の構成。

注釈: クラスター PGD ノードのステータスで、各 PGD ノードが使用するすべてのシークレットと有効期限を見つけることができます。

4.6.1 サーバー証明書の構成

サーバー証明書構成は、PGDGroup カスタムリソースの `spec.connectivity.tls.serverCert` セクションで処理されます。この構成では、サーバー CA シークレットと `cert-manager` テンプレートをを使用して、基になる Postgres インスタンスが TLS 接続を終了するために必要な TLS 証明書を生成します。

このセクションが正常に機能するには、次の前提条件を満たす必要があります。

1. `Cert-manager` をインストールする必要があります。
2. `spec.connectivity.tls.serverCert.certManager.issuerRef` で指定された発行者は、`spec.connectivity.dns.domain` で指定されたドメインと `spec.connectivity.tls.serverCert.certManager.spec.dnsNames` にリストされている追加ドメインで利用できます。

3. 発行者が使用する CA のパブリック証明書を含むサーバー CA シークレットを作成する必要があります。

注釈: `spec.connectivity.tls.clientCert.serverCA` で指定されたサーバー CA シークレットは、基になる CNP ノードで ``serverCASecret`` として参照されます。パブリック部分 `ca.crt` は、サーバー証明書を検証し、`sslrootcert` としてクライアント接続文字列に含まれます。プライベート部分 `ca.key` は、自己署名証明書が使用される場合、サーバー SSL 証明書に自動的に署名するために使用されます。

注釈: PGD グループによって生成されたサーバー TLS シークレットは、基になる CNP ノードの ``serverTLSSecret`` の値として指定されます。詳細については、[server certificates](#) を参照してください。

DNS 名

オペレーターは、次の `altDnsNames` をサーバー TLS 証明書に追加します。

```
${nodeName}${hostSuffix}.${domain}
${groupName}${hostSuffix}.${domain}
${proxyName}${hostSuffix}.${domain}
${nodeName}${additionalHostSuffix}.${additionalDomain}
${groupName}${additionalHostSuffix}.${additionalDomain}
${proxyName}${additionalHostSuffix}.${additionalDomain}
```

ユーザーは、基になるネットワークアーキテクチャたとえば、ノードにアクセスするロードバランサーなどに基づいて、`spec.connectivity.tls.serverCert.certManager.spec.dnsNames` に必要な名前を含める責任があります。

たとえば、次のように構成された PGD グループを検討します。

```
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: region-a
spec:
  instances: 3
  ...
  connectivity:
    dns:
      # we need to configure the domain for the group so it could be resolved
      domain: enterprisedb.network
      additional:
        - domain: alternate.network
          hostSuffix: -dc1
```

次のすべてのサブジェクト代替名 SAN はサーバー TLS 証明書に追加されます。

```
region-a-1-node.domain.enterprisedb.network
region-a-group.domain.enterprisedb.network
region-a-proxy.domain.enterprisedb.network
region-a-1-node-dc1.alternate.network
region-a-group-dc1.alternate.network
region-a-proxy-dc1.alternate.network
```

4.6.2 クライアント証明書の構成

クライアント証明書構成は、PGDGroup カスタムリソースの `spec.connectivity.tls.clientCert` セクションで管理されます。この構成では、クライアント CA シークレットとクライアント `cert-manager` テンプレートを使用して、`streaming_replica` Postgres ユーザーのクライアントストリーミングレプリケーション証明書を生成する必要があります。

このセクションが正常に機能するには、次の前提条件を満たす必要があります。

1. `Cert-manager` をインストールする必要があります。
2. `spec.connectivity.tls.clientCert.certManager.issuerRef` で指定された発行者が利用できます。この発行者は、一般名 `streaming_replica` を持つ証明書に署名するために使用されます。
3. クライアント CA シークレットが存在する必要があります。これには、発行者が使用する CA のパブリック証明書が含まれています。

オペレーターは、`spec.connectivity.tls.clientCert.certManager` の下の構成を使用して、`streaming_replica` Postgres ユーザーの証明書要求を作成します。結果の証明書は、ノード間の通信を安全にするために使用されます。

注釈: `spec.connectivity.tls.clientCert.clientCA` で指定されたクライアント CA シークレットは、基になる CNP ノードの ``clientCASecret`` の値として使用されます。パブリック部分 ``ca.crt`` は、`ssl_ca_file` としてすべてのインスタンスに提供され、署名したクライアント証明書を検証できます。プライベート部分 `ca.key` はオプションで、`kubectl cnp plugin` によって生成されたクライアント証明書に署名するために使用できます。

クライアントの事前プロビジョニングされたレプリケーション証明書

または、`spec.connectivity.tls.clientCert.preProvisioned.streamingReplica.secretRef` オプションを使用して、`streaming_replica` ユーザーの事前プロビジョニングされたクライアント証明書を含むシークレットを指定できます。この場合、証明書のライフサイクルは、シークレットの内容を更新するだけで、サードパーティによって手動または自動で完全に管理されます。

注釈: クライアントストリーミングレプリケーション証明書が提供される方法に関係なく、基になる CNP ノードで ``replicationTLSSecret`` の値として使用されます。詳細については、[Client certificate](#) のセクションを参照してください。

4.6.3 TLS モード

`spec.connectivity.tls.mode` を使用して、ノード間の通信中にサーバー証明書を検証する方法を決定する TLS モードを構成できます。デフォルト値は `verify-ca` です。PGD グループが設定されると、TLS モードは変更できないことに注意してください。PostgreSQL ドキュメントの [SSL Support](#) に記載されているように、`mode` は次の値を受け入れます。

- `verify-full`
- `verify-ca`
- `required`

4.6.4 自己署名の例

この例は、`cert-manager` を使用して自己署名 CA をセットアップし、必要な証明書を生成する方法を示しています。

最初に、サーバーとクライアントの CA 証明書を生成する必要があります。ターゲット名前空間に 2 つの自己署名発行者 `server-ca-issuer` および `client-ca-issuer` を作成します。

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

次の証明書オブジェクトを使用して、プライベートキーと発行者から署名された証明書を生成できます。プライベートキーと署名された証明書は、`server-ca-key-pair` および `client-ca-key-pair` という名前のシークレットに保存されます。

```

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: 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

```

PGD グループを構成できるようになりました。 `server-ca-key-pair` および `client-ca-key-pair` は、それぞれサーバー CA シークレットとクライアント CA シークレットとして使用されます。 `cert-manager` テンプレートは、対応する発行者を参照するように設定して、サーバー TLS 証明書とクライアントレプリケーション証明書を自動的に生成できます。

```

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: region-a
spec:
  ...
  connectivity:
    tls:
      mode: verify-ca

```

(次のページに続く)

(前のページからの続き)

```

clientCert:
  caCertSecret: client-ca-key-pair
  certManager:
    spec:
      issuerRef:
        name: client-ca-issuer
        kind: Issuer
        group: cert-manager.io
serverCert:
  caCertSecret: server-ca-key-pair
  certManager:
    spec:
      issuerRef:
        name: server-ca-issuer
        kind: Issuer
        group: cert-manager.io

```

証明書の仕組みの詳細については、[EDB Postgres for Kubernetes documentation](#) を参照してください。

4.7 Client TLS/SSL connections

参考:

[Certificates](#) を参照

EDB Postgres Distributed for Kubernetes 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。

EDB Postgres Distributed for Kubernetes オペレーターは、転送中の暗号化とサーバーサイドとクライアントサイドでの認証の両方で TLS / SSL と連携するように設計されました。PGD ノードは、EDB Postgres for Kubernetes オペレーターを使用してクラスターリソースとして作成されます。これには、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 を展開することが含まれます。

[EDB Postgres for Kubernetes documentation](#) を参照してください

発行者と証明書の詳細については、[こちら](#)をご覧ください。

4.8 Declarative pausing and resuming

宣言的な一時停止と再開機能は、データベース PVC を維持しながらデータベースポッドを削除することにより、CPU パワーを節約できます。

宣言的な一時停止と再開は、Kubernetes の EDB Postgres で利用可能なハイバーネーション機能を活用します。ハイバーネーションの詳細と説明については、[Postgres for Kubernetes documentation on declarative hibernation](#) を参照してください。

目的の PGD グループに `k8s.pgd.enterprisedb.io/pause` アノテーションを追加して、一時停止を要求します。

例

```
kubectl annotate pgdgroup region-a k8s.pgd.enterprisedb.io/pause=on
```

数秒後、要求された PGD グループは一時停止状態になり、すべてのデータベースポッドが削除されます。

```
kubectl get pgdgroups
```

NAME	DATA INSTANCES	WITNESS INSTANCES	PHASE	AGE
region-a	2	1	PGDGroup - Paused	25m
region-b	2	1	PGDGroup - Healthy	25m
region-c	0	1	PGDGroup - Healthy	25m

一時停止した PGD グループを再開するには、アノテーションを `off` に設定します。 `--overwrite` フラグを追加することを忘れないでください。

```
kubectl annotate pgdgroup region-a k8s.pgd.enterprisedb.io/pause=off --overwrite
```

数秒で、ノードが再開を開始し、ポッドが再作成されるのが表示されます。

```
kubectl get pgdgroups
```

NAME	DATA INSTANCES	WITNESS INSTANCES	PHASE	AGE
region-a	2	1	Pause - resume nodes	1m
region-b	2	1	PGDGroup - Healthy	25m
region-c	0	1	PGDGroup - Healthy	25m

一時停止アノテーションが PGD グループを一時停止する前に、いくつかの要件があります。理想的には、PGD グループは正常状態である必要があります。または、PGD グループ内のすべてのデータノードが個々のレベルで正常な場合、一時停止を開始することもできます。

4.9 EDB' s private container registries

EDB Postgres Distributed for Kubernetes と EDB Postgres for Kubernetes オペレーター、およびさまざまなオペランドのイメージは、`docker.enterprisedb.com` の下のプライベートコンテナイメージレジストリに保持されます。

重要: プライベートレジストリへのアクセスには、EDB のアカウントが必要で、有効な [subscription plan](#) を持つ EDB 顧客のために予約されています。資格情報は EDB アカウントを介して実行されます。トライアルについては、[トライアル](#) を参照してください。

4.9.1 どのリポジトリを選択しますか？

Kubernetes の EDB Postgres Distributed は、EDB Enterprise プランまたは EDB Standard プランに加えて、Extreme High Availability アドオンの一部として利用できます。

サブスクリプションプランによっては、Kubernetes の EDB Postgres Distributed は次のリポジトリのいずれかにあります。

Plan	Repository
EDB Standard Plan (standard)	<code>k8s_standard_pgd</code>
EDB EnterpriseDB Plan (enterprise)	<code>k8s_enterprise_pgd</code>

リポジトリの名前は、たとえば `docker login` または kubernetes.io/dockerconfigjson を介して EDB コンテナレジストリにログインしようとするときにユーザー名として使用されます。

重要: 各リポジトリには、プランでアクセスできるすべてのイメージが含まれています。演算子やオペランドイメージなどのさまざまなイメージにアクセスするために別のリポジトリに接続する必要はありません。

4.9.2 トークンの取得方法

[repositories page in EDB](#) には、Repo Token が隠れて表示されている *EDB Repos 2.0* セクションがあります。同じトークンは、Repos 2.0 token というラベルの付いた [Account profile](#) でも使用できます。

トークンの横には、トークンをコピーするボタンと、トークンのコンテンツをクリアテキストとして表示する場合の目のアイコンがあります。トークンは、EDB コンテナレジストリにアクセスしようとするときにパスワードとして使用されます。

`docker login` を使用した例

ターミナルから Docker を使用してログインできます。**Copy Token** を使用してリポトークンをコピーすることをお勧めします。`docker` コマンドは、ユーザー名とパスワードの入力を求めます。

ユーザー名はアクセスしようとしているリポジトリであり、パスワードは先ほどコピーしたトークンです。

```
$ docker login docker.enterprisedb.com
Username: k8s_enterprise_pgd
Password:
Login Succeeded
```

4.9.3 トライアル

トライアルリストまたはプレビューユーザーの場合、リポジトリの名前として `k8s_enterprise_pgd` を使用し、トークンの [トークンの取得方法](#) の指示に従ってください。

4.9.4 オペランドイメージ

オペランドの分布

EDB Postgres Distributed for Kubernetes は、3 つの PostgreSQL ディストリビューションでの EDB Postgres DistributedPGD バージョン 5 の実行をサポートするオペレーターです。

- PostgreSQL
- EDB Postgres Advanced Server
- EDB Postgres Extended PGE

重要: [Choosing a Postgres distribution](#) を参照

詳細と、サポートされているさまざまな PostgreSQL ディストリビューションでの PGD の比較については、PGD ドキュメントを参照してください。

EDB オペレータにおける不変のアプリケーションコンテナの採用のため、オペレータはコンテナイメージに、Postgres の必要なディストリビューションとバージョンに加えて PGD の要求されたバージョンを実行するために必要なすべてのバイナリが含まれることを想定しています。

これらのイメージは、[Container image requirements](#) で説明されている要件と規則に従います。

EDB Postgres for Kubernetes ドキュメントに、`bdr5` 拡張機能を追加します。

この表には、各 Postgres ディストリビューションのイメージ名前の接頭辞が示されています。

Postgres distribution	Versions	Image name	Repositories
PostgreSQL	13 - 17	postgresql-pgd	<i>k8s_standard_pgd,</i> <i>k8s_enterprise_pgd</i>
EDB Postgres Extended	13 - 17	edb-postgres-extended-pgd	<i>k8s_standard_pgd,</i> <i>k8s_enterprise_pgd</i>
EDB Postgres Advanced	13 - 17	edb-postgres-advanced-pgd	<i>k8s_enterprise_pgd</i>

イメージ名の識別

Postgres ディストリビューションに適切な特定のオペランドとプロキシイメージバージョンを選択できます。

オペランドイメージ名

オペランドイメージ名は、Postgres ディストリビューション、EDB Postgres Distributed キーワード、それらのバージョン番号、およびビルドバージョンで構成されます。

Postgres タイプ イメージ名の最初の部分は、使用している Postgres ディストリビューションです。この表は、Postgres タイプがイメージ名のこの部分にどのように対応するかを示しています。

Postgres distribution	Image name
EDB Postgres Advanced Server	<i>edb-postgres-advanced</i>
EDB Postgres Extended Server	<i>edb-postgres-extended</i>
PostgreSQL	<i>postgresql</i>

EDB Postgres Distributed: すべての環境構成で、pgd はイメージ名の次の部分です。

バージョン番号 Postgres ディストリビューションと PGD 拡張機能のバージョンは、ダッシュで区切られ、イメージ名の次にあります。

ビルドバージョン 値は、現在のビルドを反映します。例、1. EDB は、セキュリティパッチを含む新しいビルドを作成します。

これらの識別子は、そのバージョンとともに、オペランドイメージ名を形成します。表にいくつかの例を示します。

Postgres version	EDB Postgres Distributed version	Build	Operand image name
EDB Postgres Advanced 16.6.0	PGD 5.6.1	1	edb-postgres-advanced-pgd:16.6.0-5.6.1-1
EDB Postgres Extended 15.10	PGD 5.6.1	1	edb-postgres-extended-pgd:15.10-5.6.1-1
PostgreSQL 16.6	PGD 5.6.1	1	postgresql-pgd:16.6-5.6.1-1

Postgres の互換性の概要については、[Platform compatibility](#) を参照してください。

プロキシイメージ名

プロキシイメージの名前とバージョンは、EDB Postgres Distributed から取得されます。EDB Postgres Distributed release notes を参照してください

PGD バージョンと対応する PGD プロキシバージョンの概要については、表に一例を示します。

Proxy version	Proxy image name
PGD Proxy 5.6.0	edb-pgd-proxy:5.6.0

4.9.5 オペランド画像のカスタマイズ

EDB Postgres Distributed for Kubernetes オペレーターのデフォルトのオペランドイメージをカスタマイズできます。これらの変更は、将来のすべての PGDGroup 作成に適用されます。さらに、個々の PGDGroup 展開のオペランドをカスタマイズするオプションがあります。

Helm チャート展開のデフォルトのオペランドイメージをカスタマイズ

次の Helm チャートコマンドを使用して、カスタマイズされたオペランドとプロキシ名で EDB Postgres Distributed for Kubernetes オペレーターを展開します。すべてのオペランドとオペレーターイメージは、`global.repository` で定義された同じプライベートリポジトリから取得されることに注意してください。詳細については、[Helm chart](#) ページにアクセスしてください。

```
helm upgrade --dependency-update \
  --install edb-pg4k-pgd \
  --namespace pgd-operator-system \
  --create-namespace \
  edb/edb-postgres-distributed-for-kubernetes \
  --set global.repository=docker.enterprisedb.com/k8s_{{EDB_SUBSCRIPTION_PLAN}}_pgd \
  --set global.pgdImageName={{EDB_PGD_IMAGE_NAME}} \
  --set global.proxyImageName={{EDB_PROXY_IMAGE_NAME}} \
  --set image.imageCredentials.username=k8s_{{EDB_SUBSCRIPTION_PLAN}}_pgd \
  --set image.imageCredentials.password={{EDB_SUBSCRIPTION_TOKEN}}
```

注釈: 上記のインストールでは、デフォルトのオペランドとプロキシイメージがオペレーターの ConfigMap `pgd-operator-controller-manager-config` で構成されます。詳細については、[Operator configuration](#) を参照してください。

次に、カスタマイズの例を示します。

- `enterprise` サブスクリプションプランを使用し、`k8s_enterprise_pgd` リポジトリからイメージをプルします。

- Postgres オプションとして EDB Postgres Advanced Server 15.10.0 を使用します。
- Postgres Distributed バージョンとして PGD 5.6.1 を使用します。
- PGD プロキシバージョンとして 5.6.0 を使用します。

```
helm upgrade --dependency-update \
  --install edb-pg4k-pgd \
  --namespace pgd-operator-system \
  --create-namespace \
  edb/edb-postgres-distributed-for-kubernetes \
  --set global.repository=docker.enterprisedb.com/k8s_enterprise_pgd \
  --set global.pgdImageName=edb-postgres-advanced-pgd:15.10.0-5.6.1-1 \
  --set global.proxyImageName=edb-pgd-proxy:5.6.0 \
  --set image.imageCredentials.username=k8s_enterprise_pgd \
  --set image.imageCredentials.password=<my_registry_token>
```

既存のオペレーターのデフォルトのオペランドイメージの更新

pgd-operator-controller-manager-config ConfigMap または Secret を更新することにより、デフォルトのオペランドイメージを変更できます。詳細については、[Operator configuration](#) を参照してください。

ドキュメント。

単一の PGD グループ展開のオペランドイメージのカスタマイズ

単一の PGDGroup 展開のオペランドとプロキシイメージをカスタマイズすることもできます。この場合、ターゲット名前空間にイメージプルシークレットを作成する必要があります。プライベートレジストリから EDB Postgres Distributed for Kubernetes オペレーターを既にインストールしている場合、イメージプルシークレットを設定する必要があります。

```
kubect1 create secret docker-registry registry-pullsecret \
  -n <CLUSTER-NAMESPACE> --docker-server=docker.enterprisedb.com \
  --docker-username=<REGISTRY-NAME> \
  --docker-password=<TOKEN>
```

前述のように、docker-username は k8s_standard_pgd や k8s_enterprise_pgd などのレジストリ名に対応し、docker-password は [トークンの取得方法](#) から取得したトークンです。

プルシークレットを作成した後、imageName とともに PGD グループマニフェストの imagePullSecrets フィールドを設定していることを確認します。以下のマニフェストは、k8s_enterprise_pgd リポジトリから PG Extended を実行する PGD グループを作成します。

```
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
```

(次のページに続く)

```

  name: group-example-pge
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  imageName: docker.enterprisedb.com/k8s_enterprise_pgd/edb-postgres-extended-pgd:15.
→ 10-5.6.1-1
  imagePullSecrets:
    - name: registry-pullsecret
  pgdProxy:
    imageName: docker.enterprisedb.com/k8s_enterprise_pgd/edb-pgd-proxy:5.6.0
  pgd:
    parentGroup:
      name: world
      create: true
  cnp:
    storage:
      size: 1Gi

```

4.10 Predefined labels

これらの事前定義されたラベルは、EDB Postgres Distributed for Kubernetes オペレーターによって管理されます。

`k8s.pgd.enterprisedb.io/certificateType` 証明書の種類を示します。`replication` は、レプリケーションクライアントの認証に使用される証明書を示します。`server` は、サーバー認証に使用される証明書を示します。

`k8s.pgd.enterprisedb.io/group` リソースが属する PGD グループの名前。クラスターまたはインスタンスリソースに追加されます。

`k8s.pgd.enterprisedb.io/isWitnessService` サービスが監視ノード用であることを示します。

`k8s.pgd.enterprisedb.io/type` クラスターまたはインスタンスリソースに追加されるリソースのタイプ、通常は `node`。

`k8s.pgd.enterprisedb.io/workloadType` クラスターまたはインスタンスリソースに追加されたリソースのワークロードタイプを示します。`pgd-node-data` はデータノードを示します。`pgd-node-witness` 監視ノード。PGD プロキシノードの場合は `pgd-proxy`。`proxy-svc` PGD プロキシサービスの場合。`group-svc` PGD グループサービスが PGDGroup 内のノードと通信する場合。`node-svc` は、CNP サービステンプレートから作成されたサービスです。`scheduled-backup` は `scheduledBackup` リソースに追加されます。`bootstrap-cross-location-pgd-group` は、クロスロケーション PGD グループを作成するポッドに追加されます。`pgd-node-restore` は、ノードの復元プロセスを開始するポッドに追加されます。

4.10.1 事前定義されたアノテーション

`k8s.pgdb.enterprisedb.io/dirtyMetadata` バックアップから生成され、PGD ノードを作成する前にメタデータをクリーンアップする必要がある CNP クラスターに設定します。これは、復元ジョブによって書き込まれます。

`k8s.pgdb.enterprisedb.io/hash` 対応するリソースに更新が必要かどうかを判断するために、`Cluster`、`ScheduledBackup`、`StatefulSet`、`Service (node, group and proxy service)` などのエンティティで使用する PGDGroup 仕様の特定の部分のハッシュを保持します。

`k8s.pgdb.enterprisedb.io/latestCleanupExecuted` PGDGroup で設定され、クリーンアップが実行されたことを示します。

`k8s.pgdb.enterprisedb.io/node` 特定の証明書が生成されたノードの名前が含まれています。証明書リソースに追加されました。

`k8s.pgdb.enterprisedb.io/nodeRestartHash` CNP 構成のハッシュを PGDGroup に保存します。構成を変更した場合は再起動が必要です。

`k8s.pgdb.enterprisedb.io/noFinalizers` ファイナライザーの実行をスキップするように値 `true` を使用して PGDGroup で設定します。内部使用のみ。

`k8s.pgdb.enterprisedb.io/pause` PGDGroup を一時停止するように PGDGroup に設定します。

`k8s.pgdb.enterprisedb.io/recoverabilityPointsByMethod` メソッドごとに CNP クラスターの最初のリカバリアビリティポイントを改ざん防止の場所に保存するように PGDGroup で設定します。

`k8s.pgdb.enterprisedb.io/seedingServer` PGDGroup で設定して、どのサーバーを復元するかをオペレーターに示します。これは、復元ジョブによって書き込まれます。

`k8s.pgdb.enterprisedb.io/seedingSnapshots` PGDGroup で設定して、どのスナップショットを復元するかをオペレーターに示します。これは、復元ジョブによって書き込まれます。

4.11 Managed configuration

PGD オペレーターを使用すると、PG4K クラスターの `managed` セクションを構成できます。 `spec.cnp.managed` スタンザは、クラスター内でサポートされている管理対象ロールとサービスを構成するために使用されます。

この例では、PGDgroup は、デフォルトの `read only` および `read` サービスを無効にして定義されます。さらに、`postgres` で設定された指定されたプロパティを使用して `foo` という名前の管理ロールを持つように構成されます。

```
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: group-example-with-managed
spec:
  [...]
  cnp:
```

(次のページに続く)

```
[...]
managed:
  roles:
    - name: foo
      comment: Foo
      ensure: present
      connectionLimit: 100
      login: true
      superuser: true
      createdb: true
      createrole: true
      replication: true
  services:
    disabledDefaultServices:
      - ro
      - r
```

管理ロールの詳細については、EDB Postgres for Kubernetes ドキュメントの [Database role management](#) を参照してください。

4.12 Monitoring

4.12.1 モニタリングインスタンス

デフォルトでは、PG4K オペレーターは [Prometheus](#) の HTTP/HTTPS メトリックエクスポートを提供します

predefined set of metrics と、高度な構成およびカスタマイズ可能な *user defined metrics* も付属しています。PGD オペレーターは、`spec.cnp.monitoring` セクションのモニタリング設定の一元的な場所を提供します。これは、PGD グループ内のすべてのノードに適用されます。モニタリングの構成の詳細については、*monitoring section* を参照してください。

4.12.2 モニタリングオペレーター

PG4K オペレーターと同様に、PGD オペレーターも `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。

現在、PGD 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。

4.12.3 Openshift でモニタ

詳細については、[Monitoring on OpenShift](#) セクションを参照してください。

4.13 PGDGroup parting

4.13.1 削除とファイナライザー

PGD グループを削除する場合、オペレーターは最初にグループ内のすべてのノードの分割を開始します。アクティブなインスタンスに接続し、ターゲットグループ内のすべてのノードを分割します。ノードが分離されると、レプリケーションとコンセンサス操作に参加しなくなります。ノードが削除される前に正しく分割されることを確認するために、オペレーターは `k8s.pgd.enterprisedb.io/partNodes` ファイナライザーを使用します。状況については、[kubernetes document on finalizers](#) を参照してください。

注釈: PGD グループを保持する名前空間が直接削除される場合、削除と分割シーケンスが正しく実行されることを保証できません。名前空間を削除する前に、含まれるすべての PGD グループを削除することをお勧めします。

4.13.2 制限時間

ノードを分離するとき、オペレーターはアクティブなインスタンスに接続して `bdr.part_node` ファンクションを実行する必要があります。この操作のハングを回避するために、ファイナライザーの時間制限が使用されます。デフォルトでは 300 秒です。時間制限が経過すると、ファイナライザーが削除され、いずれにせよノードが削除されます。グローバル PGD カタログに古いメタデータが残る可能性があります。この時間制限は、秒単位で指定される `spec.failingFinalizerTimeLimitSeconds` を介して構成できます。

4.13.3 ファイナライザーをスキップ

テスト目的のみで、オペレーターはファイナライザー `k8s.pgd.enterprisedb.io/noFinalizers` をスキップするためのアノテーションも提供します。このアノテーションが PGD グループに追加されると、PGD グループが削除されるときにファイナライザーがスキップされ、ノードは PGD クラスタから分離されません。

4.13.4 PGD グループのクリーンアップ

分割されたノードのクリーンアップ

PGDGroup が削除されると、そのメタデータは、`bdr.node_summary` テーブルの `PARTED` 状態でカタログに残ります。PGD4K オペレーターは、PGDGroupCleanup という名前の CRD を定義して、PARTED PGDGroup のクリーンアップに役立ちます。

以下の例では、PGDGroupCleanup は `region-a` からローカルに実行され、すべてのノードが `PARTED` 状態である必要があるという前提条件で、すべてのリージョン `b` をクリーンアップします。

```
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroupCleanup
metadata:
  name: region-b-cleanup
spec:
  executor: region-a
  target: region-b
```

ターゲットグループこの例では `region-b` に `PARTED` 状態にないノードが含まれている場合、グループクリーンアップは `PGDGroupCleanup - Target PGDGroup is not parted, waiting for it to be parted before executing PGDGroupCleanup` フェーズで停止することに注意してください。非常に必要な場合は、`force` オプションを追加できます。

警告: `force` を使用すると、PGD クラスターが一貫性のない状態のままになる場合があります。他の方法でグループノードを分離できない障害から回復するためにのみ使用します。

```
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroupCleanup
metadata:
  name: region-b-cleanup
spec:
  force: true
  executor: region-a
  target: region-b
```

4.14 SQLMutations

SQLMutation は、PGD ノードが PGD グループに参加した後、スーパーユーザーロールを介してアプリケーションデータベースで実行する SQL クエリのリストで構成されます。各 SQLMutation には、`isApplied` クエリーの `isApplied` リストと `exec` クエリーの `exec` リストが含まれます。`isApplied` SQL クエリーは、変更が既に適用されているかどうかを確認するために使用されます。`isApplied` クエリーのいずれかが `false` を返した場合、SQL クエリーの `exec` リストが実行キューに追加されます。

次に、SQL ミューテーションのサンプルを示します。

```
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
[...]
spec:
  pgd:
    mutations:
      - isApplied:
          - |
            WITH node_route_config AS (
              SELECT node_name, route_dsn
              FROM bdr.node_routing_config_summary
            )
            SELECT NOT EXISTS(SELECT * FROM node_route_config WHERE regexp_
↪count(route_dsn, application_name=) >= 1);
      exec:
        - |
          WITH node_route_config AS (
            SELECT node_name,
              regexp_replace(route_dsn, concat( application_name=, node_name),
            ) as route_dsn
            FROM bdr.node_routing_config_summary
          )
          SELECT bdr.alter_node_option(node_name, route_dsn, route_dsn)
          FROM node_route_config;
```

4.14.1 SQL ミューテーションタイプ

この演算子は、spec.pgd.mutations[].type で指定する 3 種類の SQLMutation を提供します。デフォルトは always です。

- beforeSubgroupRaft
- always
- writeLeader

beforeSubgroupRaft および always 変異は、すべての reconcile ループで評価されます。2 つのミューテーションの違いは、実行フェーズにあります。

- always ミューテーションの場合、PGDgroup に制限なしで各リコンサイルループで実行されます。
- beforeSubgroupRaft ミューテーションは、PGD グループにデータノードと PGD プロキシが定義されている場合にのみ、特にサブグループ Raft の準備が整う前に実行されます。

beforeSubgroupRaft および always ミューテーションは両方とも、監視ノードを含む PGD グループ内の任意の PGD ノードで実行できます。したがって、監視ノードにはアプリケーションデータベースデータが含まれないため、アプリケーションデータベースのデータを変更するために使用しないでください。

writeLeader ミューテーションは、書き込みリーダーが選択された後にトリガーおよび実行されます。exec 操作は、書き込みリーダーノードで排他的に実行されます。

4.15 Red Hat OpenShift

EDB Postgres Distributed for Kubernetes は、Web インターフェイスを使用して OpenShift にインストールできる認定オペレーターです。

4.15.1 EDB プライベートレジストリへのアクセスの保証

重要: オペレーターイメージとオペランドイメージの両方が保存されているプライベート EDB リポジトリへのアクセスが必要です。アクセスには、有効な EDB subscription plan が必要です。詳細は、Accessing EDB private image registries を参照してください。

OpenShift インストールは、プルシークレットを使用して、プライベートリポジトリに保持されているオペランドおよびオペレータイメージにアクセスします。

プライベートリポジトリへの資格情報を取得したら、openshift-operators 名前空間に 2 つのプルシークレットを作成する必要があります。

- EDB Postgres Distributed for Kubernetes オペレータイメージの pgd-operator-pull-secret
- postgresql-operator-pull-secret Kubernetes オペレーターイメージの EDB Postgres の場合

oc create コマンドを使用して、各シークレットを作成できます。

```
oc create secret docker-registry pgd-operator-pull-secret \
  -n openshift-operators --docker-server=docker.enterprisedb.com \
  --docker-username="@@REPOSITORY@" \
  --docker-password="@@TOKEN@"

oc create secret docker-registry postgresql-operator-pull-secret \
  -n openshift-operators --docker-server=docker.enterprisedb.com \
  --docker-username="@@REPOSITORY@" \
  --docker-password="@@TOKEN@"
```

そこで

- @@REPOSITORY@ は、[どのリポジトリを選択しますか？](#) で説明したように、リポジトリの名前です。
- @@TOKEN@ は、[トークンの取得方法](#) で説明したように、EDB アカウントのリポジトリトークンです。

4.15.2 オペレーターのインストール

EDB Postgres Distributed for Kubernetes オペレーターは、OpenShift ダッシュボードから Red Hat OperatorHub で直接見つけることができます。

1. ハンバーガーメニューから、**Operators > OperatorHub** を選択します。
2. Web コンソールで、検索ボックスを使用してリストをフィルタリングします。たとえば、EDB または pgd と入力します。

OperatorHub

Discover Operators from the Kubernetes community and Red Hat partners, curated by installation, the Operator capabilities will appear in the [Developer Catalog](#) providing a

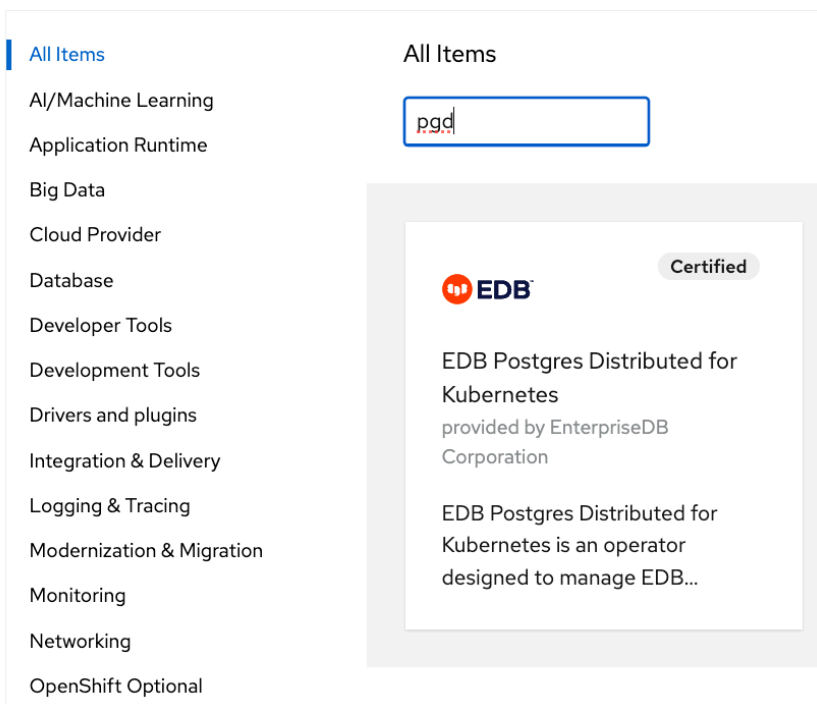


図 2: Install OperatorHub

3. オペレーターに関する情報を読み、**Install** を選択します。
4. Operator Installation ページで、次を選択します。
 - インストールモード。 **クラスター全体のインストール** は、現在唯一のモードです。
 - 更新チャンネル現在 **プレビュー**
 - Red Hat によって認定された、オペレーターの新しいリリースのマーケットプレイスでの入手可能性後の承認戦略
 - **自動**: OLM は、実行中のオペレーターを新しいバージョンにアップグレードします。
 - **マニュアル OpenShift** は、**Installed Operators** セクションでの承認を必要とすることにより、人間の介入を待機します。

クラスター全体のインストール

クラスター全体のインストールでは、オペレーターをデフォルトの `openshift-operators` 名前空間にインストールし、クラスター内のすべてのプロジェクトでできるように OpenShift に指示します。これは、Kubernetes 用の EDB Postgres Distributed をインストールするためのデフォルトであり、通常推奨されるアプローチです。

Web コンソールから、**インストールモード** で、**クラスター上のすべての名前空間(デフォルト)** を選択します。

インストールすると、オペレーターはすべての名前空間で可視されます。インストール中に問題が発生した場合は、他の OpenShift オペレーターの場合と同様に、**Workloads > Pods** ページの `openshift-operators` プロジェクトのポッドのログを確認します。

4.15.3 PGD クラスターの作成

OpenShift によるインストール後、オペレーターのデプロイメントは `openshift-operators` 名前空間にあります。EDB Postgres for Kubernetes オペレーター `postgresql-operator-controller-manager` と同様に、`cert-manager` オペレーターもインストールされたことに注意してください。

```
$ oc get deployments -n openshift-operators
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
cert-manager-operator	1/1	1	1	11m
pgd-operator-controller-manager	1/1	1	1	11m
postgresql-operator-controller-manager-1-20-0	1/1	1	1	23h
...				

`pgd-operator-controller-manager` 展開の準備ができていることを確認した後、PGD クラスターの作成を開始できます。EDB Postgres Distributed for Kubernetes リポジトリには、いくつかの役立つサンプルファイルが含まれています。

PGD クラスターは、専用の名前空間/プロジェクトに展開する必要があります。デフォルトの名前空間は予約されています。

最初に、新しい名前空間を作成し、`self-signed certificate `Issuer <Examples of configuration>`` をデプロイします

その中で

```
oc create ns my-namespace
oc apply -n my-namespace -f \
  https://raw.githubusercontent.com/EnterpriseDB/edb-postgres-for-kubernetes-charts/
  ↪main/hack/samples/issuer-selfsigned.yaml
```

単一リージョンの単一の OpenShift クラスターで PGD を使用する

これで、2つのデータグループと1つの監視グループを含むフレキシブル3リージョンなどのPGDクラスターを展開できるようになりました。ファイルにYAMLマニフェストがあります。

```
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: region-a
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  pgd:
    parentGroup:
      name: world
      create: true
    discovery:
      - host: region-a-group
      - host: region-b-group
      - host: region-c-group
  cnp:
    storage:
      size: 1Gi
  connectivity:
    dns:
      additional:
        - domain: alternate.domain
        - domain: my.domain
      hostSuffix: -dc1
    tls:
      mode: verify-ca
      clientCert:
        caCertSecret: client-ca-key-pair
        certManager:
          spec:
            issuerRef:
              name: client-ca-issuer
              kind: Issuer
              group: cert-manager.io
      serverCert:
        caCertSecret: server-ca-key-pair
        certManager:
          spec:
            issuerRef:
              name: server-ca-issuer
```

(次のページに続く)

(前のページからの続き)

```

        kind: Issuer
        group: cert-manager.io
-- --
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: region-b
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  pgd:
    parentGroup:
      name: world
    discovery:
      - host: region-a-group
      - host: region-b-group
      - host: region-c-group
  cnp:
    storage:
      size: 1Gi
  connectivity:
    tls:
      mode: verify-ca
      clientCert:
        caCertSecret: client-ca-key-pair
        certManager:
          spec:
            issuerRef:
              name: client-ca-issuer
              kind: Issuer
              group: cert-manager.io
      serverCert:
        caCertSecret: server-ca-key-pair
        certManager:
          spec:
            issuerRef:
              name: server-ca-issuer
              kind: Issuer
              group: cert-manager.io
-- --
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: region-c

```

(次のページに続く)

(前のページからの続き)

```
spec:
  instances: 0
  proxyInstances: 0
  witnessInstances: 1
  pgd:
    parentGroup:
      name: world
    discovery:
      - host: region-a-group
      - host: region-b-group
      - host: region-c-group
  cnp:
    storage:
      size: 1Gi
  connectivity:
    tls:
      mode: verify-ca
      clientCert:
        caCertSecret: client-ca-key-pair
        certManager:
          spec:
            issuerRef:
              name: client-ca-issuer
              kind: Issuer
              group: cert-manager.io
      serverCert:
        caCertSecret: server-ca-key-pair
        certManager:
          spec:
            issuerRef:
              name: server-ca-issuer
              kind: Issuer
              group: cert-manager.io
```

```
oc apply -f flexible_3regions.yaml -n my-namespace
```

PGD グループが表示され始めます。

```
$ oc get pgdgroups -n my-namespace
```

NAME	DATA INSTANCES	WITNESS INSTANCES	PHASE	PHASE DETAILS
→AGE				
region-a	2	1	PGDGroup - Healthy	
→23m				
region-b	2	1	PGDGroup - Healthy	
→23m				

(次のページに続く)

(前のページからの続き)

```
region-c  0          1          PGDGroup - Healthy
→23m
```

複数のリージョンの複数の OpenShift クラスターで PGD を使用する

複数のリージョンの複数の OpenShift クラスターに PGD をデプロイするには、最初に PGD グループが相互に通信する方法を確立する必要があります。複数の OpenShift クラスターでこれを実現する推奨方法は、*Submariner* を使用することです。接続の構成は、このドキュメントの範囲外です。ただし、OpenShift クラスター間の接続を確立すると、互いに同期した PGD グループを展開できます。

警告: この例では、各 OpenShift クラスターに 1 つずつ、3 つの PGD グループを展開し、Submariner を使用して OpenShift クラスター間の接続を確立していることを前提としています。

単一リージョンの単一の OpenShift クラスターで PGD を使用すると同様に、この例では、2 つのデータ PGD グループと 1 つの監視グループを作成します。その例とは対照的に、各グループは別の OpenShift クラスターに存在します。

OpenShift クラスター間の基本的な接続に加えて、各 OpenShift クラスターに、他の OpenShift クラスターによって信頼される認証局が含まれていることを確認する必要があります。この条件は、PGD グループが相互に通信するために必要です。

OpenShift クラスターはすべて同じ認証局を使用するか、各クラスターが独自の認証局を持つことができます。いずれの場合も、各 OpenShift クラスターの証明書が他の OpenShift クラスターの認証局を信頼していることを確認する必要があります。

この例では、すべての OpenShift クラスターのすべての証明書に使用される単一の認証局を持つ自己署名証明書を使用します。

この例では、OpenShift クラスターを OpenShift Cluster A、OpenShift Cluster B、および OpenShift Cluster C として参照します。OpenShift では、OperatorHub からの EDB Postgres Distributed for Kubernetes オペレーターのインストールに、cert-manager オペレーターのインストールが含まれます。cert-manager で証明書を作成および管理することをお勧めします。

1. OpenShift Cluster A を保持する名前空間を作成し、その中に自己署名証明書に必要なオブジェクトも作成します。PGD オペレーターと cert-manager がインストールされていると仮定して、self-signed certificate `Issuer <Examples of configuration>`

その名前空間で。

```
oc create ns pgd-group
oc apply -n pgd-group -f \
  https://raw.githubusercontent.com/EnterpriseDB/edb-postgres-for-kubernetes-charts/
  →main/hack/samples/issuer-selfsigned.yaml
```

1. しばらくして、cert-manager は発行者と証明書を作成します。pgd-group 名前空間には、server-ca-key-pair および client-ca-key-pair の 2 つのシークレットがあります。これらの

シークレットには、サーバーおよびクライアント認証局の証明書とプライベートキーが含まれています。issuer-selfsigned.yaml マニフェストを適用する前に、これらのシークレットを他の OpenShift クラスターにコピーする必要があります。oc get secret コマンドを使用して、シークレットの内容を取得できます。

```
oc get secret server-ca-key-pair -n pgd-group -o yaml > server-ca-key-pair.yaml
oc get secret client-ca-key-pair -n pgd-group -o yaml > client-ca-key-pair.yaml
```

1. これらのシークレット uid、リソースバージョン、タイムスタンプなどから OpenShift Cluster A に固有のコンテンツを削除した後、コンテキストを OpenShift Cluster B に切り替えることができます。次に、名前空間を作成し、その中にシークレットを作成して、のみに issuer-selfsigned.yaml ファイルを適用します。

```
oc create ns pgd-group
oc apply -n pgd-group -f server-ca-key-pair.yaml
oc apply -n pgd-group -f client-ca-key-pair.yaml
oc apply -n pgd-group -f \
  https://raw.githubusercontent.com/EnterpriseDB/edb-postgres-for-kubernetes-charts/
  ↪main/hack/samples/issuer-selfsigned.yaml
```

1. コンテキストを OpenShift Cluster C に切り替え、クラスター B で従った同じプロセスを繰り返すことができます。

```
oc create ns pgd-group
oc apply -n pgd-group -f server-ca-key-pair.yaml
oc apply -n pgd-group -f client-ca-key-pair.yaml
oc apply -n pgd-group -f \
  https://raw.githubusercontent.com/EnterpriseDB/edb-postgres-for-kubernetes-charts/
  ↪main/hack/samples/issuer-selfsigned.yaml
```

1. OpenShift Cluster A で、region-a と呼ばれる最初の PGD グループを作成できます。PGD グループの YAML マニフェストは次のとおりです。

```
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: region-a
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  pgd:
    parentGroup:
      name: world
      create: true
    discovery:
      - host: region-a-group.pgd-group.svc.clusterset.local
```

(次のページに続く)

(前のページからの続き)

```

- host: region-b-group.pgd-group.svc.clusterset.local
- host: region-c-group.pgd-group.svc.clusterset.local
cnp:
  storage:
    size: 1Gi
  connectivity:
    dns:
      domain: "pgd-group.svc.clusterset.local"
      additional:
        - domain: alternate.domain
        - domain: my.domain
      hostSuffix: -dc1
    tls:
      mode: verify-ca
      clientCert:
        caCertSecret: client-ca-key-pair
        certManager:
          spec:
            issuerRef:
              name: client-ca-issuer
              kind: Issuer
              group: cert-manager.io
      serverCert:
        caCertSecret: server-ca-key-pair
        certManager:
          spec:
            issuerRef:
              name: server-ca-issuer
              kind: Issuer
              group: cert-manager.io

```

重要: *discovery* セクションのホスト名の形式は、単一クラスターの例とは異なります。これは、Submariner が OpenShift クラスターの接続に使用されており、Submariner が `region-a-group は、*region-a* という名前の PGD グループ用に作成されるサービスの名前です。

1. *region-a* PGD グループ YAML を適用します。

```
oc apply -f region-a.yaml -n pgd-group
```

1. コンテキストを OpenShift Cluster B に切り替え、2 番目の PGD グループを作成できます。クラスター B の PGD グループの YAML は次のとおりです。唯一の違いは `metadata.name` です。

```

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: region-b
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  pgd:
    parentGroup:
      name: world
    discovery:
      - host: region-a-group.pgd-group.svc.clusterset.local
      - host: region-b-group.pgd-group.svc.clusterset.local
      - host: region-c-group.pgd-group.svc.clusterset.local
  cnp:
    storage:
      size: 1Gi
  connectivity:
    dns:
      domain: "pgd-group.svc.clusterset.local"
    tls:
      mode: verify-ca
      clientCert:
        caCertSecret: client-ca-key-pair
        certManager:
          spec:
            issuerRef:
              name: client-ca-issuer
              kind: Issuer
              group: cert-manager.io
      serverCert:
        caCertSecret: server-ca-key-pair
        certManager:
          spec:
            issuerRef:
              name: server-ca-issuer
              kind: Issuer
              group: cert-manager.io

```

1. region-b PGD グループ YAML を適用します。

```
oc apply -f region-b.yaml -n pgd-group
```

1. コンテキストを OpenShift Cluster C に切り替え、3 番目の PGD グループを作成できます。PGD グループの YAML は次のとおりです。

```

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: region-c
spec:
  instances: 0
  proxyInstances: 0
  witnessInstances: 1
  pgd:
    parentGroup:
      name: world
    discovery:
      - host: region-a-group.pgd-group.svc.clusterset.local
      - host: region-b-group.pgd-group.svc.clusterset.local
      - host: region-c-group.pgd-group.svc.clusterset.local
  cnp:
    storage:
      size: 1Gi
  connectivity:
    dns:
      domain: "pgd-group.svc.clusterset.local"
    tls:
      mode: verify-ca
      clientCert:
        caCertSecret: client-ca-key-pair
        certManager:
          spec:
            issuerRef:
              name: client-ca-issuer
              kind: Issuer
              group: cert-manager.io
      serverCert:
        caCertSecret: server-ca-key-pair
        certManager:
          spec:
            issuerRef:
              name: server-ca-issuer
              kind: Issuer
              group: cert-manager.io

```

1. region-c PGD グループ YAML を適用します。

```
oc apply -f region-c.yaml -n pgd-group
```

これで、コンテキストを OpenShift Cluster A に戻し、そこで PGD グループのステータスを確認できます。

```
oc get pgdgroup region-a -n pgd-group
```

PGD グループはフェーズ PGD - Waiting for node discovery にあります。

各 OpenShift クラスターで PGD グループを作成し、各ノードのサービスを作成した後、サービスを他の OpenShift クラスターに公開する必要があります。これはさまざまな方法で行うことができます。

サブマリーナーを使用している場合、`subctl` を使用して行うことができます

コマンド。 `-group` または `-node` サフィックスを持つ `pgd-group` 名前空間のサービスごとに `subctl export service` コマンドを実行します。これは、各クラスターで次の `bash for` ループを実行することにより実行できます。

```
for service in $(oc get svc -n pgd-group --no-headers -o custom-columns="NAME:.
↪metadata.name" | grep -E (-group|-node)$); do
  subctl export service $service -n pgd-group
done
```

数分後、ステータスは PGD グループが正常であることを示します。各 PGD グループが正常な状態になると、2つのデータノード `region-a` または `region-b` のいずれかで `app` データベースに書き込むことができます。データは他のデータノードに複製されます。

4.15.4 チャンネル

EDB Postgres Distributed for Kubernetes オペレーターは、単一の `OLM channel` で利用できます

`v1.0.0` から `stable` という名前になりました。すべてのリリースはこのチャンネルで利用できます。

4.15.5 依存関係

OpenShift の EDB Postgres Distributed for Kubernetes オペレーター PGD4K には、次の依存関係があります。

- Kubernetes オペレーター PG4K 用 EDB Postgres リリース `v1.0.1` 以降では、PGD4K は PG4K の LTS リリースを使用するように制約されます。
- `cert-manager` リリース `v1.1.1` 以降では、Red Hat OpenShift の `cert-manager` オペレーターと Community の `cert-manager` オペレーターの両方がサポートされています。

各リリースでサポートされているバージョンの詳細については、[サポートされているバージョン](#) を参照してください。これらの依存関係が満たされる場合にのみ、PGD4K オペレーターをインストールまたはアップグレードできることに注意してください。

4.16 Transparent data encryption (TDE)

重要: TDE は、それをサポートするオペランドでのみ使用できます。EDB EDB Postgres Advanced Server バージョン 15 以降および EDB Postgres Extended バージョン 15 以降。

透過的データ暗号化 TDE は、いくつかのデータベースベンダーが使用している技術であり、保存データ、つまりディスク上のデータベースファイルを暗号化します。ただし、TDE は使用中のデータを暗号化しません。

TDE は、バージョン 15 以降の EDB Postgres Advanced Server または EDB Postgres Extended に含まれており、EDB Postgres Distributed for Kubernetes でサポートされています。

重要: 続行する前に、時間をかけて *TDE feature in the EPAS documentation* を理解してください。

TDE をアクティブにすると、WAL ファイルとテーブルのファイルの両方が暗号化されます。データの暗号化/復号化は、アプリケーションの変更やクライアントドライバーの更新を必要とせずにデータベースによって管理されるため、ユーザーに対して完全に透過的です。

EDB Postgres Distributed for Kubernetes での TDE のサポートは、EDB Postgres for Kubernetes PG4K からの実装に依存しています。 *the PG4K documentation* を参照

完全なコンテキストについては、

EDB Postgres Advanced Server バイナリキーの暗号化に使用される Kubernetes シークレットに保存されたパスフレーズで TDE を使用できます。

参考:

the EDB Postgres Advanced Server documentation を参照

この暗号化キーの詳細については、

Kubernetes の EDB Postgres Distributed の TDE は、PG4K 実装に依存しています。クラスターで TDE をアクティブにするには、マニフェストの `epas` セクションを使用します。これは、ストレージなどの PG4K レベルの指示子に使用される `cnp` セクションにあります。 `tde` スタンザを使用して TDE を有効にし、TDE 暗号化キーを保持する Kubernetes シークレットの名前を設定します。

次の YAML 部分には、パスフレーズ base-64 エンコードを保持するシークレットと、パスフレーズで TDE をアクティブにする `epas` セクションの両方が含まれています。

```
-- --
apiVersion: v1
kind: Secret
metadata:
  name: tde-key
data:
  key: ↵
  ↵bG9zcG9sbG10b3NkaWN1bnBpb3Bpb3Bpb2N1YW5kb3RpZW51bmhhbWJyZW51YW5kb3RpZW51bmZyaW8=
```

(次のページに続く)

(前のページからの続き)

```

-- --
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
[...]
```

spec:

```

  instances: 3
[...]
```

cnp:

```

  postgresql:
    epas:
      tde:
        enabled: true
        secretKeyRef:
          name: tde-key
          key: key
    storage:
      size: 1Gi
```

再度、[the PG4K documentation](#) を参照してください

暗号化シークレットの作成方法や TDE を使用する追加の方法など、さらなる詳細については。

[TDE feature documentation](#) に示されているように、情報は保存時に暗号化されます。

たとえば、データノードの 1 つで psql ターミナルを開きます。

```
kubectl exec -ti <DATA-NODE> -- psql app
```

テキスト列を含む新しいテーブルを作成します。

```
create table foo(bar int, baz varchar);
insert into foo(bar, baz) values (1, hello), (2, goodbye);
```

新しく定義されたテーブルがディスク上の保存される場所を確認します。

```
select pg_relation_filepath(foo);
pg_relation_filepath
-----
base/16385/16387
```

同じデータノードでターミナルを開くことができます。

```
kubectl exec -ti <DATA-NODE> -- bash
```

そこで、ファイルが暗号化されたことを確認できます。

```
cd $PGDATA/base/16385
hexdump -C 16387 | grep hello
hexdump -C 16387 | grep goodbye
```

4.17 LDAP authentication

Kubernetes の EDB Postgres Distributed は、LDAP 認証をサポートしています。EDB Postgres Distributed for Kubernetes の LDAP 構成は、EDB Postgres for Kubernetes PG4K からの実装に依存しています。[PG4K documentation](#) を参照してください

完全なコンテキストについては。

重要: 続行する前に、[LDAP authentication feature in the Postgres documentation](#) について理解してください。

LDAP サポートでは、ユーザー認証のみが LDAP に送信されるため、ユーザーは postgres データベースに既に存在する必要があります。

この例は、PGDGroup で simple bind モードを使用した LDAP 構成を示しています。Postgres サーバーは、prefix + username + suffix とパスワードを使用して LDAP サーバーをバインドし、認証を実行します。

```
spec:
  [...]
  cnp:
    postgresql:
      ldap:
        server: ldap-service.ldap.svc.cluster.local
        bindAsAuth:
          prefix: "uid="
          suffix: ",dc=example,dc=org"
```

この例は、PGDGroup で search+bind モードを使用して LDAP の構成を示しています。このモードでは、Postgres インスタンスは、最初に bindDN を使用して LDAP にバインドされ、パスワードはシークレット bindPassword に保存されます。次に、Postgres は baseDN の下で検索を実行して、searchAttribute で指定された項目と一致するユーザー名を見つけようとします。一致するものが見つかった場合、Postgres は最終的に LDAP サーバーに対してエントリとパスワードを検証します。

```
spec:
  [...]
  cnp:
    postgresql:
      ldap:
        server: ldap-service.ldap.svc.cluster.local
        bindSearchAuth:
```

(次のページに続く)

(前のページからの続き)

```

baseDN: "dc=example,dc=org"
bindDN: "cn=admin,dc=example,dc=org"
searchAttribute: "uid"
bindPassword:
  name: ldap-bind-password
  key: password

```

4.18 API Reference

パッケージ `v1beta1` には、`pgd v1beta1` API グループの API スキーマ定義が含まれています

4.18.1 Resource Types

- `:ref:ClusterImageCatalog`` リソースを使用したクラスター全体のカタログの例 <*ClusterImageCatalog* リソースを使用したクラスター全体のカタログの例>
- *ImageCatalog* を使用してオペランドイメージを指定する
- *PGDGroup parting*
- *PGDGroupCleanup*

4.18.2 CertificateKeystores

使用箇所

- *CertificateSpec*

`CertificateKeystores` は、証明書の出力シークレットで作成される追加のキーストア出力形式を構成します。

Field	Description
<code>jks _JKSKeystore_</code>	JKS は、JKS キーストアを <code>`spec.secretName` Secret</code> リソースに保存するためのオプションを構成します。
<code>pkcs12 _PKCS12Keystore_</code>	PKCS12 は、PKCS12 キーストアを <code>`spec.secretName` Secret</code> リソースに保存するためのオプションを構成します。

4.18.3 CertificatePrivateKey

使用箇所

- *CertificateSpec*

CertificatePrivateKey には、証明書コントローラが使用するプライベートキーの構成オプションが含まれています。これにより、プライベートキーをローテーションする方法を制御できます。

Field	Description
<i>rotationPolicy</i> _PrivateKeyRotationPolicy_	RotationPolicy は、再発行が処理されているときにプライベートキーを再生成する方法を制御します。Never に設定している場合、プライベートキーはターゲット{" "}" <code>`spec.secretName`</code> にまだ存在しない場合のみ生成されます。存在するが、正しいアルゴリズムまたはサイズがない場合、警告が発生してユーザーの介入を待ちます。Always に設定されている場合、指定された要件と一致するプライベートキーは、再発行が発生するたびに生成されます。下位互換性を維持するために、デフォルトは'Never' です。
<i>encoding</i> _PrivateKeyEncoding_	この証明書のプライベートキーのプライベートキー暗号化標準 PKCS エンコードでエンコードされます。提供される場合、許可される値は `PKCS1` および `PKCS8` で、それぞれ PKCS#1 および PKCS#8 を表します。指定しない場合、デフォルトの `PKCS1`。
<i>algorithm</i> _PrivateKeyAlgorithm_	Algorithm は、この証明書に対応するプライベートキーのプライベートキーアルゴリズムです。提供される場合、許可される値は{" "}" <code>RSA</code> 、 <code>Ed25519</code> または <code>ECDSA</code> のいずれかです。`algorithm` が指定され、`size` が提供されない場合、256 のキーサイズが <code>ECDSA</code> キーアルゴリズムに使用され、2048 のキーサイズが <code>RSA</code> キーアルゴリズムに使用されます。`Ed25519` キーアルゴリズムを使用する場合、キーサイズは無視されます。
<i>size</i> _int_	Size は、この証明書に対応するプライベートキーのキービットサイズです。`algorithm` が <code>RSA</code> に設定されている場合、有効な値は <code>2048</code> 、 <code>4096</code> または {" "}" <code>8192</code> であり、指定しない場合はデフォルトの <code>2048</code> になります。`algorithm` が <code>ECDSA</code> に設定されている場合、有効な値は <code>256</code> 、 <code>384</code> または {" "}" <code>521</code> であり、指定しない場合はデフォルトの <code>256</code> になります。`algorithm` が <code>Ed25519</code> に設定されている場合、Size は無視されます。他の値は許可されません。

4.18.4 CertificateSpec

使用箇所

- *CertManagerTemplate*

CertificateSpec は、証明書の目的の状態を定義します。有効な証明書には、CommonName、DNSName、または URISAN の少なくとも 1 つが有効である必要があります。

Field	Description
<i>subject</i> <i>_X509Subject_</i> <i>commonName</i> <i>_string_</i>	完全な X509 名前仕様 https://golang.org/pkg/crypto/x509/pkix/#Name 。 CommonName は、証明書で使用される一般的な名前です。無効な CSR の生成を回避するには、CommonName の長さが 64 文字以下である必要があります。サブジェクトの代替名が設定されている場合、この値は TLS クライアントによって無視されます。これは x509 動作です。 https://tools.ietf.org/html/rfc6125#section-6.4.4
<i>duration</i> <i>_Duration_</i>	証明書の要求された「期間」つまり有効期間。このオプションは、一部の発行者タイプでは無視/オーバーライドされる場合があります。設定しない場合、これはデフォルトの 90 日です。証明書は、その期間の 2/3、または有効期限が切れる前の `renewBefore` 期間のいずれか遅い方で更新されます。受付される最小期間は 1 時間です。値は Go で受け入れられる単位である必要があります <code>time.ParseDuration</code> https://golang.org/pkg/time/#ParseDuration
<i>renewBefore</i> <i>_Duration_</i>	現在発行されている証明書の有効期限がどれくらい前に、 <code>cert-manager</code> が証明書を更新するか。デフォルトは、発行された証明書の期間の 2/3 です。最小許容値は 5 分です。値は Go で受け入れられる単位である必要があります <code>time.ParseDuration</code> https://golang.org/pkg/time/#ParseDuration
<i>dnsNames</i> <i>_[]string_</i>	DNSNames は、証明書に設定する DNS <code>subjectAltName</code> のリストです。
<i>ipAddresses</i> <i>_[]string_</i>	IPAddresses は、証明書に設定する IP アドレスの <code>subjectAltNames</code> のリストです。
<i>uris</i> <i>_[]string_</i>	URIs は、証明書に設定する URI <code>subjectAltName</code> のリストです。
<i>emailAddresses</i> <i>_[]string_</i>	EmailAddresses は、証明書に設定する電子メールの <code>subjectAltNames</code> のリストです。
<i>secretName</i> [Required] <i>_string_</i>	SecretName は、この証明書リソースによって自動的に作成および管理されるシークレットリソースの名前です。これには、指定された発行者によって署名されたプライベートキーと証明書が入力されます。重要 このフィールドは、元の <code>cert-manager</code> API 宣言では必須でした
<i>keystores</i> <i>_CertificateKeystores_</i>	Keystores は、 <code>secretName</code> Secret リソースに保存される追加のキーストア出力形式を構成します。
<i>issuerRef</i> [Required] <i>_ObjectReference_</i>	IssuerRef は、この証明書の発行者への参照です。{" " } `kind` フィールドが設定されていない場合、または `Issuer` に設定されている場合、証明書と同じ名前空間にある指定された名前の発行者リソースが使用されます。`kind` フィールドが {" " } `ClusterIssuer` に設定されている場合、指定された名前の <code>ClusterIssuer</code> が使用されます。このスタンザの `name` フィールドは常に必要です。
<i>isCA</i> <i>_bool_</i>	IsCA は、この証明書を証明書署名に対して有効なものとしてマークします。これにより、`cert sign` の使用法が {" " } `usages` のリストに自動的に追加されます。
<i>usages</i> <i>_[]KeyUsage_</i> (KeyUsage)	使用法は、証明書のために要求された x509 使用法のセットです。指定しない場合、デフォルトの `digital signature` および {" " } `key encipherment` になります。
<i>privateKey</i> <i>_CertificatePrivateKey_</i>	証明書に使用されるプライベートキーを制御するオプション。
<i>encodeUsagesInRequest</i> <i>_bool_</i>	EncodeUsagesInRequest は、キーの使用法を証明書要求に存在するかどうかを制御します

次のページに続く

表 9 – 前のページからの続き

Field	Description
<i>revisionHistoryLimit</i> _int32_	versionHistoryLimit は、証明書の履歴に維持される CertificateRequest リビジョンの最大数です。各リビジョンは、この証明書が作成、更新、または仕様が変更されたときに作成された単一の `CertificateRequest` を表します。リビジョンの数がこの数を超える場合、リビジョンは古い順に削除されます。設定する場合、revisionHistoryLimit は `1` 以上の値である必要があります。`nil` を解除すると、リビジョンはガベージコレクションされません。デフォルト値は `{ " " }` `nil` です。

4.18.5 ConditionStatus

string のエイリアス

ConditionStatus は、条件のステータスを表します。

4.18.6 JKSKeystore

使用箇所

- *CertificateKeystores*

JKSKeystore は、JKS キーストアを{" " } spec.secretName シークレットリソースに保存するためのオプションを構成します。

Field	Description
<i>create</i> _bool_	[Required] Create は、証明書の JKS キーストアの作成を有効にします。true の場合、`keystore.jks` という名前のファイルがターゲットシークレットリソースに作成され、`passwordSecretRef` に保存されているパスワードを使用して暗号化されます。キーストアファイルは、再発行の場合にのみ更新されます。`truststore.jks` という名前のファイルもターゲットシークレットリソースに作成され、発行認証局を含む `passwordSecretRef` に保存されたパスワードを使用して暗号化されます
<i>passwordSecretRef</i> [Required] _SecretKey-Selector_	PasswordSecretRef は、JKS キーストアの暗号化に使用されるパスワードを含む Secret リソースのキーへの参照です。

4.18.7 KeyUsage

string のエイリアス

使用箇所

- *CertificateSpec*

KeyUsage は、キーの有効な使用コンテキストを指定します。以下を参照してください。 <https://tools.ietf.org/html/rfc5280#section-4.2.1.3>

<https://tools.ietf.org/html/rfc5280#section-4.2.1.12>

有効な KeyUsage 値は次のとおりです。“signing”、“digital 署名”、“content commitment”、“key encipherment”、“key Agreement”、“data encipherment”、“certsign”、“crlsign”、“encipher Only”、“復号のみ”、“any”、“サーバー認証”、“クライアント認証”、“コード署名”、“電子メール保護”、“s/mime”、“ipsec エンドシステム”、“ipsec トンネル”、“ipsec user”、“timestamping”、“ocsp signed”、“microsoft sgc”、“netscape sgc”

4.18.8 LocalObjectReference

使用箇所

- *SecretKeySelector*

LocalObjectReference は、参照対象と同じ名前空間内のオブジェクトへの参照です。参照対象がクラスタースコープのリソース ClusterIssuer である場合、参照は代わりに、構成済みの’クラスターリソース名前空間’で指定された名前を持つリソースを参照します。これは、コントローラコンポーネントのフラグとして設定され、デフォルトで cert-manager が実行される名前空間)。

Field	Description
<i>name</i> _string_	[Required] 参照されるリソースの名前。詳細 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names

4.18.9 ObjectReference

使用箇所

- *CertificateSpec*

ObjectReference は、指定された名前、種類、およびグループを持つオブジェクトへの参照です。

Field	Description
<code>name</code> _string_	[Required] 参照されるリソースの名前。
<code>group</code> _string_	参照されるリソースのグループ。

4.18.10 PKCS12Keystore

使用箇所

- *CertificateKeystores*

PKCS12Keystore は、PKCS12 キーストアを `spec.secretName` Secret リソースに保存するためのオプションを構成します。

Field	Description
<code>create</code> _bool_	[Required] Create は、証明書の PKCS12 キーストアの作成を有効にします。 <code>true</code> の場合、 <code>`keystore.p12`</code> という名前のファイルがターゲットシークレットリソースに作成され、 <code>`passwordSecretRef`</code> に保存されているパスワードを使用して暗号化されます。キーストアファイルは、再発行の場合にのみ更新されます。 <code>`truststore.p12`</code> という名前のファイルもターゲットシークレットリソースに作成され、発行認証局を含む <code>`passwordSecretRef`</code> に保存されたパスワードを使用して暗号化されます
<code>passwordSecretRef</code> [Required] _SecretKey-Selector_	PasswordSecretRef は、PKCS12 キーストアの暗号化に使用されるパスワードを含む Secret リソースのキーへの参照です。

4.18.11 PrivateKeyAlgorithm

`string` のエイリアス

使用箇所

- *CertificatePrivateKey*

PrivateKeyAlgorithm はプライベートキーアルゴリズムを表します

4.18.12 PrivateKeyEncoding

string のエイリアス

使用箇所

- *CertificatePrivateKey*

PrivateKeyEncoding はプライベートキーエンコーディングを表します

4.18.13 PrivateKeyRotationPolicy

string のエイリアス

使用箇所

- *CertificatePrivateKey*

PrivateKeyRotationPolicy は、証明書が発行されるときにプライベートキーを生成またはソースする方法を示します。

4.18.14 SecretKeySelector

使用箇所

- *JKSKeystore*

SecretKeySelector は、Secret リソース内の特定の「キー」への参照です。場合によっては、key は必須フィールドです。

Field	Description
<i>LocalObjectReference</i> <u><i>LocalObjectReference</i></u>	<i>LocalObjectReference</i> のメンバーはこのタイプに埋め込まれます。参照されるシークレットリソースの名前。
<i>key_string</i>	使用する Secret リソースの <code>data</code> フィールドのエントリのキー。このフィールドの一部のインスタンスはデフォルトである場合も、他のインスタンスは必須である場合があります。

4.18.15 X509Subject

使用箇所

- *CertificateSpec*

X509 件名 完全な X509 名前仕様

Field	Description
<i>organizations</i> <code>[]string</code>	証明書で使用する組織。
<i>countries</i> <code>[]string</code>	証明書で使用する国。
<i>organizationalUnits</i> <code>[]string</code>	証明書で使用する組織単位。
<i>localities</i> <code>[]string</code>	証明書で使用する都市。
<i>provinces</i> <code>[]string</code>	証明書で使用する州/地方。
<i>streetAddresses</i> <code>[]string</code>	証明書で使用する住所。
<i>postalCodes</i> <code>[]string</code>	証明書で使用する郵便番号。
<i>serialNumber</i> <code>string</code>	証明書で使用するシリアル番号。

4.18.16 ClusterImageCatalog

ClusterImageCatalog は、clusterimagecatalogs API のスキーマです

Field	Description
<i>apiVersion</i> [Required] <code>string</code>	<i>pgd.k8s.enterprisedb.io/v1beta1</i>
<i>kind</i> [Required] <code>string</code>	<i>ClusterImageCatalog</i>
<i>spec</i> [Required] <i><code>ImageCatalogSpec</code></i>	ClusterImageCatalog の目的の動作の仕様。詳細情報 https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status

4.18.17 ImageCatalog

ImageCatalog は、imagecatalogs API のスキーマです

Field	Description
<i>apiVersion</i> [Required] <code>string</code>	<i>pgd.k8s.enterprisedb.io/v1beta1</i>
<i>kind</i> [Required] <code>string</code>	<i>ImageCatalog</i>
<i>spec</i> [Required] <i><code>ImageCatalogSpec</code></i>	ImageCatalog の目的の動作の仕様。詳細情報 https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status

4.18.18 PGDGroup

PGDGroup は、pgdgroups API のスキーマです

Field		Description
<i>apiVersion</i>	[Required]	<i>pgd.k8s.enterprisedb.io/v1beta1</i>
string		
<i>kind</i>	[Required]	<i>PGDGroup</i>
<i>spec</i>	[Required]	説明はありません。
<i>_PGDGroupSpec_</i>		
<i>status</i>	[Required]	説明はありません。
<i>_PGDGroupStatus_</i>		

4.18.19 PGDGroupCleanup

PGDGroupCleanup は、pgdgroupcleanups API のスキーマです

Field		Description
<i>apiVersion</i>	[Required]	<i>pgd.k8s.enterprisedb.io/v1beta1</i>
string		
<i>kind</i>	[Required]	<i>PGDGroupCleanup</i>
<i>spec</i>	[Required]	説明はありません。
<i>_PGDGroupCleanup-Spec_</i>		
<i>status</i>	[Required]	説明はありません。
<i>_PGDGroupCleanup-Status_</i>		

4.18.20 Backup

使用箇所

- *PGDGroupSpec*

Backup は cnp-pgd ノードのバックアップを構成します

Field	Description
<i>configuration</i> [Required] <i>Bootstrap</i>	バックアップに使用される CNP 構成。ServerName 値は、オペレーターによって予約されています。
<i>cron _ScheduledBackup-Spec_</i>	データのスケジュールされたバックアップ。このフィールドは非推奨であり、将来のバージョンでは削除される予定です。
<i>schedulers</i> [Required] <i>[_]ScheduledBackup-Spec_](ScheduledBackupS</i>	バックアップのスケジューラーを定義します。スケジューラーごとに異なるバックアップ方法があります。`backup.cron`または`{" "}` `backup.schedulers`のいずれか 1 つのみを定義できます。

4.18.21 BackupStatus

使用箇所

- *PGDGroupStatus*

BackupStatus には、pgd バックアップの現在のステータスが含まれます

Field	Description
<i>clusterName</i> [Required] <i>_string_</i>	Cluster バックアップを取るように選択されたクラスターの名前を付けます。バックアップはさまざまな方法でスケジュールできますが、1 つのクラスターのみがバックアップを取得します
<i>scheduledBackupName</i> [Required] <i>_string_</i>	ScheduledBackupName は、スケジュールされたバックアップの名前です。このフィールドは非推奨であり、ScheduledBackupStatus に移行されます
<i>scheduledBackupHash</i> [Required] <i>_string_</i>	ScheduledBackupHash は、スケジュールされたバックアップ構成のハッシュです。このフィールドは非推奨であり、ScheduledBackupStatus に移行されます
<i>scheduledBackups</i> [Required] <i>[_]ScheduledBackupStatus_](ScheduledBackupSta</i>	ScheduledBackupStatus には、スケジュールされたすべてのバックアップのステータスが含まれます

4.18.22 CNPStatus

使用箇所

- *PGDGroupStatus*

CNPStatus には、CNP に関するオペレーターの関連ステータスが含まれています

Field	Description
<i>dataInstances</i> [Required] <i>_int32_</i>	説明はありません。

次のページに続く

表 22 – 前のページからの続き

Field	Description
<i>witnessInstances</i> [Required] <i>_int32_</i>	説明はありません。
<i>clusterStatus</i> [Required] <i>[_[]ClusterStatus_](ClusterStatus)</i>	ClusterStatus には、CNP クラスターのステータスのリストが含まれています
<i>firstRecoverabilityPointsByMethod</i> [Required] <i>[_map[string]RecoverabilityPoints_]</i>	方法別の復旧可能ポイント、CNP ごとにキー付け クラスター Name nolint: lll
<i>firstRecoverabilityPoints</i> [Required] <i>_map[string]string_</i>	RFC3339 形式の日付としての CNP クラスター名前ごとにキー付けられた復旧ポイント
<i>superUserSecretIsPresent</i> [Required] <i>_bool_</i>	説明はありません。
<i>applicationUserSecretIsPresent</i> [Required] <i>_bool_</i>	説明はありません。
<i>podDisruptionBudgetIsPresent</i> [Required] <i>_bool_</i>	説明はありません。

4.18.23 CatalogImage

使用箇所

- *ImageCatalogSpec*

CatalogImage はイメージとメジャーバージョンを定義します

Field	Description
<i>image</i> [Required] <i>_string_</i>	画像参照
<i>major</i> [Required] <i>_int_</i>	イメージの PostgreSQL メジャーバージョン。カタログ内で一意である必要があります。

4.18.24 CertManagerTemplate

使用箇所

- *ClientCertConfiguration*

CertManagerTemplate には、証明書要求を生成するためのデータが含まれています

Field	Description
<i>spec</i> [Required] <i>_CertificateSpec_</i>	証明書オブジェクト仕様
<i>metadata</i> [Required] <i>_Metadata_</i>	ラベルと注釈のメタデータ

4.18.25 ClientCertConfiguration

使用箇所

- *OTELTLSConfiguration*

ClientCertConfiguration には、streaming_replica ユーザーの証明書を生成するための情報が含まれています

Field	Description
<i>caCertSecret</i> [Required] <i>_string_</i>	CACertSecret は、CloudNativePG ClientCASsecret 構成に注入される CA のシークレットです。
<i>certManager</i> [Required] <i>_CertManagerTemplate_</i>	証明書の生成に使用される cert-manager テンプレート
<i>preProvisioned</i> [Required] <i>_ClientPreProvisionedCertificates_</i>	PreProvisioned には、事前生成されたクライアント証明書を取得する方法が含まれています

4.18.26 ClientPreProvisionedCertificates

使用箇所

- *ClientCertConfiguration*

ClientPreProvisionedCertificates は、事前生成されたクライアント証明書を取得する方法を指示します

Field	Description
<i>streamingReplica</i> [Required] <i>_PreProvidedCertificate_</i>	StreamingReplica 'streaming_replica' ユーザーの事前生成された証明書

4.18.27 ClusterStatus

使用箇所

- *CNPStatus*

ClusterStatus には、CNP クラスターの現在のステータスが含まれます

Field	Description
<i>name</i> [Required] <i>_string_</i>	Name は CNP クラスターの名前です
<i>phase</i> [Required] <i>_string_</i>	Phase は、CNP クラスターの現在のフェーズです

4.18.28 CnpBaseConfiguration

使用箇所

- *CnpConfiguration*

CnpBaseConfiguration には、CNP Witness ノードとデータノードの両方に適用できる構成パラメーターが含まれています

Field	Description
<i>startDelay</i> [Required] <i>_int32_</i>	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
<i>stopDelay</i> [Required] <i>_int32_</i>	PostgreSQL インスタンスノード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 180
<i>smartShutdownTimeout</i> <i>_int32_</i>	Postgres のスマートシャットダウンが完了するために予約された時間枠を制御する秒単位の時間。オペレーターが Postgres の高速シャットダウンを要求するための十分な時間を予約してくださいつまり{" "} <i>stopDelay- smartShutdownTimeout</i> 。
<i>storage</i> [Required] <i>_StorageConfiguration_</i>	インスタンスのストレージの構成
<i>walStorage</i> [Required] <i>_StorageConfiguration_</i>	インスタンスの WAL ストレージの構成

次のページに続く

表 28 – 前のページからの続き

Field	Description
<i>clusterMaxStartDelay</i> [Required] <i>_int32_</i>	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 300
<i>affinity</i> <i>_AffinityConfiguration_</i>	ポッドのアフィニティ/アンチアフィニティルール
<i>resources</i> <i>_ResourceRequirements_</i>	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。
<i>postgresql</i> <i>_PostgresConfiguration_</i>	PostgreSQL サーバーの構成
<i>monitoring</i> [Required] <i>Bootstrap</i>	このクラスターの監視インフラストラクチャの構成
<i>logLevel</i> [Required] <i>_string_</i>	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
<i>serviceAccountTemplate</i> [Required] <i>_ServiceAccountTemplate_</i>	CNP に渡されるサービスアカウントテンプレート
<i>otel</i> [Required] <i>_OTEL-Configuration_</i>	OpenTelemetry 構成
<i>postInitSQL</i> <i>_[]string_</i>	ノード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
<i>postInitTemplateSQL</i> <i>_[]string_</i>	ノードが作成された後、{" "} <code>`template1`</code> 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。細心の注意を払って使用しますデフォルトでは空
<i>postInitApplicationSQL</i> <i>_[]string_</i>	クラスターが作成された直後にアプリケーションデータベースでスーパーユーザーとして実行される SQL クエリーのリスト。論理的結合の前にアプリケーションデータベースで作成されたデータが結合に失敗するため、細心の注意を払って使用されますデフォルトでは空
<i>postInitApplicationSQL-Refs</i> <i>_SQLRefs_</i>	クラスターが作成された直後にアプリケーションデータベースでスーパーユーザーとして実行される SQL ファイルを含む ConfigMap または Secret への参照のリスト。参照は特定の順序で処理されます。最初にすべてのシークレットが処理され、次にすべての ConfigMap が処理されます。各グループ内で、処理順序は、それぞれの配列で指定されたシーケンスに従います。細心の注意を払って使用しますデフォルトでは空
<i>postInitTemplateSQL-Refs</i> <i>_SQLRefs_</i>	クラスターが作成された直後に <code>`template1`</code> データベースでスーパーユーザーとして実行される SQL ファイルを含む ConfigMap または Secret への参照のリスト。参照は特定の順序で処理されます。最初にすべてのシークレットが処理され、次にすべての ConfigMap が処理されます。各グループ内で、処理順序は、それぞれの配列で指定されたシーケンスに従います。細心の注意を払って使用しますデフォルトでは空

次のページに続く

表 28 – 前のページからの続き

Field	Description
<code>postInitSQLRefs</code> <code>_SQLRefs_</code>	クラスターが作成された直後に `postgres` データベースでスーパーユーザーとして実行される SQL ファイルを含む ConfigMap または Secret への参照のリスト。参照は特定の順序で処理されます。最初にすべてのシークレットが処理され、次にすべての ConfigMap が処理されます。各グループ内で、処理順序は、それぞれの配列で指定されたシーケンスに従います。細心の注意を払って使用しますデフォルトでは空
<code>seccompProfile</code> [Required] <code>_SeccompProfile_</code>	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 <code>{" "}</code> <i>RuntimeDefault</i>
<code>metadata</code> [Required] <code>_InheritedMetadata_</code>	生成されたクラスターリソースにのみ適用されるメタデータ。AppArmor プロファイルを適用する場合に役立ちます。
<code>managed</code> [Required] <code>_Bootstrap_</code>	CNP インスタンスマネージャーによって管理される PostgreSQL の部分で使われる構成
<code>projectedVolumeTemplate</code> <code>_ProjectedVolumeSource_</code>	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
<code>tablespaces</code> <code>[_]TablespaceConfiguration_</code> (https://www.enterprisedb.com/docs/postgres_for_kubernetes/latest/pg4k.v1/#tablespaceconfiguration)	テーブルスペース構成
<code>topologySpreadConstraints</code> <code>[_]TopologySpreadConstraint_</code> (https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#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/
<code>plugins</code> <code>[_]PluginConfiguration_</code> (https://www.enterprisedb.com/docs/postgres_for_kubernetes/latest/pg4k.v1/#pluginconfiguration)	プラグイン構成。対応する構成でロードされるプラグインを含む

4.18.29 CnpConfiguration

使用箇所

- *PGDGroupSpec*

CnpConfiguration には、PGD グループを構成する結果のクラスターに注入されるデータノードの構成が含まれています

Field	Description
<i>CnpBaseConfiguration</i> <i>_CnpBaseConfigura-</i> <i>tion_</i>	<code>`CnpBaseConfiguration`</code> のメンバーはこのタイプに埋め込まれます。説明はありません。
<i>joinMethod</i> [Required] <i>_JoinMethod_</i>	JoinMethod は、データノードがローカルグループに参加するために使用する方法を指定します。PGD は、ローカルグループに参加する 2 つの方法、論理と物理をサポートしています。論理参加では、ノードは <code>{ " " } `bdr.join_node_group`</code> を使用してグループに参加し、データを最初から同期します。物理的参加では、ノードは <code>{ " " } `bdr_init_physical`</code> を使用してグループに参加し、物理的バックアップからデータを同期します。
<i>enableSuperuserAccess</i> <i>_bool_</i>	このオプションが有効になっている場合、CNP オペレーターは SuperuserSecret で定義されたシークレットを作成または使用して、スーパーユーザー postgres のデータベースへのアクセスを許可します。実行中のグループでこのオプションが無効になっている場合、オペレーターはシークレットの内容を無視し、 <code>`postgres`</code> ユーザーのパスワードを <code>`NULL`</code> に設定します。デフォルトで有効になっています。
<i>superuserSecret</i> <i>_LocalObjectRefer-</i> <i>ence_</i>	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を使用して新しいシークレットが作成されます。このフィールドは、CNP インスタンス構成でのみ許可されます。Witness ノードは常に他のインスタンスと同じスーパーユーザーシークレットを使用します。

4.18.30 ConnectionString

(``map[string]string`` のエイリアス)

使用箇所

- *PgdConfiguration*

ConnectionString は、PostgreSQL クラスターに接続するためのパラメーターを表します

4.18.31 ConnectivityConfiguration

使用箇所

- *PGDGroupSpec*

ConnectivityConfiguration では、PGDGroup のサービスと証明書を生成する方法について説明します

Field	Description
<i>dns</i> [Required] <i>_RootDNSConfiguration_</i>	リソースの FQDN を生成する方法について説明します
<i>tls</i> [Required] <i>_TLSConfiguration_</i>	TLS インフラストラクチャの構成
<i>nodeServiceTemplate</i> [Required] <i>_ServiceTemplate_</i>	各ノードのサービスを生成する方法を指示します
<i>groupServiceTemplate</i> [Required] <i>_ServiceTemplate_</i>	PGDGroup のサービスを生成する方法を指示します
<i>proxyServiceTemplate</i> [Required] <i>_ServiceTemplate_</i>	書き込みリーダーノードルーティングの PGD プロキシを指すサービスを生成する方法を指示します
<i>proxyReadServiceTemplate</i> [Required] <i>_ServiceTemplate_</i>	読み取りノードルーティングの PGD プロキシを指すサービスを生成する方法を指示します

4.18.32 ConnectivityStatus

使用箇所

- *PGDGroupStatus*

ConnectivityStatus には、接続に関するオペレーターの関連ステータスが含まれています

Field	Description
<i>replicationTLSCertificate</i> [Required] <i>_ReplicationCertificateStatus_</i>	ReplicationTLSCertificate は、レプリケーション TLS 証明書の名前です。それがある場合
<i>nodeTLSCertificates</i> [Required] <i>[_[]NodeCertificateStatus_](NodeCertificateStatus_)</i>	NodeTLSCertificates は、PGD ノード用に作成された証明書の名前です。

次のページに続く

表 31 – 前のページからの続き

Field	Description
<i>unusedCertificates</i> [Required] <i>_[]string_</i>	未使用の証明書は、PGD ノードで使用されなくなった証明書の名前です。
<i>nodesWithoutCertificates</i> [Required] <i>_[]string_</i>	<i>NodesWithoutCertificates</i> は、サーバー証明書を持っていないノードの名前です。
<i>nodesNeedingServiceReconciliation</i> [Required] <i>_[]string_</i>	<i>NodesNeedingServiceReconciliation</i> は、サーバー証明書を持っていないノードの名前です。
<i>configurationHash</i> [Required] <i>_string_</i>	<i>ConfigurationHash</i> は、接続構成のハッシュコードであり、構成に変更があったかどうかを確認するために使用されます

4.18.33 DNSConfiguration

使用箇所

- [RootDNSConfiguration](#)

DNSConfiguration は、リソースの FQDN を生成する方法を説明します

Field	Description
<i>domain</i> [Required] <i>_string_</i>	PGDGroup 内のすべてのサービスのドメイン名が含まれます。ここで指定された値がレンダリングされた <i>nodeServiceTemplate</i> および <i>groupServiceTemplate</i> と一致することを確認するのはユーザーの責任です。
<i>hostSuffix</i> [Required] <i>_string_</i>	PGDGroup 内のすべてのサービス名に追加するオプションのサフィックスが含まれます。この設定の意味は、ユーザーがルーティングの目的で場所で作成されたすべてのサービスを簡単にマークできるようにすることです。つまり、汎用ルールを CoreDNS に追加して、一部のサービスサフィックスをローカルとして書き換えます

4.18.34 DiscoveryJobConfig

使用箇所

- [PgdcConfiguration](#)

DiscoveryJobConfig には、ディスカバリージョブを構成する一連のフィールドが含まれています

Field	Description
<i>delay</i> [Required] <i>_int_</i>	再試行間のスリープまでの遅延時間秒単位で測定
<i>retries</i> [Required] <i>_int_</i>	Retries 操作を再試行する回数
<i>timeout</i> [Required] <i>_int_</i>	タイムアウト 操作が成功するまでに与えられた時間秒単位で測定

4.18.35 ImageCatalogRef

使用箇所

- *PGDGroupSpec*

ImageCatalogRef は、参照される ImageCatalog と参照されるメジャー Postgres バージョンを定義します

Field	Description
<i>TypedLocalObjectReference</i> <i>_TypedLocalObjectReference_</i>	<i>TypedLocalObjectReference`</i> のメンバーはこのタイプに埋め込まれます。説明はありません。
<i>major</i> [Required] <i>_int_</i>	ImageCatalog から使用する PostgreSQL のメジャーバージョン

4.18.36 ImageCatalogSpec

使用箇所

- `:ref:ClusterImageCatalog`` リソースを使用したクラスター全体のカタログの例 `<ClusterImageCatalog` リソースを使用したクラスター全体のカタログの例

ImageCatalogSpec は、目的のイメージカタログを定義します

Field	Description
<i>pgdImages</i> [Required] <i>[_[]CatalogImage_](CatalogImage)</i>	カタログで使用可能なカタログ画像のリスト
<i>proxyImage</i> [Required] <i>_string_</i>	カタログで使用可能なプロキシイメージ

4.18.37 ImageStatus

使用箇所

- *PGDGroupStatus*

ImageStatus は、現在使用されているイメージを定義します

Field	Description
<i>pgd</i> [Required] <i>_string_</i>	PGD は、現在使用されている pgd イメージです
<i>proxy</i> [Required] <i>_string_</i>	PGDProxy は、現在使用されている pgd プロキシイメージです

4.18.38 InheritedMetadata

使用箇所

- *CnpBaseConfiguration*

InheritedMetadata 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ます

Field	Description
<i>labels</i> [Required] _map[string]string_	説明はありません。
<i>annotations</i> [Required] _map[string]string_	説明はありません。

4.18.39 JoinMethod

string のエイリアス

使用箇所

- *CnpConfiguration*

JoinMethod は、サポートされているノードに参加する方法の 1 つを表します

4.18.40 Metadata

使用箇所

- *CertManagerTemplate*

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。

Field	Description
<i>labels</i> _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ピング。レプリケーションコントローラとサービスのセクターと一致する場合があります。詳細 http://kubernetes.io/docs/user-guide/labels
<i>annotations</i> _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ピングであり、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

4.18.41 NodeCertificateStatus

使用箇所

- *ConnectivityStatus*

NodeCertificateStatus は、CNP ノードのサーバー証明書のステータスをカプセル化します

Field	Description
<i>ReplicationCertificateStatus</i> <i>_ReplicationCertificateStatus_</i>	<i>ReplicationCertificateStatus`</i> のメンバーはこのタイプに埋め込まれます。説明はありません。
<i>nodeName</i> [Required] <i>_string_</i>	nodeName は、この証明書を使用する CNP クラスターの名前です。

4.18.42 NodeKindName

string のエイリアス

使用箇所

- *NodeSummary*

NodeKindName は、bdr.node_summary からの node_kind_name の潜在的な値を含むタイプです

4.18.43 NodeSummary

使用箇所

- *PGDGGroupStatus*

Node 概要は bdr.node_summary からの関連情報を表示します

Field	Description
<i>node_name</i> [Required] <i>_string_</i>	ノードの名前
<i>node_group_name</i> [Required] <i>_string_</i>	NodeGroupName は、参加したグループの名前です。
<i>peer_state_name</i> [Required] <i>_string_</i>	人間が判読可能な形式でのノードの一貫した状態
<i>peer_target_state_name</i> [Required] <i>_string_</i>	ノードが到達しようとしている状態参加または昇格中に

次のページに続く

表 40 – 前のページからの続き

Field	Description
<i>node_kind_name</i> [Required] <i>_NodeKind-Name_</i>	ノードの種類 witness または data

4.18.44 NodesExtensionsStatus

(`[github.com/EnterpriseDB/pg4k-pgd/api/v1beta1.NodeExtensionStatus]` のエイリアス)

NodesExtensionsStatus には、NodeExtensionStatus エントリのリストが含まれています

4.18.45 OTELConfiguration

使用箇所

- *CnpBaseConfiguration*

OTELConfiguration は、外部 openTelemetry の構成です

Field	Description
<i>metricsURL</i> [Required] <i>_string_</i>	メトリックデータを受け入れる OpenTelemetry HTTP エンドポイント URL
<i>traceURL</i> [Required] <i>_string_</i>	トレースデータを受け入れる OpenTelemetry HTTP エンドポイント URL
<i>traceEnable</i> [Required] <i>_bool_</i>	トレースデータを OpenTelemetry の <i>traceUrl</i> にプッシュするかどうか
<i>tls</i> [Required] <i>_OTELTLSConfiguration_</i>	TLSConfiguration は、MetricsURL および TraceURL が HTTPS を使用している場合に TLS 証明書構成を提供します

4.18.46 OTELTLSConfiguration

使用箇所

- *OTELConfiguration*

OTELTLSConfiguration には、openTelemetry への TLS 接続の証明書構成が含まれています

Field	Description
<i>caBundleSecretRef</i> [Required] <i>_SecretKeySelector_</i>	CABundleSecretRef は、openTelemetry サーバー証明書を検証するための CA バンドルを含むシークレットフィールドへの参照です
<i>clientCertSecret</i> [Required] <i>_LocalObjectReference_</i>	ClientCertSecret は、openTelemetry への接続に使用されるクライアント証明書を含むシークレットの名前です。これには、PEM 形式でエンコードされた標準の「tls.crt」ファイルと「tls.key」ファイルの両方が含まれる必要があります。

4.18.47 OperatorPhase

string のエイリアス

使用箇所

- *PGDGroupStatus*

OperatorPhase PGDGroup コントローラのフェーズを表します

4.18.48 OperatorPhaseCleanup

string のエイリアス

使用箇所

- *PGDGroupCleanupStatus*

OperatorPhaseCleanup PGDGroupCleanup コントローラのフェーズを表します

4.18.49 PGDGroupCleanupSpec

使用箇所

- *PGDGroupCleanup*

PGDGroupCleanupSpec は、PGDGroupCleanup の目的の状態を定義します

Field	Description
<i>executor</i> [Required] <i>_string_</i>	説明はありません。
<i>target</i> [Required] <i>_string_</i>	説明はありません。
<i>force</i> [Required] <i>_bool_</i>	Force は、ターゲット PGDGroup ノードが分離されていない場合でも、PGDGroup の削除を強制します

4.18.50 PGDGroupCleanupStatus

使用箇所

- *PGDGroupCleanup*

PGDGroupCleanupStatus は、PGDGroupCleanup の観察された状態を定義します

Field	Description
<i>phase</i> [Required] <i>_OperatorPhaseCleanup_</i>	説明はありません。

4.18.51 PGDGroupSpec

使用箇所

- *PGDGroup parting*

PGDGroupSpec は、PGDGroup の目的の状態を定義します

Field	Description
<i>imageName</i> [Required] <i>_string_</i>	コンテナイメージの名前、タグ <code><image>:<tag></code> と決定的で反復可能な展開用のダイジェスト <code><image>:<tag>@sha256:<digestValue></code> の両方をサポート
<i>imageCatalogRef</i> <i>_ImageCatalogRef_</i>	参照される ImageCatalog と参照される Postgres のメジャーバージョンを定義します
<i>imagePullPolicy</i> <i>_PullPolicy_</i>	イメージプルポリシー。 <i>Always</i> 、 <i>Never</i> または <code>{" "}</code> <i>IfNotPresent</i> のいずれか。定義されていない場合、デフォルトの <code>{" "}</code> <i>IfNotPresent</i> になります。更新できません。詳細 https://kubernetes.io/docs/concepts/containerimages#updating-images
<i>imagePullSecrets</i> [Required] <i>[_LocalObjectReference_](https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#localobjectreference-v1-c</i>	演算子またはオペランドイメージをプルするために使用されるプルシークレットのリスト
<i>inheritedMetadata</i> [Required] <i>_InheritedMetadata_</i>	pgdGroup 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
<i>instances</i> [Required] <i>_int32_</i>	クラスターに必要なインスタンスの数
<i>proxyInstances</i> [Required] <i>_int32_</i>	クラスターに必要なプロキシインスタンスの数

次のページに続く

表 45 – 前のページからの続き

Field	Description
<i>witnessInstances</i> [Required] <i>_int32_</i>	クラスターに必要な監視インスタンスの数
<i>backup</i> [Required] <i>_Backup_</i>	CNP インスタンスのバックアップに使用される構成。
<i>restore</i> [Required] <i>_Restore_</i>	オブジェクトストアサービスからこの PGD グループを復元する構成
<i>cnp</i> [Required] <i>_CnpConfiguration_</i>	PGD グループを構成する CNP クラスターに注入されるインスタンス構成
<i>witness</i> [Required] <i>_CnpBaseConfiguration_</i>	WitnessInstances CNP クラスターに注入される WitnessInstances 構成定義されていない場合、デフォルトのインスタンス構成
<i>pgd</i> [Required] <i>_PgdConfiguration_</i>	Pgd には、このクラスターをブートストラップするための指示が含まれています
<i>pgdProxy</i> [Required] <i>_PGDProxyConfiguration_</i>	PGDProxy には、PGD プロキシの構成手順が含まれています
<i>connectivity</i> [Required] <i>_ConnectivityConfiguration_</i>	使用されるサービスや証明書など、PGDGroup の接続を構成します。
<i>failingFinalizerTimeLimitSeconds</i> [Required] <i>_int32_</i>	ファイナライザーが失敗した場合にオペレーターが待機する秒数。オペレーターが PGD グループのノードに到達できない場合、ファイナライザーは失敗したと考えられます。

4.18.52 PGDGroupStatus

使用箇所

- *PGDGroup parting*

PGDGroupStatus は、PGDGroup の観察された状態を定義します

Field	Description
<i>latestGeneratedNode</i> [Required] <i>_int32_</i>	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
<i>phase</i> [Required] <i>_OperatorPhase_</i>	このクラスターの初期化フェーズ
<i>phaseDetails</i> [Required] <i>_string_</i>	現在のフェーズの詳細
<i>phaseTroubleshootHints</i> [Required] <i>_string_</i>	PhaseTroubleshootHints 特定のフェーズの一般的なトラブルシューティングの兆候
<i>phaseType</i> [Required] <i>_PhaseType_</i>	PhaseType は、フェーズカテゴリを説明します。

次のページに続く

表 46 – 前のページからの続き

Field	Description
<i>conditions</i> [Required] <i>[_]Condition_</i> (https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#condition-v1-meta)	PGDGroup オブジェクトの条件
<i>nodes</i> [Required] <i>[_]NodeSummary_</i> (NodeSummary)	グループ内のノードのサマリーのリスト
<i>backup</i> [Required] <i>_BackupStatus_</i>	この PGD グループのバックアップを取得しているノード
<i>restore</i> [Required] <i>_RestoreStatus_</i>	復元プロセスのステータス
<i>PGD</i> [Required] <i>_PGDStatus_</i>	PGD の最後の既知のステータス
<i>CNP</i> [Required] <i>_CNPStatus_</i>	CNP の最後に知られているステータス
<i>PGDProxy</i> [Required] <i>_PGDProxyStatus_</i>	PGDP プロキシの最後の既知のステータス
<i>connectivity</i> [Required] <i>_ConnectivityStatus_</i>	接続の最後の既知のステータス
<i>pause</i> [Required] <i>_PauseStatus_</i>	一時停止の最後の既知のステータス
<i>image</i> [Required] <i>_ImageStatus_</i>	使用されたイメージの最後の既知のステータス

4.18.53 PGDNodeGroupEntry

使用箇所

- *PGDStatus*

PGDNodeGroupEntry は、PGD 構成で使用可能なノードグループに関する情報を示します

Field	Description
<i>name</i> [Required] <i>_string_</i>	Name はノードグループの名前です
<i>enableProxyRouting</i> [Required] <i>_bool_</i>	EnableProxyRouting は true です。ノードグループで PGD プロキシの実行が許可されています
<i>enableRaft</i> [Required] <i>_bool_</i>	ノードグループにサブグループ raft インスタンスがある場合、EnableRaft は true です

次のページに続く

表 47 – 前のページからの続き

Field	Description
<i>routeWriterMaxLag</i> [Required] _int64_	RouteWriterMaxLag 書き込みリーダーとして選択される新しい書き込み候補のバイト単位の最大遅延。これを通過する候補がない場合、自動的に選択されるライタはありません
<i>routeReaderMaxLag</i> [Required] _int64_	RouteReaderMaxLag 実行可能な読み取り専用ノードと考えられるノードのバイト単位の最大遅延
<i>routeWriterWaitFlush</i> [Required] _bool_	RouteWriterWaitFlush <code>bdr.routing_leadership_transfer()</code> を使用する場合、新しいリーダーに切り替える前にレプリケーションキューのフラッシュを待機するかどうか <code>bdr.routing_leadership_transfer()</code>

4.18.54 PGDNodeGroupSettings

使用箇所

- *PgdConfiguration*

PGDNodeGroupSettings には、PGD グループの設定が含まれています

Field	Description
<i>routeWriterMaxLag</i> [Required] _int64_	RouteWriterMaxLag 書き込みリーダーとして選択される新しい書き込み候補のバイト単位の最大遅延。候補がこれを通過しない場合、ライタは自動的に選択されません デフォルト-1
<i>routeReaderMaxLag</i> [Required] _int64_	RouteReaderMaxLag 実行可能な読み取り専用ノードと考えられるノードのバイト単位の最大遅延デフォルト-1
<i>routeWriterWaitFlush</i> [Required] _bool_	RouteWriterWaitFlush {" " } <code>bdr.routing_leadership_transfer()</code> を使用する場合、新しいリーダーに切り替える前にレプリケーションキューのフラッシュを待機するかどうか デフォルト false

4.18.55 PGDProxyConfiguration

使用箇所

- *PGDGroupSpec*

PGDProxyConfiguration は、PGD プロキシの構成を定義します

Field	Description
<i>imageName</i> [Required] _string_	PGDProxy コンテナイメージの名前
<i>logLevel</i> [Required] _string_	PGD プロキシログレベル、次の値のいずれかエラー、警告、情報デフォルト、デバッグ、トレース

次のページに続く

表 49 – 前のページからの続き

Field	Description
<i>logEncoder</i> [Required] <i>_string_</i>	ログ出力のフォーマット
<i>proxyAffinity</i> [Required] <i>_Affinity_</i>	ポッドのプロキシアフィニティ/アンチアフィニティルール
<i>proxyNodeSelector</i> [Required] <i>_map[string]string_</i>	ポッドの ProxyNodeSelector ルール
<i>proxyTolerations</i> [Required] <i>[_][]Toleration_](https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.28/#toleration-v1-core)</i>	ポッドのプロキシトレランスルール
<i>proxyResources</i> <i>_ResourceRequirements_</i>	プロキシに割り当てられるリソースを定義します。定義されていない場合、デフォルトの要求を使用し、値を制限します。

4.18.56 PGDProxyEntry

使用箇所

- *PGDStatus*

PGDProxyEntry は、PGD 構成で使用可能なプロキシに関する情報を表示します

Field	Description
<i>name</i> [Required] <i>_string_</i>	Name はプロキシの名前です
<i>fallbackGroupNames</i> [Required] <i>[_][]string_</i>	FallbackGroupNames は、このプロキシ用に構成されたフォールバックグループの名前です。
<i>parentGroupName</i> [Required] <i>_string_</i>	ParentGroupName は、このプロキシの親 PGD グループです。
<i>maxClientConn</i> [Required] <i>_int_</i>	MaxClientConn プロキシが受け入れる接続の最大数
<i>maxServerConn</i> [Required] <i>_int_</i>	MaxServerConn プロキシが Postgres ノードへの接続の最大数
<i>serverConnTimeout</i> [Required] <i>_int64_</i>	ServerConnTimeout サーバー接続の秒単位の接続タイムアウト
<i>serverConnKeepalive</i> [Required] <i>_int64_</i>	ServerConnKeepalive サーバー接続の秒単位のキープアライブ間隔
<i>fallbackGroupTimeout</i> [Required] <i>_int64_</i>	FallbackGroupTimeout ルーティングが fallback_group のいずれかにフォールバックするまでの間隔

次のページに続く

表 50 – 前のページからの続き

Field	Description
<i>consensusGracePeriod</i> [Required] _int64_	コンセンサスグレースピリオド Raft リーダーを失ってもプロキシがルーティングを継続する秒単位の期間。
<i>readListenPort</i> [Required] _int_	[Required] ReadListenPort は、プロキシがクエリを監視し、読み取りノードにルーティングするポートです

4.18.57 PGDProxySettings

使用箇所

- *PgdConfiguration*

PGDProxySettings には、プロキシの設定が含まれています

Field	Description
<i>fallbackGroups</i> [Required] _[]string_	[Required] FallbackGroups は、この PGD グループのすべてのデータノードが利用できないときにプロキシが接続を転送するグループのリストです
<i>maxClientConn</i> [Required] _int_	[Required] MaxClientConn プロキシが受け入れる接続の最大数。デフォルト 32767
<i>maxServerConn</i> [Required] _int_	[Required] MaxServerConn プロキシが Postgres ノードへの接続の最大数。デフォルト 32767
<i>serverConnTimeout</i> [Required] _int64_	[Required] ServerConnTimeout サーバー接続の秒単位の接続タイムアウト。デフォルト 2
<i>serverConnKeepalive</i> [Required] _int64_	[Required] ServerConnKeepalive サーバー接続の秒単位のキープアライブ間隔。デフォルト 10
<i>fallbackGroupTimeout</i> [Required] _int64_	[Required] FallbackGroupTimeout ルーティングが fallback_group のいずれかにフォールバックするまでの間隔。デフォルト 60
<i>consensusGracePeriod</i> [Required] _int64_	[Required] コンセンサスグレースピリオド Raft リーダーを失ってもプロキシがルーティングを継続する秒単位の期間。0 に設定すると、プロキシはルーティングをすぐに停止します。デフォルト 6
<i>enableReadNodeRouting</i> [Required] _bool_	[Required] EnableReadNodeRouting は、プロキシが read_listen_port を介して読み取りノードにクエリーをルーティングするかどうかを制御するスイッチです。デフォルトでは false です

4.18.58 PGDProxyStatus

使用箇所

- *PGDGroupStatus*

PGDProxyStatus PGDP プロキシに関するオペレーターの関連ステータス

Field	Description
<i>proxyInstances</i> [Required] <i>_int32_</i>	説明はありません。
<i>writeLead</i> [Required] <i>_string_</i>	WriteLead はオペレーターの予約フィールドであり、外部使用を目的としたものではありません。将来のバージョンでは削除される予定
<i>proxyHash</i> [Required] <i>_string_</i>	ProxyHash には、プロキシを調整する必要があるかどうかを検出するために使用するハッシュが含まれています

4.18.59 PGDRaftStatus

string のエイリアス

使用箇所

- *PGDStatus*

PGDRaftStatus は、PGDRaft の既知のステータスを示します

4.18.60 PGDStatus

使用箇所

- *PGDGroupStatus*

PGDStatus PGD に関するオペレーターの関連ステータス

Field	Description
<i>raftConsensus-LastChangedStatus</i> [Required] <i>_PGDRaft-Status_</i>	RaftConsensusLastChangedStatus は、bdr.monitor_group_raft から最新に報告されたステータスを示します
<i>raftConsensus-LastChangedMessage</i> [Required] <i>_string_</i>	RaftConsensusLastChangedMessage は、bdr.monitor_group_raft からの最新の報告されたメッセージを示します
<i>raftConsensus-LastChangedTimestamp</i> [Required] <i>_string_</i>	RaftConsensusLastChangedTimestamp は、ステータスとメッセージが最初に報告されたときを示します
<i>registeredProxies</i> [Required] <i>[_[]PGDProxyEntry_](PGDProxyEntry)</i>	RegisteredProxies は、登録されたプロキシのステータスです
<i>nodeGroup</i> [Required] <i>_PGDNodeGroupEntry_</i>	NodeGroup は、PGDGroup に関連付けられたノードグループのステータスです

4.18.61 ParentGroupConfiguration

使用箇所

- *PgdConfiguration*

ParentGroupConfiguration には、PGD のトポロジ構成が含まれます

Field	Description
<i>name</i> [Required] <i>_string_</i>	親グループの名前
<i>create</i> [Required] <i>_bool_</i>	オペレーターが親グループが存在しない場合に作成する必要がある場合に、Create は true です

4.18.62 PauseStatus

使用箇所

- *PGDGroupStatus*

PauseStatus には、グループ冬眠の情報が含まれています

Field	Description
<i>active</i> [Required] <i>_bool_</i>	Active は、PGD グループが次のいずれかであることを示します。 <i>_一時停止中</i> <i>_既に一時停止中</i> <i>*再開中</i>
<i>instances</i> [Required] <i>_int32_</i>	Instances は、一時停止された PGD インスタンスの数です
<i>lastStartedTime</i> [Required] <i>_Time_</i>	LastStartedTime は、PGDGroup が最後に一時停止を開始した時間です
<i>lastCompletedTime</i> [Required] <i>_Time_</i>	LastCompletedTime は、PGDGroup が最後に一時停止を完了した時間です
<i>lastResumeStartedTime</i> [Required] <i>_Time_</i>	LastResumeStartedTime は、PGDGroup が最後に再開を開始した時間です
<i>lastResumeCompletedTime</i> [Required] <i>_Time_</i>	LastCompletedTime は、PGDGroup が最後に再開を完了した時間です

4.18.63 PgDConfiguration

使用箇所

- *PgDGroupSpec*

PgDConfiguration は、PGD グループ構造の構成です

Field	Description
<i>parentGroup</i> [Required] <i>_ParentGroupConfigura-</i> <i>tion_</i>	ParentGroup は PGD グループのトポロジを構成します
<i>groupJoinMethod</i> <i>_Join-</i> <i>Method_</i>	GroupJoinMethod は、クロスリージョン結合の場合の方法を定義します
<i>discovery</i> [Required] <i>[_](Connection-</i> <i>String_)(ConnectionString</i>	親 PGD グループに属するノードに接続するために使用するパラメーター。指定した場合でも、次のパラメーターはデフォルト値でオーバーライドされます。 <i>application_name</i> 、 <i>sslmode</i> 、{" "} <i>dbname</i> および <i>user</i> 。次のパラメーターは、デフォルトでオーバーライドされないため、提供または使用しないでください。 <i>sslkey</i> 、 <i>sslcert</i> 、{" "} <i>sslrootcert</i>
<i>discoveryJob</i> [Required] <i>_DiscoveryJobConfig_</i>	DiscoveryJob PGD Discovery ジョブの構成
<i>databaseName</i> [Re- quired] <i>_string_</i>	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 {" "} <i>app</i>
<i>ownerName</i> [Required] <i>_string_</i>	アプリケーションが使用するインスタンス内のデータベースの所有者の名前。 `database` キーの値のデフォルト。
<i>ownerCredentialsSecret</i> [Required] <i>_LocalOb-</i> <i>jectReference_</i>	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
<i>proxySettings</i> [Re- quired] <i>_PGDProxySet-</i> <i>tings_</i>	プロキシの構成
<i>nodeGroupSettings</i> [Required] <i>_PGDNode-</i> <i>GroupSettings_</i>	PGD グループの構成
<i>globalRouting</i> [Re- quired] <i>_bool_</i>	GlobalRouting は、グローバルルーティングが有効になっている場合に true であり、この場合、プロキシは親グループに作成されます
<i>mutations</i> [Required] <i>_SQLMutations_</i>	ノードグループに適用する SQL ミューテーションのリスト

4.18.64 PhaseType

`string` のエイリアス

使用箇所

- *PGDGroupStatus*

PhaseType は、OperatorPhase のタイプを説明します

4.18.65 PreProvisionedCertificate

使用箇所

- *ClientPreProvisionedCertificates*

PreProvisionedCertificate には、事前に生成された証明書を提供するために必要なデータが含まれています

Field	Description
<i>secretRef</i> [Required] <i>_string_</i>	SecretRef tls.crt および tls.key を含むシークレットを指す名前

4.18.66 ReconciliationStage

`string` のエイリアス

使用箇所

ReconciliationStage は、調整プロセスがどのステージにあるかを説明します

4.18.67 RecoverabilityPointsByMethod

(`map[github.com/EnterpriseDB/cloud-native-postgres/api/v1.BackupMethod]k8s.io/apimachinery/pkg/apis/meta/v1.Time`` のエイリアス)

使用箇所

- *CNPStatus*

RecoverabilityPointsByMethod には、特定のバックアップ方法の最初の復旧ポイントが含まれます

4.18.68 ReplicationCertificateStatus

使用箇所

- *ConnectivityStatus*

ReplicationCertificateStatus は証明書ステータスをカプセル化します

Field	Description
<i>name</i> [Required] <i>_string_</i>	Name は証明書の名前です
<i>hash</i> [Required] <i>_string_</i>	ハッシュは、生成された構成のハッシュです
<i>isReady</i> [Required] <i>_bool_</i>	証明書の準備ができている場合、Ready は true です
<i>preProvisioned</i> [Required] <i>_bool_</i>	証明書が事前プロビジョニングされている場合、PreProvisioned は true です

4.18.69 Restore

使用箇所

- *PGDGroupSpec*

Restore は、オブジェクトストアからの PGD グループのリストアを構成します

Field	Description
<i>volumeSnapshots</i> <i>_VolumeSnapshotsConfiguration_</i>	volumeSnapshot リストアの構成
<i>barmanObjectStore</i> [Required] <i>_BarmanObjectStoreConfiguration_</i>	barman-cloud ツールスイートの構成
<i>recoveryTarget</i> [Required] <i>Bootstrap</i>	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 で予想されるように `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
<i>serverNames</i> [Required] <i>_[]string_</i>	復旧元として使用されるサーバー名のリスト。リカバリターゲットを評価するときに、これらのサーバーのいずれかがシードサーバーとして選択されます。このオプションは、barmanObjectStore から復元する場合にのみ使用されます。

4.18.70 RestoreStatus

使用箇所

- *PGDGroupStatus*

RestoreStatus には、復元プロセスの現在のステータスが含まれています

Field	Description
<i>serverName</i> [Required] _string_	復元するサーバーの名前
<i>VolumeSnapshots</i> [Required] <i>[_]VolumeSnapshotRestoreStatus_</i> (VolumeSnapshotRes	選択したボリューム復元するスナップショット

4.18.71 RootDNSConfiguration

使用箇所

- *ConnectivityConfiguration*

RootDNSConfiguration は、リソースの FQDN を生成する方法を説明します

Field	Description
<i>DNSConfiguration</i> _DNSConfiguration_	`DNSConfiguration` のメンバーはこのタイプに埋め込まれます。説明はありません。
<i>additional</i> [Required] _[]DNSConfiguration_](DNSConfiguration)	AdditionalDNSConfigurs は、リソースに使用可能な FQDN をさらに追加します

4.18.72 SQLMutation

SQLMutation は、アトミックに適用する一連の SQL ステートメントです

Field	Description
<i>isApplied</i> [Required] _[]string_	ブール値を返す SQL クエリーのリスト。それらのいずれかが false を返す場合、ミューテーションが適用されます
<i>exec</i> [Required] _[]string_	この変更を適用するために実行される SQL クエリーのリスト

次のページに続く

表 62 – 前のページからの続き

Field	Description
<i>type</i> _SQLMutation-Type_	Type は、SQL ミューテーションが発生する時期を決定します。'always' 各調整サイクルで Mutation を調整します 'beforeSubgroupRaft' サブグループラフトが有効になる前にのみ実行されます 指定しない場合、タイプはデフォルトの'always' になります。

4.18.73 SQLMutationType

`string` のエイリアス

使用箇所

- [SQLMutations](#)

SQLMutationType は、サポートされている SQL ミューテーションのタイプです

4.18.74 SQLMutations

(`[github.com/EnterpriseDB/pg4k-pgd/api/v1beta1.SQLMutation`` のエイリアス)

使用箇所

- [PgdConfiguration](#)

SQLMutation SQLMutation のリスト

4.18.75 ScheduledBackupSpec

使用箇所

- [バックアップとリストア](#)

ScheduledBackupSpec は、ScheduledBackup の目的の状態を定義します

Field	Description
<i>suspend</i> _bool_	[Required] このバックアップが一時停止されているかどうか
<i>immediate</i> _bool_	[Required] 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
<i>schedule</i> _string_	[Required] スケジュールは、追加の 2 番目の指定子が含まれるため、Kubernetes 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

次のページに続く

表 63 – 前のページからの続き

Field	Description
<i>backupOwnerReference</i> [Required] <i>_string_</i>	作成したバックアップリソース内にどの <i>ownerReference</i> を配置するかを示します。 <i>_none</i> 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 <i>_self</i> 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 * <i>cluster</i> クラスターをバックアップの所有者として設定します
<i>target</i> [Required] <i>Bootstrap</i>	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 <code>cluster.spec.backup.target</code> になります。使用可能なオプションは、空の文字列、 <i>primary</i> 、および <code>" "</code> です。 <i>prefer-standby.primary</i> は常にプライマリインスタンスでバックアップを実行します。 <i>prefer-standby</i> は、可能な場合は最も更新されたスタンバイでバックアップを実行します。
<i>method</i> <i>Bootstrap</i>	使用するバックアップ方法、可能なオプションは <code>" "</code> <i>barmanObjectStore</i> および <i>volumeSnapshot</i> です。デフォルト <code>" "</code> <i>barmanObjectStore</i> 。
<i>online</i> <i>_bool_</i>	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 <i>true</i> 、default またはオフライン/コールド <i>false</i> かどうか クラスターフィールド <code>'spec.backup.volumeSnapshot.online'</code> で指定されたデフォルト設定をオーバーライドします
<i>onlineConfiguration</i> <i>_OnlineConfiguration_</i>	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 <code>'backup.volumeSnapshot.onlineConfiguration'</code> スタンザ

4.18.76 ScheduledBackupStatus

使用箇所

- *BackupStatus*

ScheduledBackupStatus には、スケジュールされたバックアップの状態が含まれます

Field	Description
<i>method</i> [Required] <i>Bootstrap</i>	Method は、バックアップを取得するために使用されるバックアップ方法です
<i>name</i> [Required] <i>_string_</i>	Name はスケジュールされたバックアップの名前です
<i>hash</i> [Required] <i>_string_</i>	Hash は、スケジュールされたバックアップ構成のハッシュです

4.18.77 ServerCertConfiguration

使用箇所

- *OTELTLSConfiguration*

ServerCertConfiguration には、ノードの証明書を生成するための情報が含まれています

Field	Description
<i>caCertSecret</i> [Required] <i>_string_</i>	CACertSecret は、CloudNativePG ServerCASecret 構成に注入される CA のシークレットです。
<i>certManager</i> [Required] <i>_CertManagerTemplate_</i>	証明書の生成に使用される cert-manager テンプレート

4.18.78 ServiceTemplate

使用箇所

- *ConnectivityConfiguration*

ServiceTemplate は、ユーザーが Service 生成のテンプレートを設定できる構造です。

Field	Description
<i>metadata</i> <i>_Metadata_</i>	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
<i>spec</i> <i>_ServiceSpec_</i>	サービスの目的な動作の仕様。詳細情報 https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status
<i>updateStrategy</i> <i>_ServiceUpdateStrategy_</i>	UpdateStrategy は、このテンプレートによって生成されたサービスを更新する方法を示します。

4.18.79 ServiceUpdateStrategy

string のエイリアス

使用箇所

- *ServiceTemplate*

ServiceUpdateStrategy は、ロードバランサーを更新するためのタイプを定義します。使用できる値は「patch」および「replace」です。

4.18.80 TLSConfiguration

使用箇所

- *ConnectivityConfiguration*

TLSConfiguration は、PGD がノードに接続するために使用する TLS インフラストラクチャの構成です

Field	Description
<i>mode</i> [Required] <i>_TLSMode_</i>	説明はありません。
<i>serverCert</i> [Required] <i>_ServerCertConfiguration_</i>	サーバー証明書の構成
<i>clientCert</i> [Required] <i>_ClientCertConfiguration_</i>	クライアント証明書の構成

4.18.81 TLSMode

string のエイリアス

使用箇所

- *OTELTLSConfiguration*

TLSMode は、ノード間の通信にどのモードを使用するかを説明します

4.18.82 VolumeSnapshotRestoreStatus

使用箇所

- *RestoreStatus*

VolumeSnapshotRestoreStatus 復元する volumeSnapshot

Field	Description
<i>snapshotName</i> [Required] <i>_string_</i>	SnapshotName は、復元するスナップショットの名前です。
<i>pvcRole</i> [Required] <i>_string_</i>	PVCRole は、復元する pvcRole スナップショットです
<i>tableName</i> [Required] <i>_string_</i>	TableName は、pvcRole が `PG_TABLESPACE` の場合、スナップショットが属するテーブルスペース名です

4.18.83 VolumeSnapshotsConfiguration

使用箇所

- バックアップとリストア

VolumeSnapshotsConfiguration には、volumeSnapshots リストアの構成が含まれます

Field	Description
<code>selector [Required] _LabelSelector_</code>	復元するボリュームスナップショットの選択に使用されるラベルセクター

4.19 Supported versions

このページには、*Kubernetes* の EDB Postgres Distributed の現在サポートされているリリースのステータスをリストしています。

4.19.1 命名スキーム

EDB Postgres Distributed for Kubernetes PGD4K リリースは、セマンティックバージョニング 2.0.0 に従い、次のような構造になっています。

<major>.<minor>.<patch>

- <major> は、サポートされている PGD バージョンにマップします。バージョン 1.x は、PGD バージョン 5.x に対応します。
- <minor> は、サポートされている PG4K 長期サポートリリースの依存関係が変更されるごとにインクリメントされます。
- <patch> は、現在の<minor> リリースのパッチをインデックスし、<minor> リリースと比較して小さな変更を表します。

バージョンの Git タグには、先頭に **v** が付きます。

4.19.2 一般的なサポートステータス

Kubernetes バージョン 1.x の EDB Postgres Distributed は、Postgres Distributed バージョン 5.x を管理するために使用されます。一般的なサポートステータスは次のとおりです。

Version	Currently Supported	Release Date	Supported Kubernetes Versions	Supported PG4K Versions	Support
1.x	Yes	Apr 24, 2024	v1.28 - v1.32	v1.22, v1.25	v13 - v

Postgres オペラントバージョンは [EDB Postgres Distributed \(PGD\)](#) でサポートされているものに制限されています

重要: このページは情報提供のみを目的としています。EDB ウェブサイトの [Platform Compatibility](#) ページには、サポートされているソフトウェアと Kubernetes ディストリビューションの公式リストが含まれています。

4.19.3 詳細なサポートステータス

EDB Postgres Distributed for Kubernetes PGD4K オペレーターは、EDB Postgres for Kubernetes PG4K オペレーターを使用して、各 PGD ノードを PG4K クラスターとして管理します。各マイナーリリースは、対応する PG4K LTS リリースに合わせてライフ終了を迎えます。各 PG4K LTS リリースのサポート範囲の詳細については、[platform compatibility](#) を参照してください。

Version	Currently Supported	Release Date	Supported Kubernetes Versions	Supported PG4K Versions	Supported PG4K LTS Versions
1.1.x	Yes	Dec 25, 2024	1.29, 1.30, 1.31, 1.32	v1.22.8, v1.25.x	v13 - v15
1.0.x	Yes	Apr 24, 2024	1.28, 1.29, 1.30, 1.31	v1.22.x	v13 - v15

4.19.4 Openshift Platform でサポートされているバージョン

Version	Supported Open-Shift Versions	Supported PG4K Versions	Supported Cert-manager versions
1.1.1	v4.12 - v4.18	v1.22.8, v1.22.9, v1.25.x	>v1.10.0
1.1.0	v4.12 - v4.17	v1.22.8, v1.25.x	>v1.10.0
1.0.1	v4.12 - v4.16	v1.22.x	>v1.10.0
1.0.0	v4.12 - v4.15	>v1.18.0	>v1.10.0

注釈: EDB Postgres Distributed for Kubernetes PGD4K バージョン 1.0.0 は、サポートされている PG4K バージョンを LTS リリースに制限しませんでした。PGD4K を 1.0.0 から 1.0.1 にアップグレードする場合、インストールされた PG4K バージョンが 1.22 を超える場合、最初に PG4K オペレーターをアンインストールし、1.22.x で再インストールして依存関係要件を満たす必要があります。

4.20 Known issues and limitations

これらの既知の問題と制限は、Kubernetes の EDB Postgres Distributed の現在のリリースに含まれています。

4.20.1 Postgres メジャーバージョンのアップグレード

Kubernetes の EDB Postgres Distributed のこのバージョンは、Postgres のメジャーバージョンのアップグレードをサポートしていません**。

4.20.2 データの移行

EDB Postgres Distributed for Kubernetes のこのバージョンは、他の Postgres データベースからのデータの宣言的インポートをサポートしていません。スキーマとデータを移行するには、[EDBLoader](#) または [Migration Toolkit / Replication Server](#) などの従来の Postgres 移行ツールを使用できます。ソースデータベースで `pg_dump ...` を使用し、コマンドの出力を `psql -c` を使用してターゲットデータベースにパイプすることもできます。

4.20.3 PgBouncer との接続

Kubernetes 用 EDB Postgres Distributed は、[PgBouncer](#) を使用してクライアント接続要求をプールすることをサポートしていません。この制限は、PgBouncer のオープンソースと EDB バージョンの両方に適用されます。

4.20.4 バックアップ操作

Kubernetes 環境の EDB Postgres Distributed を構成するには、PGDGroup YAML オブジェクトを各 Kubernetes クラスターに適用する必要があります。このオブジェクトを適用すると、分散アーキテクチャを実装するために必要なすべてのサービスが作成されます。

バックアップ構成をセットアップする目的で `spec.backup` セクションをこの PGDGroup オブジェクトに追加した場合、`spec.backup.cron.schedule` 値も設定しない限り、バックアップは失敗します。

エラー出力例

```
The PGDGroup "region-a" is invalid: spec.backup.cron.schedule: Invalid value: "":  
→ Empty spec string
```

回避策

この問題をワークアラウンドには、要件を満たすスケジュールを使用して `spec.backup.cron.schedule` セクションを追加します。例

```
spec:
  instances: 3
  proxyInstances: 2
  pgd:
    parentGroup:
      create: true
      name: world
    backup:
      configuration:
        barmanObjectStore:
          ...
      cron:
        suspend: false
        immediate: true
        schedule: "0 */5* ** *"
```

4.20.5 EDB Postgres Distributed の既知の問題と制限

展開に含める EDB Postgres Distributed バージョンの既知の問題と制限はすべて、EDB Postgres Distributed for Kubernetes インスタンスに影響を与えます。

たとえば、使用している EDB Postgres Distributed バージョンが 5.x の場合、EDB Postgres Distributed for Kubernetes インスタンスは、これらの [5.x known issues](#) および [5.x limitations](#) の影響を受けます。

4.21 Image catalog

PGD グループには、独自の ImageCatalog および ClusterImageCatalog があります。ImageCatalog および ClusterImageCatalog は、PGDGroup を作成するための PGD および PGD プロキシイメージを定義できる重要なリソースです。

主な違いはスコープにあります。ImageCatalog は名前空間ですが、ClusterImageCatalog はクラスタースコープです。

どちらも、PGD イメージのリストとプロキシイメージで構成される共通の構造を共有しています。各 PGD イメージには、イメージのメジャー Postgres バージョンを示す `major` フィールドが含まれています。

警告: オペレーターはユーザー定義のメジャーバージョンを信頼し、PostgreSQL バージョンの検出を実行しません。カタログで宣言されたメジャーバージョンと PostgreSQL イメージ間のアライメントを保証するのはあなたの責任です。

major フィールドの値は、カタログ内で一意を維持し、イメージ間の重複を防ぐ必要があります。ただし、個別のカタログは、同じ major 値で異なるイメージを公開できます。

4.21.1 名前空間の例 ImageCatalog

```
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: ImageCatalog
metadata:
  name: pgd-postgres-extended
  namespace: default
spec:
  images:
    - major: 15
      image: docker.enterprisedb.com/k8s_enterprise_pgd/edb-postgres-extended:15.12-5.
→7.0-1
    - major: 16
      image: docker.enterprisedb.com/k8s_enterprise_pgd/edb-postgres-extended:16.8-5.
→7.0-1
  proxyImage: docker.enterprisedb.com/k8s_enterprise_pgd/edb-pgd-proxy:5.7.0-1
```

4.21.2 ClusterImageCatalog リソースを使用したクラスター全体のカタログの例

```
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: ClusterImageCatalog
metadata:
  name: pgd-postgres-extended
spec:
  images:
    - major: 15
      image: docker.enterprisedb.com/k8s_enterprise_pgd/edb-postgres-extended:15.12-5.
→7.0-1
    - major: 16
      image: docker.enterprisedb.com/k8s_enterprise_pgd/edb-postgres-extended:16.8-5.
→7.0-1
  proxyImage: docker.enterprisedb.com/k8s_enterprise_pgd/edb-pgd-proxy:5.7.0-1
```

PGDGroup リソースには、ImageCatalog または ClusterImageCatalog を参照して、目的の画像を正確に指定する柔軟性があります。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: PGDGroup
metadata:
  name: group-example
```

(次のページに続く)

(前のページからの続き)

```
spec:
  [...]
  imageCatalogRef:
    apiGroup: pgd.k8s.enterprisedb.io
    kind: ImageCatalog
    name: pgd-postgres-extended
    major: 16
  [...]
```

これらのカタログを利用する PGD グループは、継続的なモニタリングを維持します。カタログ内の画像が変更されると、その特定のエントリを参照する関連するすべての PGD グループの自動更新がトリガーされます。

4.22 Join method

データノードを追加するか、クロスリジョingroupを作成することにより PGD グループをスケールアップする場合、新しいノードが既存のグループに適切に参加することが重要です。参加には2つの方法があります。

- 論理的結合

この方法は、`bdr.join_node_group` ファンクションを使用して、新しいノードを PGD グループに統合します。参加ノードには、PGD グループに存在するスキーマまたはデータが含まれていないことが重要です。結合中にデータの同期が発生するため、新しいデータベースには BDR 拡張機能のみを含めることをお勧めします。

- 物理的結合

この方法では、`bdr_init_physical` コマンドを使用して、参加プロセスを高速化します。`bdr_init_physical` を実行する前に、事前にデータを準備できます。

結合方法の詳細については、[Creating and joining PGD groups](#) を参照してください。

4.22.1 初期グループ

PGD4K オペレーターを使用すると、PGD グループに参加するノード、および他のグループに参加するグループの参加方法を構成できます。

単一の PGD グループを作成するときは、**親グループ** `spec.pgd.parentGroup.create=true` を作成するように構成する必要があります。クロス参加する複数の PGD グループを作成する場合は、これらのグループの少なくとも1つを指定して、親グループを形成する必要があります。このグループは、**初期 PGD グループ**と呼ばれます。

initial PGD グループの最初のノードは、バックアップから復元するリカバリドキュメントの説明に従って、またはスクラッチから作成できます。このノードは、常に論理結合を使用して、作成した親グループに接続します。逆に、**非初期** PGD グループの最初のノードは、他の PGD グループとのクロスジョインに使用されます。

最初のノードグループ参加と後続のノードノード参加の参加方法を個別に構成できます。

注釈: グループが別のグループに参加するシナリオでは、参加するグループの最初のノードが実際に別のグループに参加する前に最初にグループを作成します物理的または論理的。

4.22.2 ノード参加の参加方法を構成する

PGD グループをスケールアップし、新しいノードをローカルグループに追加するときに、グループに参加するノードの参加方法を構成できます。このコンテキストでは、参加グループはノードグループと同じです。

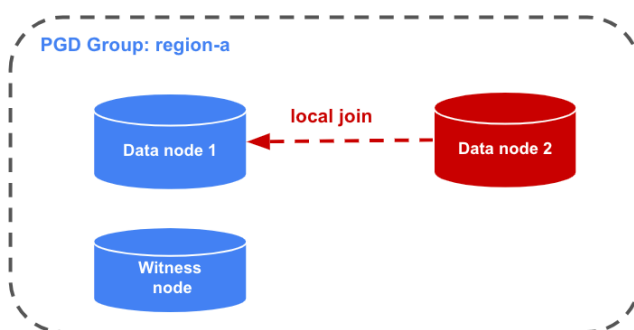


図 3: node join local group

`spec.cnp.joinMethod` パラメーターは、ノードがローカルグループに参加する方法を構成します。デフォルトでは、論理結合が使用されます。新しいノードの物理的参加を有効にするには、`spec.cnp.joinMethod=physical` を設定します。

次の例では、2 番目のノードは `pg_basebackup` コマンドを使用して、`region-a` グループの最初のノードからデータを複製します。次に、`bdr_init_physical` を使用して `region-a` グループに参加します。

この例は、物理的結合を示しています。

```
apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: region-a
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  cnp:
    joinMethod: physical
    storage:
```

(次のページに続く)

(前のページからの続き)

size: 1Gi

...

4.22.3 グループ参加の参加方法を構成する

クロスリージョンの PGD グループを作成するときに、グループ参加の参加方法を構成できます。新しい PGD グループの最初のノードは、既存の PGD グループに参加します。このコンテキストでは、参加グループはノードグループとは異なります。

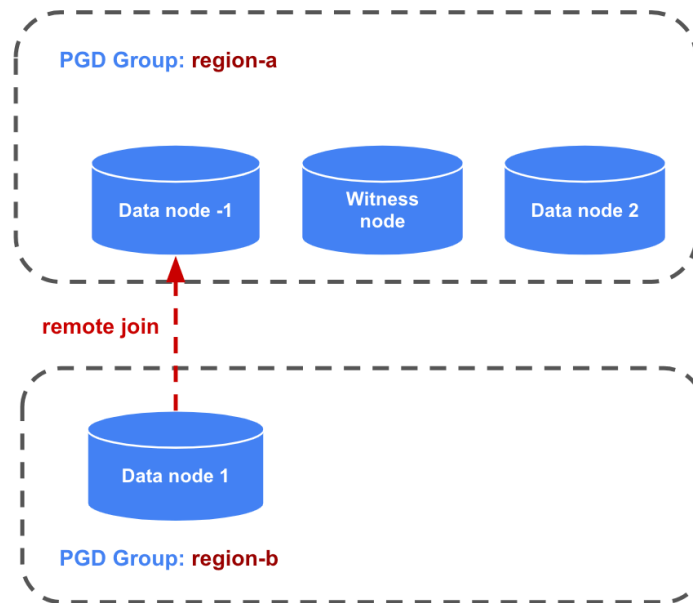


図 4: node join remote group

PGD4K オペレーターは、`spec.pgd.joinMethod` セクションデフォルト `logical` を使用して、PGD グループが相互に参加する方法を決定します。`spec.pgd.discovery` セクションを使用して、参加するターゲットを指定します。

この例は、論理結合を示しています。`spec.pgd.groupJoinMethod=logical`。region-b グループの最初のノードは、`spec.pgd.discovery` で指定されたすべてのホストが利用可能になるまで待機します。次に、`bdr.join_node_group` ファンクションを使用して論理結合を実行する適切なターゲットを特定します。すべてのリモートグループのグループサービスは、通常、このタイプの参加の検出セクションで定義されます。

```

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: region-b
  namespace: region-b
spec:
  
```

(次のページに続く)

(前のページからの続き)

```

instances: 2
proxyInstances: 2
witnessInstances: 1
pgd:
  parentGroup:
    name: world
  groupJoinMethod: logical
  discovery:
    - host: region-a-group
    - host: region-c-group
cnp:
  storage:
    size: 1Gi

```

この例は、spec.pgd.groupJoinMethod=physical で構成された物理的結合を示しています。region-b グループの最初のノードは、spec.pgd.discovery で定義されたすべてのノードをトラバースし、成功した Raft コンセンサスで 1 つのノードを見つけて、bdr_init_physical を使用して物理的ジョインを実行するのを待機します。通常、リモートグループのノードサービスは、このタイプの参加の検出セクションで指定されます。

```

apiVersion: pgd.k8s.enterprisedb.io/v1beta1
kind: PGDGroup
metadata:
  name: region-b
spec:
  instances: 2
  proxyInstances: 2
  witnessInstances: 1
  pgd:
    parentGroup:
      name: world
    groupJoinMethod: physical
    discovery:
      - host: region-a-3-node
      - host: region-a-1-node
      - host: region-a-2-node
  cnp:
    joinMethod: physical
    storage:
      size: 1Gi

```

注釈: グループ参加は本質的にノード参加のタイプであり、特にグループ内の最初のノードが別のグループのノードに参加する場合に発生します。

注釈: Witness ノードは常に論理結合を使用してグループに接続します。
