

Cloud Native PostgreSQL

EnterpriseDB

Contents

- [Cloud Native PostgreSQL](#)
 - [要件](#)
 - [サポートされているPostgreSQLバージョン](#)
 - [メインな機能](#)
 - [このガイドについて](#)
- [Before You Start](#)
 - [Kubernetesの用語](#)
 - [PostgreSQLの用語](#)
 - [クラウドの用語](#)
 - [次にすること](#)
- [Free evaluation](#)
 - [PostgreSQLの評価](#)
 - [EDB Postgres Advancedの評価](#)
- [Use cases](#)
 - [ケース1：Kubernetes内のアプリケーション](#)
 - [ケース2：Kubernetes外のアプリケーション](#)
- [Architecture](#)
 - [読み取り/書き込みワークロード](#)
 - [読み取り専用のワークロード](#)
 - [アプリケーションの展開](#)
 - [DNS解決](#)
 - [環境変数](#)
 - [秘密](#)
- [Installation](#)
 - [Kubernetesでのインストール](#)
 - [Openshiftでのインストール](#)
 - [展開に関する詳細](#)
- [Quickstart](#)
 - [パート1-ローカルKubernetes / Openshiftブレイグラウンドのセットアップ](#)
 - [パート2-Cloud Native PostgreSQLのインストール](#)
 - [パート3- PostgreSQLクラスターをデプロイする](#)
- [Cloud Setup](#)
 - [Microsoft Azure Kubernetes Service \(AKS\)](#)
 - [Amazon Elastic Kubernetes Service \(EKS\)](#)
 - [Google Kubernetes Engine \(GKE\)](#)
- [Bootstrap](#)
 - [initdb](#)
 - [リカバリ](#)
- [Resource management](#)
- [Security](#)
 - [コード](#)
 - [コンテナ](#)
 - [クラスター](#)
- [Failure Modes](#)
 - [活性と準備のプロープ](#)
 - [ストレージスペースの使用](#)
 - [失敗モード](#)
 - [自己修復](#)
 - [手動介入](#)
- [Rolling Updates](#)
- [Backup and Recovery](#)

- クラウド認証情報
- クラスターの設定
- オンデマンドバックアップ
- スケジュールされたバックアップ
- WALアーカイビング
- 回復
- PostgreSQL Configuration
 - `postgresql` セクション
 - `pg_hba` セクション
 - 設定の変更
 - 固定パラメーター
- Storage
 - カスタムPVCテンプレートを使用する
 - インスタンスに使用されるストレージサイズの拡張
- Configuration Samples
- Monitoring
 - ユーザー定義のメトリック
- Exposing Postgres Services
 - Minikubeでのテスト
- Client SSL Connections
 - SSLを使用してポッドに接続する
- Kubernetes Upgrade
- End-to-End Tests
- Cloud Native PostgreSQL Plugin
 - インストール
 - 使用
- License and License Keys
 - 会社レベルのライセンスキー
 - クラスターレベルのライセンスキー
- Container Image Requirements
 - イメージタグの要件
- Operator Capability Levels
 - レベル1-基本インストール
 - レベル2-シームレスなアップグレード
 - レベル3-完全なライフサイクル
 - レベル4-深い洞察
 - レベル5-オートパイロット
- API Reference
 - バックアップ
 - BackupList
 - BackupSpec
 - BackupStatus
 - AffinityConfiguration
 - BackupConfiguration
 - BarmanObjectStoreConfiguration
 - BootstrapConfiguration
 - BootstrapInitDB
 - BootstrapRecovery
 - クラスター
 - ClusterList
 - ClusterSpec
 - ClusterStatus
 - DataBackupConfiguration
 - MonitoringConfiguration
 - NodeMaintenanceWindow
 - PostgresConfiguration
 - RecoveryTarget
 - RollingUpdateStatus
 - S3Credentials
 - StorageConfiguration

- [WalBackupConfiguration](#)
- [ScheduledBackup](#)
- [ScheduledBackupList](#)
- [ScheduledBackupSpec](#)
- [ScheduledBackupStatus](#)
- [Credits](#)

Cloud Native PostgreSQL

クラウドネイティブPostgreSQLは宣言型の構成と不変のインフラとしてDevOpsチームの原則とconceptssuchにプライベート、公共の場で行われているすべてのサポートされているクラスタ上のワークロード、またはハイブリッドクラウドenvironments.CloudネイティブPostgreSQLの付着を管理することにより、設計されています。

高可用性と読み取り専用クエリのオフロードのために、選択されたKubernetes名前空間と共存する単一のプライマリとオプションの数のレプリカで構成されるPostgreSQLclusterを表す「Cluster」と呼ばれる新しいKubernetesリソースを定義します。

同じKubernetesクラスターに存在するアプリケーションは、演算子のみが管理するサービスを使用して、フェイルオーバーまたはスイッチオーバー後のプライマリロールの変更を心配することなく、PostgreSQLデータベースにアクセスできます。Kubernetesクラスターの外部にあるアプリケーションは、TCP経由でサービスを公開するためにIngressオブジェクトを構成する必要があります。

クラウドネイティブのPostgreSQLで動作し、かつ下で利用可能です。

あなたは[evaluate Cloud Native PostgreSQL for free](#).Youは稼動にクラウドネイティブのPostgreSQLを使用するには、有効なライセンスキーを必要とすることができます。

!!! IMPORTANT * 現在、[Operator Capability Levels model](#)に基づいて、ユーザーは「レベルIII-フルライフサイクル」クラウドネイティブPostgreSQLオペレーターの機能セットを期待できます。

要件

Cloud Native PostgreSQLには、Kubernetes 1.16以降が必要です。AWS、Google、Azure（マルチプルのアベイラビリティゾーンを使用）でテスト済みです。

Cloud Native PostgreSQLも4.5+で認定されており、.OpenShift Container Platformから直接入手できます。これは、Red HatによるKubernetesのオープンソースディストリビューションです。

サポートされているPostgreSQLバージョン

PostgreSQLおよびEDB Postgres Advanced 13、12、11、および10が現在サポートされています。

メインな機能

- Kubernetes APIサーバーとの直接統合による高可用性、外部ツールを必要とせずに
- 自己修復機能：
- 最も整合したレプリカを昇格させることによるプライマリインスタンスのフェイルオーバー
- レプリカの自動作成
- 選択したレプリカを昇格させることにより、プライマリインスタンスの計画的な切り替え
- スケールアップ/スケールダウン機能
- 任意の数のインスタンスの定義（最小1-1つのプライマリーサーバ）
- アプリケーションをクラスターの唯一のプライマリーサーバに接続するための* 読み書き*サービスの定義
- *読み取り専用*サービスの定義。ワークロードを読み取るためにアプリケーションを任意のインスタンスに接続します。
- PVCテンプレートを使用したローカル永続ボリュームのサポート
- ポッドでの永続ボリュームストレージの再利用
- PostgreSQLマイナーバージョンおよび演算子アップグレードのローリング更新
- TLS接続とクライアント証明書認証
- S3互換オブジェクトストアへの継続的なバックアップ
- S3と互換性のあるオブジェクトストアバックアップからの完全リカバリとポイントインタイムリカバリ
- 同期レプリカのサポート
- `nodeSelector` を介したノードアフィニティのサポート
- PostgreSQLエラーメッセージの標準出力ロギング

このガイドについて

“Quickstart”の指示に従って、MinikubeまたはKindを使用してローカルKubernetesクラスターでCloud Native PostgreSQLをテストします。

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

!!! Note * このガイドでは主にKubernetesを取り上げていますが、すべての概念はOpenShiftにも拡張できます。

Before You Start

始める前に、KubernetesおよびPostgreSQLに固有の用語を検討することが不可欠です。

Kubernetesの用語

Resource	Description
Node	ノードはKubernetesの仮想マシンまたは物理のマシンのワーカーマシンであり、ポッドの実行に必要なすべてのサービスはコントロールプレーンノードによって管理されます。
Pod	* pod *は、Kubernetesクラスターに展開できる最小のコンピューティングユニットであり、ネットワークとストレージを共有する1つ以上のコンテナで構成されます。
Service	サービスは、ポッドのグループで実行され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し、アプリケーション間のサービス検出、負荷分散、フェイルオーバーなどの重要な機能を標準化する抽象化です。
Secret	シークレットは、パスワード、アクセスキー、トークンなどの少量の機密データを保存し、ポッドで使用するように設計されたオブジェクトです。
Storage Class	ストレージクラスを使用すると、管理者は、プロビジョニング担当者（AWS EBSなど）、再生ポリシー、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、管理者が手動でプロビジョニングしたストレージ、またはストレージクラスコントローラが動的にプロビジョニングし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とは無関係です。通常、PVは、特にパブリッククラウドでは、ネットワークボリュームです。 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的なボリュームです。
Persistent Volume Claim	永続ボリューム要求（PVC）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を含むストレージの要求を表します。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クラスターのロジカルかつ分離されたサブセットであり、より広い物理的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個別の環境を作成できます。
RBAC	役割ベースのアクセス制御も役割ベースのセキュリティとして知られている（RBAC）は、許可されたユーザだけにシステムのネットワークとリソースへのアクセスを制限するためにコンピュータシステムのセキュリティに使用されるメソッドです。Kubernetesには、名前空間とクラスターレベルでロールを制御し、特定のリソースや個人に関連付けるためのネイティブ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	演算子は、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人間の演算子によって通常実行されるステップを自動化するカスタムリソースです。演算子はKubernetesを支援して、リソースの定義された状態が常に観察された状態と一致するようにします。
kubectl	<code>kubectl</code> 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。

Cloud Native PostgreSQLにはKubernetes 1.16以降が必要です。

PostgreSQLの用語

Resource	Description
Instance	ペア* IPアドレスおよびTCPポート*（通常5432）で実行およびリッスンするPostgresサーバプロセス。
Primary	読み取り操作と書き込み操作の両方を受け入れることができるPostgreSQLインスタンス。

Resource	Description
Replica	クラスター内の唯一のプライマリインスタンスからレプリケートするPostgreSQLインスタンスは、先行書き込みログ（WAL）レコードのストリームを読み取ることで更新され続けます。レプリカは、スタンバイまたは二次的なサーバーとも呼ばれます。PostgreSQLは、物理的ストリーミングレプリケーション（async / sync）とファイルベースのログ SHIPPING（async）に依存しています。
Hot Standby	レプリカが読み取り専用ワークロードを受け入れることができるPostgreSQL機能。
Cluster	高可用性（HA）クラスターとしての使用：単一のプライマリとオプションの任意の数のレプリカで構成されるPostgreSQLインスタンスのセット。

クラウドの用語

Resource	Description
Region	クラウド内のリージョンは、 <i>可用性ゾーン</i> で編成された独立した地理的領域です。リージョン内のゾーンには、往復のネットワークレイテンシがほとんどありません。
Zone	クラウド内の <i>可用性ゾーン</i> （ゾーンとも呼ばれる）は、リソースを展開できる領域内の領域です。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隔離された建物に対応します。

次にすること

用語を理解したので、選択したクラウド環境に演算子をデプロイする前に[tote Cloud Native PostgreSQL on your laptop using a local cluster](#)を決定できます。

Free evaluation

Cloud Native PostgreSQLはフリーで評価できます。プロセスはVanilla / Community PostgreSQLとEDB Postgres Advancedで異なります。

用語と詳細については、[“License and License keys” section](#)を参照してください。

PostgreSQLの評価

デフォルトでは、Cloud Native PostgreSQLはCommunity PostgreSQLの利用可能な最新バージョンをインストールします。演算子は、30日間続くクラスターの暗黙的な試用ライセンスを自動的に生成します。

このライセンスは、評価、概念実証、CI / CDパイプラインとの統合などに最適です。

PostgreSQLコンテナイメージは[atquay.io/enterprisedb/postgresql](#)で入手できます。

EDB Postgres Advancedの評価

EDB Postgres AdvancedtooでCloud Native PostgreSQLを使用できます。[EnterpriseDB website](#)からトライアルライセンスキーを要求する必要があります。

EDB Postgres Advancedコンテナイメージが[atquay.io/enterprisedb/edb-postgres-advanced](#)で利用可能です。

ライセンスキーを受け取ったら、`Cluster` デプロイメント構成ファイルの `spec` セクションのEDB Postgres Advancedby設定を使用できます。

- `quay.io/enterprisedb/edb-postgres-advanced` リポジトリを指す `imageName`
- `licenseKey` をライセンスキーに（文字列のフォームで）

[configuration samples](#)セクションの完全な例を参照してください。

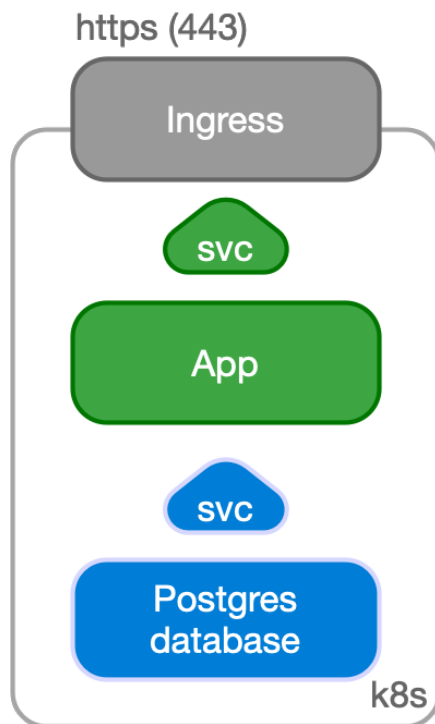
Use cases

クラウドネイティブPostgreSQLは、完全なクラウドネイティブ体験のために、同じKubernetesクラスターに存在するアプリケーションで動作するように設計されています。

ただし、データベースはKubernetesクラスター内でホストできますが、アプリケーションを同時にコンテナ化することはできず、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。

ケース1：Kubernetes内のアプリケーション

典型的なシチュエーションでは、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。

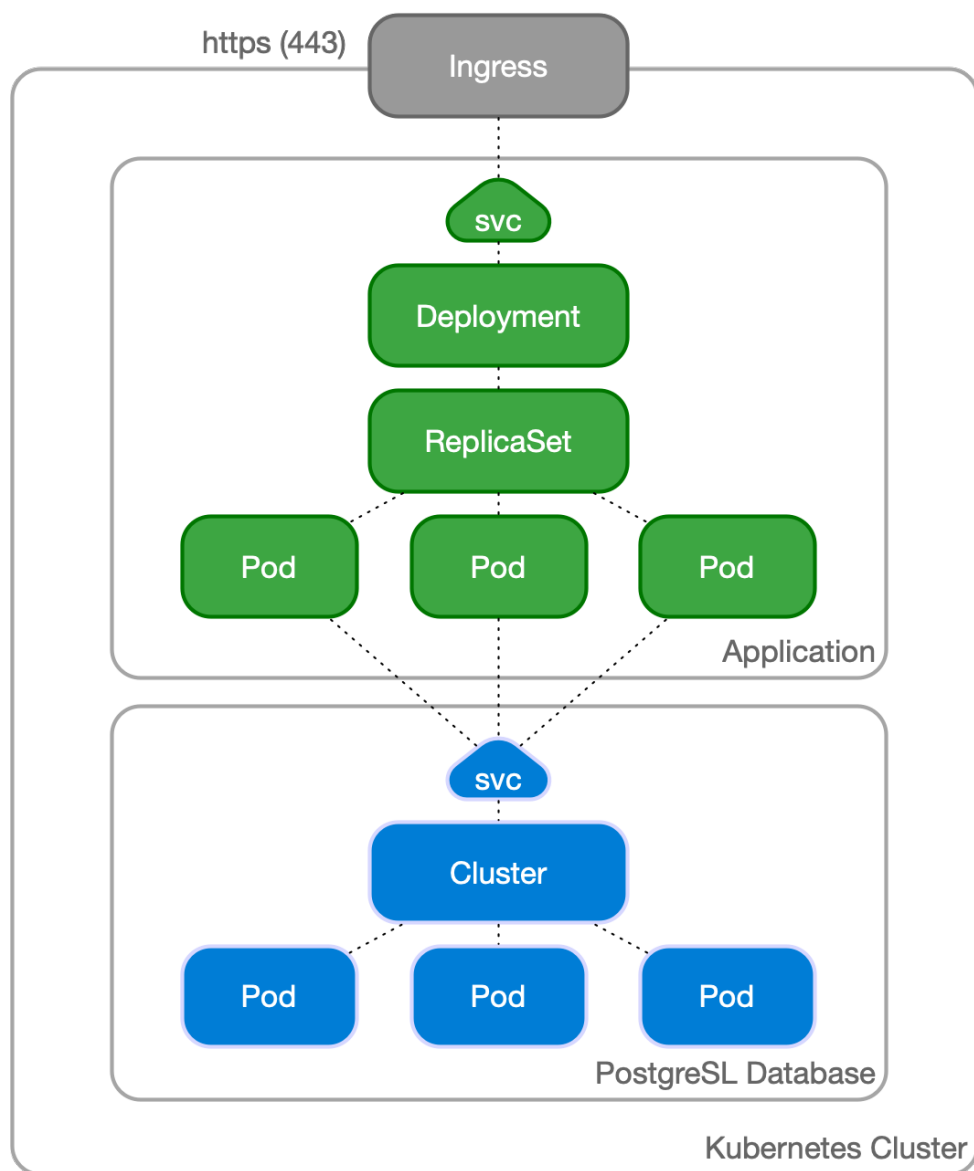


Application and Database inside Kubernetes

通常はステートレスであるアプリケーションは、マルチプルのレプリカが異なるKubernetesノードに分散され、`ClusterIP` サービスを介して内部的に公開されている標準の `Deployment` として管理されます。

サービス経由で、`Ingress` と `the provider` のロード機能によって、エンドユーザに外部に露出されます。

アプリケーションは、バックエンド PostgreSQL データベースを使用して、信頼できる永続的な方法で状態を追跡します。アプリケーションは、Cloud Native PostgreSQL によって定義された `Cluster` リソースによって公開された読み取り/書き込みサービスを指します。これは、TLS 接続を介して現在のプライマリインスタンスを指します。`Cluster` リソースには、単一のプライマリアーキテクチャとマルチプルのスタンバイアーキテクチャのロジックが組み込まれているため、Postgres で高可用性クラスターを管理する複雑さが隠されています。

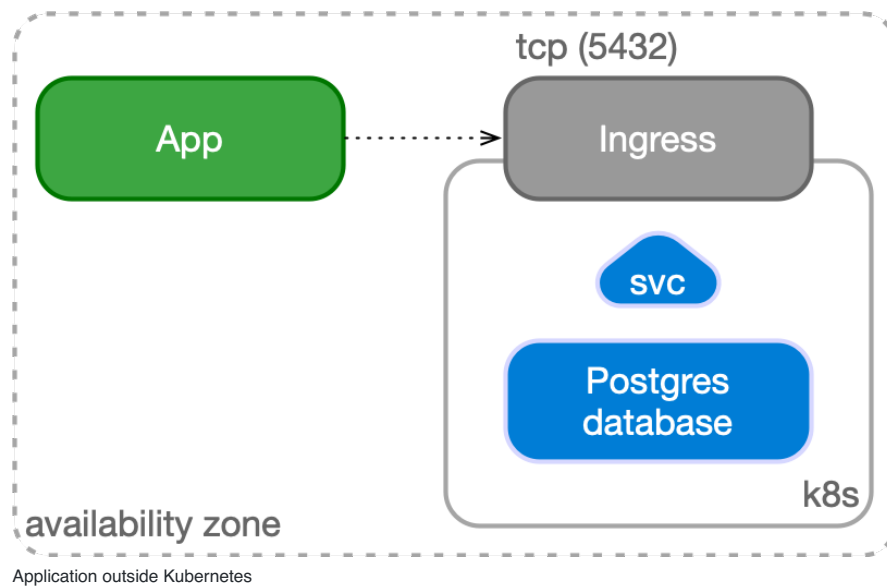


Close-up view of application and database inside Kubernetes

ケース2：Kubernetes外のアプリケーション

もう1つの可能なユースケースは、Kubernetes内でPostgreSQLデータベースを管理し、外部にアプリケーションを配置することです（例、仮想化環境）。この場合、PostgreSQLはKubernetesで定義されたIngressresourceに対応するIPアドレス（またはホスト名前）とTCPポートで表されます。

アプリケーションは、PostgreSQLへのTLS接続の恩恵を受けることができます。



Architecture

高可用性を実現するために、PostgreSQLデータベース管理システムは、** Write Ahead Log (WAL) シッピングに基づくビルトインの物理的レプリケーション**機能を管理者に提供します。

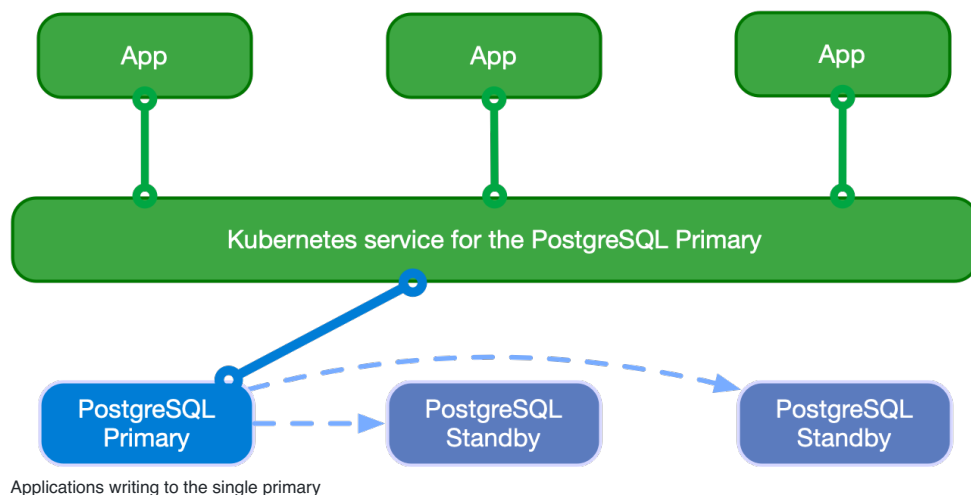
PostgreSQLは、非同期および同期のストリーミングレプリケーションと、非同期ファイルベースのログシッピング（通常、例、WAL ファイルをオブジェクトストアに格納するためのフォールバックオプションとして使用）をサポートしています。レプリカは通常スタンバイサーバと呼ばれ、ホットスタンバイ機能のおかげで、読み取り専用のワークロードにも使用できます。

Cloud Native PostgreSQLは現在、非同期および同期のストリーミングレプリケーションに基づくクラスターをサポートしており、次の仕様でマルチプルのホットスタンバイレプリカを管理します。

- 1つのプライマリ、高可用性のためのオプションのマルチプルのホットスタンバイレプリカ
- アプリケーションで利用可能なサービス：
 - `-rw` : アプリケーションはクラスターの唯一のプライマリインスタンスに接続します
 - `-ro` : アプリケーションは、読み取り専用ワークロードの唯一のホットスタンバイレプリカに接続します
 - `-r` : アプリケーションは読み取り専用ワークロードのインスタンスのいずれかに接続します
- PostgreSQLクラスターの復元力を高めるために、シェアドナッシングアーキテクチャを推奨：
- PostgreSQLインスタンスは異なるKubernetesワーカーノードに存在し、ネットワークのみを共有する必要があります
- PostgreSQLインスタンスは、同じリージョンの異なるアベイラビリティゾーンに存在できます
- PostgreSQLクラスターのすべてのノードは同じリージョンに存在する必要があります

読み取り/書き込みワークロード

次の図に示すように、アプリケーションはKubernetes演算子によって* current プライマリ *として選択されたPostgreSQLインスタンスに接続することを決定できます。



アプリケーションは `-rw` サフィックスサービスを使用できます。

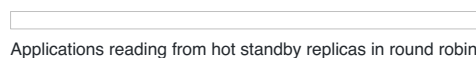
プライマリが一時的または永続的に利用できない場合、Kubernetesは高可用性のために `-rw` をクラスターの別のインスタンスに移動します。

読み取り専用のワークロード

!!! Important * アプリケーションは、[Hot Standby](#)が提示する制限を認識し、これらのワークロードを処理する際のPostgreSQLの動作方法に精通している必要があります。

アプリケーションは、演算子が使用可能にした `-ro` サービスを介してホットスタンバイレプリカにアクセスできます。このサービスにより、アプリケーションは、プライマリノードから読み取り専用クエリをオフロードできます。

次の図は、アーキテクチャを示しています。



アプリケーションは、接続時に `-r` サービスを通じていつでもPostgreSQLインスタンスにアクセスできます。

アプリケーションの展開

アプリケーションは、同じKubernetesクラスターでCloud Native PostgreSQLによって作成されたサービスと連携するはずですが。

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

これらのサービスはKubernetesクラスターによって完全に管理されており、“[Service](#)” [page of the Kubernetes Documentation](#)で説明されているフォームの仮想IPを実装しています。

!!! Hint * アプリケーションでこれらのサービスを使用し、特定のPostgreSQLインスタンスへの直接接続を避けることを強くお勧めします。後者はクラスターの存続期間中に変更される可能性があります。

これらのサービスは、次の方法でアプリケーションで使用できます。

- DNS解決
- 環境変数

PostgreSQLに接続するためのクレデンシャルに関する限り、演算子によって生成されたシークレットを使用できます。

!!! Warning * 演算子は、`[cluster name]-any` 名前付けの別のサービスを作成します。このサービスは、PostgreSQLインスタンスの検出を管理するために内部的に使用されます。アプリケーションが直接使用することは想定されていません。

DNS解決

この演算子で必要なKubernetes DNSサービスを使用して、特定のサーバーを指すことができます。アプリケーションがPostgreSQLクラスターと同じネームスペースにデプロイされている場合は、サービスの名前を使用するだけで可能です。PostgreSQLクラスターは別のネームスペースに存在します。フル修飾子 `service-name.namespace-name` を使用できます。

DNSは、推奨される発見メソッドです。

環境変数

PostgreSQLクラスターを含む同じネームスペースにアプリケーションをデプロイする場合、環境変数を使用してデータベースに接続することもできます。

例、あなたがあなたのアプリケーションで以下の環境変数を使用することができ、あなたのPostgreSQLクラスターが `pg-database` と呼ばれているとします。

- `PG_DATABASE_R_SERVICE_HOST` : サービスのIPアドレス 読み取り専用ワークロード用のすべてのPostgreSQLインスタンスを指す
- `PG_DATABASE_RO_SERVICE_HOST` : のIPアドレス クラスターのすべてのホットスタンバイレプリカを指すサービス
- `PG_DATABASE_RW_SERVICE_HOST` : のIPアドレス クラスターの* プライマリ*インスタンスを指すサービス

秘密

PostgreSQL演算子は、展開するすべてのPostgreSQLクラスターに対して2つのシークレットを生成します。

- `[cluster name]-superuser`
- `[cluster name]-app`

シークレットには、ユーザー名、パスワード、および `postgres` ユーザとデータベースの* owner *のworking `.pgpass file` がそれぞれ含まれます。

`-app` クレデンシャルは、PostgreSQLクラスターに接続するアプリケーションで使用する必要があるものです。

`-superuser` のものは、管理目的でのみ使用されることになっています。

Installation

Kubernetesでのインストール

演算子マニフェストを直接使用する

演算子は、`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インストールできます。

[latest operator manifest](#)は次のようにインストールできます。

```
kubectl apply -f \
  * https://get.enterprisedb.io/cnp/postgresql-operator-1.2.1.yaml
```

`kubectl` コマンドを実行すると、Cloud Native PostgreSQLがKubernetesクラスターにインストールされます。

次の方法で確認できます。

```
kubectl get deploy -n postgresql-operator-system postgresql-operator-controller-manager
```

Operator Lifecycle Manager (OLM) を使用する

OperatorHubは、コミュニティ向けのオペレータのインデックスであり、the [Operator Lifecycle Manager](#) を介して利用できます。これは、オペレータ用のパッケージ管理システムです。

そのページにリストされているインストール手順に従って、から利用可能なメタデータを使用して、Cloud Native PostgreSQLをインストールできます。

OpenShiftでのインストール

Webインタフェース経由

`kubeadmin` としてコンソールにログインし、`Operator` → `OperatorHub` ページに移動します。

スクロールまたは検索フィルターを使用して、`Cloud Native PostgreSQL` ボックスを見つけます。

演算子を選択して、`Install` をクリックします。デフォルト設定を使用して、次の `Install Operator` で再び `Install` をクリックします。これらの設定の詳細な説明については、[OpenShift documentation](#)を参照してください。

演算子はまもなくすべてのネームスペースで使用可能になります。

OpenShiftクラスターに適用されるセキュリティレベルに応じて、異なるネームスペースで使われる演算子の適切なロールと権限のセットを作成する必要があります。この問題の詳細については、[OpenShift documentation](#)を参照してください。

oc コマンドライン経由

次のように、[subscription](#) を追加して、すべてのネームスペースに演算子をインストールできます。

```
oc apply -f \
  * https://docs.enterprisedb.io/cloud-native-postgresql/latest/samples/subscription.yaml
```

演算子はまもなくすべてのネームスペースで使用可能になります。

詳細については、[onhow to install operators via CL](#)^がOpenShiftの文書されています。

展開に関する詳細

Kubernetesでは、演算子はデフォルトで `postgresql-operator-system` 名前空間に `postgresql-operator-controller-manager` と呼ばれるKubernetes Deployment としてインストールされます。以下を実行することにより、詳細情報を取得できます。

```
kubectl describe deploy -n postgresql-operator-system postgresql-operator-controller-manager
```

あらゆる展開と同様に、レプリカセットの上に置かれ、ローリングアップグレードをサポートします。デフォルトでは、現在1つのレプリカのみをサポートしています。将来のバージョンでは、Kubernetesコントロールプレーンでの展開を可能にするために、マルチプルのレプリカとリーダーの選択、および汚染と許容をサポートする予定です。

ケースでポッドを実行しているノードがもはや到達できない、ポッドは、別のノードで再スケジュールされます。

OpenShiftに関する限り、詳細は選択したインストールメソッドによって異なる場合があります。

Quickstart

このセクションでは、またはローカルのKubernetesクラスターでCloud Native PostgreSQLを使用して、ラップトップ/コンピューターでPostgreSQLクラスターをテストする方法について説明します。

!!! Tip “Live demonstration” * まだローカルにインストールしたくないですか？ブラウザで直接デモを試してください：

- [Cloud Native PostgreSQL Operator Interactive Quickstart](#)

RedHat OpenShift Container Platformユーザーは、OpenShiftの[Red Hat CodeReady Containers \(CRC\)](#)でCloud Native PostgreSQLの認定演算子をテストできます。

!!! Warning * このセクションに含まれる命令は、デモンストレーション、テスト、練習のみを目的としており、稼働に使用することはできません。

他のKubernetesアプリケーションと同様に、Cloud Native PostgreSQLは、YAMLで記述された通常のマニフェストを使用して展開されます。

このページの指示に従うことで、ローカルKubernetes / OpenshiftインストールでPostgreSQLclusterをスタート、それを試すことができます。

!!! Important * Kubernetesクラスターに接続するオーダーにマシンに `kubectl` がインストールされていることを確認します。OpenShiftにCRCを使用する場合は `oc` をインストールします。KubernetesのドキュメントまたはOpenshiftの文書に従ってください。

!!! Note * Openshiftを実行している場合、この文書で `kubectl` について言及するたびに `oc` を使用します。 `kubectl` コマンドは、`oc` コマンドと互換性があります。

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

最初のパートは、Minikube、Kind、またはCRCのインストールについてです。システムについて時間をかけて読んで、どちらのシステムに進むかを決めてください。いずれかのシステムをセットアップしたら、パート2に進んでください。

ミニクベ

Minikubeは、Kubernetesをローカルで簡単に実行できるツール。Minikubeは、Kubernetesを毎日試したり開発したりするために、ラップトップ上の仮想マシン（VM）内で単一ノードKubernetesクラスターとして実行されます。通常、VirtualBoxと組み合わせて使用します。

詳細については、ローカルの個人環境の公式[Kubernetes documentation on how toinstall Minikube](#)を参照してください。インストールしたら、次のコマンドを実行してminikubeクラスターを作成します。

```
minikube start
```

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

```
kubectl get nodes
```

minikube という1つのノードが表示されます。

種類

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

[Quickstart](#)の指示に従って環境に `kind` をインストールし、次に以下を使用してKubernetesクラスターを作成します。

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

CodeReadyコンテナ（CRC）

[Download RedHat CRC](#)と `PATH` のディレクトリ内のバイナリを移動します。

```
crc setup
crc start
```

`crc start` の出力は、処理方法を説明しています。次に、`crc oc-env` コマンドの出力を実行する必要があります。その後、印刷された `oc login` コマンドで `kubeadmin` としてログできます。`crc console` を実行しているWebコンソールを開くこともできます。ユーザーとパスワードは `oc login` コマンドの場合と同じです。

CRCにはStorageClassが付属していないため、設定する必要があります。[Dynamic volume provisioning wiki page](#)に従って `rancher/local-path-provisioner` をインストールできます。

パート2-Cloud Native PostgreSQLのインストール

ラップトップでKubernetesまたはOpenShiftをインストールして実行しているので、Cloud Native PostgreSQLのインストールを続行できます。

["Installation"](#)セクションを参照して、PostgreSQLクラスターの展開を続行してください。

パート3- PostgreSQLクラスターをデプロイする

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

[cluster-example.yaml](#) サンプルファイルは、デフォルトのストレージクラスを使用してシンプル `Cluster` を定義し、ディスクベースを割り当てます。

```
# Example of PostgreSQL cluster
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  * name: cluster-example
spec:
  * instances: 3
  * # Example of rolling update strategy:
  * # - unsupervised: automated update of the primary once all
  * #                   replicas have been upgraded (default)
  * # - supervised: requires manual supervision to perform
  * #                   the switchover of the primary
  * primaryUpdateStrategy: unsupervised
  * # Require 1Gi of space
  * storage:
  *   size: 1Gi
```

!!! Note “There’s more” * 利用可能なオプションの詳細については、[“API Reference” section](#)を参照してください。

3-ノードのPostgreSQLを作成するには、次のコマンドを実行する必要があります。

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

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

```
kubectl get pods
```

デフォルトでは、<演算子>released.Youが spec セクションofthe Cluster 定義で imageName キーを設定することで、これをオーバライドすることができたのPostgreSQLの最新メジャーバージョンversionof入手可能な最新のマイナーをインストールします。例、PostgreSQLの12.5をインストールします。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  * # [...]
spec:
  * # [...]
  * imageName: quay.io/enterprisedb/postgresql:12.5
  * # [...]
```

!!! Important * 不変のインフラストラクチャパラダイムでは、コンテナイメージの特定のバージョンを常にポイントする必要があります。稼動環境では、 latest や 13 などのタグを使用しないでください。クラスター内の更新ポリシーとバージョンの一貫性に関して予測できないシナリオが発生する可能性があります。

Cloud Setup

このセクションでは、 Cluster などを使用してパブリッククラウド内のクラスターでPostgreSQL高可用性クラスターの展開と管理を調整する方法について説明します。他のKubernetesアプリケーションと同様に、YAMLで記述された通常のマニフェストを使用して展開されます。

Cloud Native PostgreSQL Operatorは、次のパブリッククラウド環境で体系的にテストされています。

- [Microsoft Azure Kubernetes Service \(AKS\)](#)
- [Amazon Elastic Kubernetes Service \(EKS\)](#)
- [Google Kubernetes Engine \(GKE\)](#)

このページで説明されている手順が完了し、 kubectl が目的のクラスターに接続できたら、the["Installation"](#)にある指示に従って演算子をインストールし、 PostgreSQL Clusters の作成を開始できます。セクション。

!!! Important * kubectl はセットアップを続行するために必要です。

Microsoft Azure Kubernetes Service (AKS)

Microsoftの文書されている["Quickstart: Deploy an Azure Kubernetes Service \(AKS\) cluster using the Azure portal"](#)に含まれている指示に従って、AKSでKubernetesクラスターをセットアップします。

特に、 az aks get-credentials コマンドを使用して kubectl を設定し、Kubernetesクラスター（ myResourceGroup グループのリソースを使用して myAKSCluster と呼ばれる）に接続する必要があります。

```
az aks get-credentials --resource-group myResourceGroup --name myAKSCluster
```

!!! Note * Azureポータルから myAKSCluster クラスターとリソースグループ myResourceGroup の名前を変更できます。

Azureディスクで動作するストレージクラスのいずれかを使用できます。

- default
- managed-premium

!!! Seealso ["About AKS storage classes"](#) * AKSで利用可能なストレージクラスの詳細と詳細については、["Storage classes" section in the official documentation from Microsoft](#)を参照してください。

Amazon Elastic Kubernetes Service (EKS)

AWS文書にある["Creating an Amazon EKS Cluster"](#)に含まれる指示に従って、EKSでKubernetesクラスターをセットアップします。

!!! Important * Amazonはノードが作成できるポッドの数に制限を設けていることに注意してください。クラスターの作成時に使用することを選択したインスタンスのタイプによって異なります。

セットアップ後、 kubectl は新しく作成されたEKSクラスターを指す必要があります。

デフォルトでは、クラスターの作成後に gp2 ストレージクラスを使用できます。ただし、Amazon EKSは、 Clusters 'ボリューム用の他のストレージクラスを作成するために活用できる複数のストレージタイプを提供します。

- gp2 : 汎用SSDボリューム

- `io1` : プロビジョニングされたIOPS SSD
- `st1` : スループット最適化HDD
- `sc1` : コールドHDD

!!! Seealso “About EKS storage classes” * EKSで使用可能なストレージクラスの詳細と詳細については、AWSのオフィシャルドキュメントとKubernetes文書を参照してください。

Google Kubernetes Engine (GKE)

Google Cloudの文書されている“[Creating a cluster](#)”に含まれている指示に従って、GKEでKubernetesクラスターをセットアップします。

!!! Warning * Google Kubernetes Engineは、推奨されるCoreDNSではなく、廃止された `kube-dns` サーバーを使用します。Cloud Native PostgreSQL Operatorを使用するには、`kube-dns` を無効にし、`coredns` に置き換える必要があります。

GKEクラスターで `kube-dns` を `coredns` に置き換えるには、次の手順に従います。

```
kubectl scale --replicas=0 deployment/kube-dns-autoscaler --namespace=kube-system
kubectl scale --replicas=0 deployment/kube-dns --namespace=kube-system
git clone https://github.com/coredns/deployment.git
./deployment/kubernetes/deploy.sh | kubectl apply -f -
```

デフォルトでは、クラスターの作成後、標準のハードを使用して `standard` ストレージクラスを使用できます。他のストレージタイプの場合、特定のストレージクラスを作成する必要があります。

!!! Seealso “About GKE storage classes” * GKEで利用可能なストレージタイプの詳細と詳細については、Kubernetesのドキュメントと、Google Cloudのオフィシャルドキュメントの関連文書を参照してください。

Bootstrap

このセクションでは、新しいPostgreSQLクラスターを作成するために必要なオプションとその背後にあるデザイン原理について説明します。

PostgreSQLクラスターが定義されている場合、クラスター仕様の `bootstrap` セクションを使用してブートストラップメソッドを構成できます。

次の例では：

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  * name: cluster-example-initdb
spec:
  * instances: 3
  * bootstrap:
  *   initdb:
  *     database: appdb
  *     owner: appuser
  * storage:
  *   size: 1Gi
```

`initdb` ブートストラップメソッドが使用されます。

現在、次のブートストラップメソッドをサポートしています。

- `initdb` : 空のPostgreSQLクラスターを初期化します
- `recovery` : 既存のバックアップから復元するPostgreSQLクラスターを作成します 使用可能なすべてのWALファイルを再生します。

initdb

`initdb` ブートストラップメソッドは、新しいPostgreSQLクラスターをゼロから作成するために使用されます。別に指定しない限り、これはデフォルト。

次の例には、`initdb` 構成の完全な構造が含まれています。

```

apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  * name: cluster-example-initdb
spec:
  * instances: 3
  * superuserSecret:
    * name: superuser-secret
  * bootstrap:
    * initdb:
      * database: appdb
      * owner: appuser
      * secret:
        * name: appuser-secret
  * storage:
    * size: 1Gi

```

上記のブートストラップの例は次のとおり例。

1. PostgreSQLのネイティブ `initdb` コマンドを使用して、新しい PGDATA フォルダーを作成します2. 名前付けのシークレットからスーパーユーザパスワードを設定します。名前付けの非特権ユーザを作成します。 `appuser-secret` `secret5`のパスワードを使用して、後者のパスワードを設定します。 `appuser` ユーザが所有する `appdb` というデータベースを作成します。
- 設定パラダイムよりも規約のおかげで、オペレータにデフォルトデータベース名前（ `app` ）とデフォルトのアプリケーションユーザー名前（データベース名と同じ）を選択させ、スーパーユーザとアプリケーションユーザの両方に安全なパスワードをランダムに生成させることができますPostgreSQLで。

あるいは、上記の例で説明したように、パスワードを生成し、秘密として保存し、 PostgreSQLクラスターで使用できます。

提供されるシークレットは、 kubernetes.io/basic-auth typeの仕様に準拠する必要があります。演算子は、 `username` のフィールドを無視して、シークレットの `password` フィールドのみを使用します。アプリケーション接続にシークレットを再利用する予定がある場合は、 `username` フィールドを `owner` と同じ値に設定できます。

以下は、 `basic-auth` シークレットの例です。

```

apiVersion: v1
data:
  * password: cGFzc3dvcmQ=
kind: Secret
metadata:
  * name: cluster-example-app-user
type: kubernetes.io/basic-auth

```

アプリケーションデータベースは、 `applicationdata` を格納するために使用する必要があるデータベースです。アプリケーションは、アプリケーションデータベースを所有するユーザでクラスターに接続する必要があります。

演算子の!!! Important * 将来の実装では、宣言型の構成方式で、追加のユーザーを作成することができる場合があります。

スーパーユーザと `postgres` データベースは、クラスターを構成するために演算子が使用することになっています。

データベース名前を指定しない場合、演算子は慣例に従い、 `app` データベースを作成し、 `* defaulting webhook *`を使用してクラスター定義に追加します。データベースを所有するユーザは、代わりにデータベース名前にデフォルト設定されます。

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

!!! Important * 現時点では、スーパーユーザシークレットの名前の変更はクラスターに適用されません。

実際のPostgreSQLデータディレクトリは、 `initdb` PostgreSQLコマンドの呼び出しによって作成されます。そのコマンドにカスタムオプションを追加する必要がある場合（つまり、テンプレートデータベースに使用されるロケールを変更する、またはデータチェックサムを追加する）、次の例のようにそれらを `options` セクションに追加できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  * name: cluster-example-initdb
spec:
  * instances: 3
  * bootstrap:
    * initdb:
      * database: appdb
      * owner: appuser
      * options:
        * - "-k"
        * - "--locale=en_US"
      * storage:
        * size: 1Gi
```

互換性機能

EDB Postgres Advancedは、ブレーンコミュニティPostgreSQLに多くの互換性機能を追加します。詳細については、[EDB Postgres Advanced](#)をご覧ください。

これらの機能は、EPASでのクラスター作成時に既に有効になっており、コミュニティPostgreSQLイメージではサポートされていません。それらを無効にするには、次の例のように `initdb` セクションで `redwood` フラグを使用します：

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  * name: cluster-example-initdb
spec:
  * instances: 3
  * imageName: <EPAS-based image>
  * licenseKey: <LICENSE_KEY>
  * bootstrap:
    * initdb:
      * database: appdb
      * owner: appuser
      * redwood: false
      * storage:
        * size: 1Gi
```

!!! Important * EDB Postgresの拡張がスタートする有効なライセンスキー（試用版または稼働）が必要です。

リカバリ

`recovery` ブートストラップモードでは、既存のバックアップ新しいクラスターを作成できます。“[Backup and recovery](#)” pageで `recovery` featureの詳細を見つけることができます。

次の例には、`recovery` セクションの完全な構造が含まれています。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  * name: cluster-example-initdb
spec:
  * instances: 3
  * superuserSecret:
    * name: superuser-secret
  * bootstrap:
    * recovery:
      * backup:
        * name: backup-example
      * storage:
        * size: 1Gi
```

このブートストラップメソッドでは、復元ニーズバックアップへのリファレンスのみを指定できます。

アプリケーションデータベース名前とアプリケーションデータベースユーザは、復元されるバックアップから保持されます。演算子は現在、Kubernetesクラスター自分自身の通常のメンテナンスアクティビティのパートであるため、基になる秘密をバックアップしようとはしていません。

`superuserSecret` を指定しない場合、安全でランダムパスワードで新しいパスワードが自動的に生成されます。次に、その秘密を使用して、クラスターの `postgres` ユーザのパスワードをリセットします。

デフォルトでは、リカバリはデフォルトのターゲットタイムラインで利用可能な最新のWALまで継続します（PostgreSQLの場合は `cu`

`rrent`、バージョン12以上の場合は `latest` ）。

ポイントインタイムリカバリ

最新のものまですべてのWALを再生する代わりに、任意の時点でWALの再生を停止するようにPostgreSQLに依頼することができます。PostgreSQLはこの手法を使用して* point-in-time *リカバリを実装します。これにより、ペースバックアップの取得後いつでもデータベースをその状態に復元できます。

演算子は、次の例のようにリカバリターゲットが指定されている場合、この機能が動作するために必要な構成パラメーターを生成します。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  * name: cluster-restore-pitr
spec:
  * instances: 3
  * storage:
    * size: 5Gi
  * bootstrap:
  * recovery:
    * backup:
      * name: backup-example
    * recoveryTarget:
      * targetTime: "2020-11-26 15:22:00.00000+00"
```

`targetTime` のほかに、次の基準を使用してリカバリを停止できます。

- `targetXID` は、リカバリを続行するトランザクションIDを指定します
- `targetName` はリストアポイントを指定します（ `pg_create_restore_point` で作成） リカバリの続行先）
- `targetLSN` は、ログ先行書き込みの場所のLSNを指定します リカバリが進みます
- `targetImmediate` は、一貫した状態になったらすぐに停止するように指定します
 - 達した

each `recoveryTarget` 構成では、上記のターゲットから1つだけ選択できます。

さらに、 `targetTLI` を指定して、特定のタイムラインに強制的にリカバリできます。

デフォルトでは、以前のパラメータは排他的であると見なされ、リカバリターゲットの直前で停止します。次の例のように、 `exclusive` パラメータを `false` に設定して、リカバリターゲットの直後で停止する包括的な動作を要求できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  * name: cluster-restore-pitr
spec:
  * instances: 3
  * storage:
    * size: 5Gi
  * bootstrap:
  * recovery:
    * backup:
      * name: backup-example
    * recoveryTarget:
      * targetName: "maintenance-activity"
      * exclusive: false
```

Resource management

典型的なKubernetesクラスターでは、ポッドは無制限のリソースで実行されます。デフォルトでは、必要なだけのCPUとRAMを使用できます。

Cloud Native PostgreSQLを使用すると、管理者は2つのノブを使用して、マニフェストの `resources` セクションを介して、クラスターのポッドによるリソース使用を制御および管理できます。

- `requests` : 初期要件
- `limits` : リソースのニーズが動的に増加した場合の最大使用量

たとえば、次の例に、32MiB（128MiBにスケーラビリティ）のRAMと50mのCPU（100mにスケーラビリティ）の初期RAMを要求できます。

```
* resources:
*   requests:
*     memory: "32Mi"
*     cpu: "50m"
*   limits:
*     memory: "128Mi"
*     cpu: "100m"
```

メモリ要求および制限は、コンテナに関連付けられているが、メモリrequestand制限を持つようポッドを考えると便利ですされています。ポッドのメモリリクエストは、ポッド内のすべてのコンテナのメモリリクエストの総和です。

ポッドスケジューリングは、制限ではなく要求に基づいています。ポッドは、ポッドのメモリ要求を満たすのに十分なメモリがノードにある場合にのみ、ノードで実行するようにスケジュールされます。

リソースごとに、コンテナを優先度の高いオーダーに3つのQuality of Service (QoS) クラスに分割します。

- 保証
- パースタブル
- ベストエフォート

詳細については、Kubernetes文書の["Configure Quality of Service for Pods"](#)セクションを参照してください。

PostgreSQLワークロードの場合、「保証」QoSを設定することをお勧めします。

Kubernetesのリソース関連の問題を回避するには、クラスターの作成中に「リソース不足」を処理するためのベストプラクティスを参照できます。

- マニフェストファイルのリソースセクションで、メモリとCPUに必要な値を指定します。このようにして、OOM Killed（「OOM」はOut Of Memoryを表します）およびCPU throttle またはその他を回避できます。インスタンスの実行に関するリソース関連の問題。
- クラスターのポッドを「保証」QoSクラスに割り当てるには、制限とリクエストを設定する必要があります。メモリとCPUの両方を同じ値にします。
- 必要なPostgreSQLメモリパラメータをポッドリソースと一貫して指定します（あなたがするように）VMまたは物理的マシンのシナリオ以下を参照）。
- nodeSelectorを使用して、専用ノードでデータベースサーバポッドをセットアップします。["nodeSelector field of the affinityconfiguration resource on the API reference page"](#)を参照してください。

次の例マニフェストを参照できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: postgresql-resources
spec:
  instances: 3
  postgresql:
    parameters:
      shared_buffers: "256MB"
  resources:
    requests:
      memory: "1024Mi"
      cpu: 1
    limits:
      memory: "1024Mi"
      cpu: 1
  storage:
    size: 1Gi
```

上記の例では、shared_buffers の値で shared_buffers パラメータを指定しました。つまり、データをキャッシュするためにPostgreSQLサーバー専用のメモリ量を指定しています（定義されていない場合、このパラメータのデフォルト値は 128MB です）。

shared_buffers の適切な開始値は、システムのメモリの25%です。例、shared_buffers が256 メガバイトの場合、コンテナメモリサイズの推奨値は1 ギガバイトです。つまり、ポッド内ではすべてのコンテナがKubernetesが常に保持する合計1 ギガバイトのメモリ。コンテナを期待どおりに動作させることができます。詳細については、PostgreSQL文書の["Resource Consumption"](#)sectionを参照してください。

!!! Seealso “Managing Compute Resources for Containers” * リソース管理の詳細については、Kubernetes文書の["Managing Compute Resources for Containers"](#)ページを参照してページ。

Security

このセクションには、コード、コンテナ、クラスターの3つの異なるレイヤーで分析されたクラウドネイティブPostgreSQLのセキュリティ

ティに関する情報が含まれています。

!!! Warning * このページに含まれる情報は、Kubernetesクラスターでの通常のInfoSec業務の実行を免除するものであってはなりません。Kubernetes文書の[“Overview of Cloud Native Security”](#)ページに精通してください。

!!! Seealso “About the 4C’s Security Model” * [“The 4C’s Security Model in Kubernetes”](#) ブログの記事を参照して、Cloud Native PostgreSQLのセキュリティでEDBが採用しているアプローチのより良い理解とコンテキストを取得してください。

コード

Cloud Native PostgreSQLのソースコードは、CI / CDパイプラインで直接呼び出されるGoの人気のあるオープンソースリンターを使用して、セキュリティ上の問題を含む静的分析目的で体系的にスキャンされ静的。GolangCI-Lintはいくつかのリンターを実行できまず同じソースコード。

これらの1つは[Golang Security Checker](#)、または単に `gosec` です。これは、ハードコード資格情報、整数などのコードに隠されている既知の脆弱性、脅威、および脆弱性の発見を目的としたルールセットに対してソースのアブストラクト構文ツリーをスキャンするリンターですオーバーフローやSQL-いくつかの名前ます。

!!! Important * CI / CDパイプラインの静的コード分析フェーズの失敗は、Cloud Native PostgreSQLの配信全体のブロッカーです。つまり、各コミットはGolangCI-Lintによって定義されたすべてのリンターに対して検証されます。

ソースコードは、EnterpriseDBの内部CI / CDパイプラインを介して[Coverity Scan by Synopsys](#)を介して定期的に検査されます。

コンテナ

Cloud Native PostgreSQLのパートであるすべてのコンテナイメージは、コミットごとにCI / CDパイプラインを介して自動的に構築されます。このようなイメージには、オペレーターだけでなく、オペランドも含まれます。具体的には、サポートされるすべてのPostgreSQLおよびEDB Postgres Advancedバージョン画像は以下でスキャンされます：

- [Dockle](#)：用語のベストプラクティス コンテナ構築プロセスの
- [Clair](#)：両方で見つかった脆弱性 基盤となるオペレーティングシステム、およびそれらが実行するライブラリとアプリケーション

!!! Important * すべてのオペランドイメージは、ベースイメージおよびパッケージレベルでのセキュリティ更新の場合、パイプラインによって1日に1回自動的に再構築され、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更新。

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

- [“Container Image Creation and Deployment Guide”](#)、
 - 米国国防総省（DoD）の防衛情報システム局（DISA）が開発
- [“CIS Benchmark for Docker”](#)、
 - インターネットセキュリティセンター（CIS）が開発

!!! Seealso “About the Container level security” * Cloud Native PostgreSQLのコンテナレベルでのEDBのセキュリティに関するアプローチの詳細については、[“Security and Containers in Cloud Native PostgreSQL”](#) ブログ記事を参照してください。

クラスター

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

ポッドセキュリティポリシー

[Pod Security Policy](#)は、ポッドがクラスターで実行するために満たすニーズのあるセキュリティルールと仕様を定義するKubernetesの方法です。InfoSecの理由により、すべてのKubernetesプラットフォームでそれらを実装する必要があります。

Cloud Native PostgreSQLは、コンテナの実行に**特権**モードを必要としません。PostgreSQLサーバーは `postgres` システムユーザとして実行されます。 `root` として実行する必要のあるコンポーネントは一切ありません。

同様に、ボリュームアクセスには*** privileges** *モードや `root` 特権も必要ありません。適切な権限はKubernetesプラットフォームや管理者によって適切に割り当てられる必要があります。

ネットワークポリシー

`Cluster` リソースによって作成されたポッドをKubernetes[network policies](#)で制御して、IPおよびTCPレベルでインバウンドおよびアウトバウンドのネットワークアクセスを有効/無効にすることができます。

ネットワークポリシーは、このドキュメントの範囲外です。詳細については、Kubernetes文書の[“Network policies”](#)セクションを参照してください。

PostgreSQL

Cloud Native PostgreSQLの現在の実装では、`postgres` スーパーユーザとデータベース所有者のパスワードと `.pgpass` ファイルが自

動的に作成されます。[“Secrets” section in the “Architecture” page](#)を参照してください。

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

デフォルトでは、すべてのレプリカは、`streaming_replica` という特別なユーザーを使用して、** physicalasyncストリーミングレプリケーションで現在のプライマリインスタンスに接続するように自動的に構成されます。ノード間の接続は暗号化され、認証は TLSクライアント証明書**を介して行われます。

現在、演算子は、管理者が `postgresql` 構成の `pg_hba` セクションのmanifestas/パートに `pg_hba.conf` 行を直接追加することを許可しています。それらのanifestで定義された行は、デフォルトの `pg_hba.conf` に追加されます。

演算子による `pg_hba.conf` の管理方法の詳細については、文書の[“PostgreSQL Configuration” page](#)を参照してください。

!!! Important * の例では、Kubernetesクラスターがプライベートで安全なネットワークで実行されることを想定しています。

Failure Modes

このセクションでは、PostgreSQLが存続期間中にKubernetesクラスターで直面する可能性があるメジャー障害シナリオの概要を説明します。

!!! Important * 発生している障害シナリオがこのセクションでカバーされていない場合は、すぐにEnterpriseDBに連絡してサポートと支援を受けてください。

活性と準備のプロープ

`Cluster` の各ポッドには、** liveness および readiness **[probe](#)を持つ `postgres` コンテナがあります。

データベースは、プロープコマンドは、各チェックのインターバルA10秒で3回失敗した場合、スーパーユーザcredentials.The二つのプロープは失敗をレポートます使用してacceptconnectionsにできるアップしているとあれば生存性と即応プロープがチェック。

現時点では、Kubernetes 1.17でのみ起動プロープが導入されているため、演算子はPodで `startupProbe` を設定しません。

livenessプロープは、 PostgreSQLインスタンスが中断状態にあり、再起動ニーズがあるかどうかを検出するために使用されます。`startDelay` の値は、プロープの実行を遅らせるために使用されます。これは、起動時間が長いインスタンスの再スタートアップを防ぐために使用されます。

ストレージスペースの使用

演算子は、 PostgreSQLインスタンスごとに1つのPVCをインスタンス化して、`PGDATA` コンテンツを格納します。

このようなストレージスペースは、次の2つの場合に再利用できるように設定されています。

- 対応するポッドがユーザによって削除された場合（および新しいポッドが再作成されます）
- 対応するポッドが別のノードで削除およびスケジュールされる場合

演算子が特定のPVCを再利用できないようにするには、Podを削除する前にPVCを削除する必要があります。このために、次のコマンドを使用できます。

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

例：

```
$ kubectl delete -n default pvc/cluster-example-1 --wait=false
persistentvolumeclaim "cluster-example-1" deleted

$ kubectl delete -n default pod/cluster-example-1
pod "cluster-example-1" deleted
```

失敗モード

`Cluster` に属するポッドは、次の方法で失敗する可能性があります。

- ポッドはユーザによって明示的に削除されます。
- `postgres` コンテナのレディネスプロープが失敗します。
- `postgres` コンテナの活性プロープが失敗します。
- Kubernetesワーカーノードが排出されました。
- ポッドがスケジュールされているKubernetesワーカーノードは失敗します。

これらの各障害は、`Cluster` および演算子が管理するサービスに異なる影響を及ぼします。

ユーザによって削除されたポッド

演算子に削除が通知され削除。the `Cluster` に属する新しいポッドは、存在する場合は既存のPVCを再利用するか、そうでない場合は `*プライマリ*` の物理的バックアップから開始して自動的に作成されます。

!!! Important * ポッドを意図的に削除場合、 `PodDisruptionBudget` ポリシーは施行されません。

- `apiserver*` に通知されるとすぐに自己修復が行われます。

準備プロープの失敗

3回失敗すると、ポッドは「準備ができていません」と見なされます。ポッドは引き続き `Cluster` のパートであり、新しいポッドは作成されません。

失敗の原因を修正できない場合は、ポッドを手動で削除することができます。そうでない場合、障害が解決されると、ポッドは以前のロールを再開します。

自己修復は、プロープが3回失敗すると発生します。

活性プロープの失敗

3回失敗すると、 `postgres` コンテナは失敗したと見なされます。ポッドは依然として `Cluster` のパートであり、 `*kubelet*` はコンテナを再起動しようとします。失敗の原因を修正できない場合は、ポッドを手動で削除することができます。

自己修復は、プロープが3回失敗すると発生します。

ドレインされたワーカーノード

ポッドはワーカーノードから削除され、サービスから削除されます。新しいポッドは、 `nodeMaintenanceWindow` パラメーターの `reusePVC` オプションが `off` に設定されている場合（デフォルト：メンテナンスウィンドウ中に `on` 、そうでない場合は `off` ）、 `プライマリ` の物理的バックアップとは異なるワーカーノードで作成されます。

`PodDisruptionBudget` は、準備ができていないノードが少なくとも1つある場合、ポッドの削除を防ぐことができます。

- `apiserver*` に通知されるとすぐに自己修復が行われます。

ワーカーノードの障害

ノードに障害が発生したため、 `*kubelet*` は活性プロープと準備プロープを実行しません。その特定の障害原因に対してKubernetesクラスター管理者が設定した許容秒数後に、ポッドは削除としてマークされます。 `Kubernetes` クラスターの構成方法に基づいて、ポッドは以前にサービスから削除される場合があります。

新しいポッドは、 `プライマリ` の物理的バックアップとは異なるワーカーノードで作成されます。 `Kubernetescluster` のそのパラメータのデフォルト値は5分です。

自己回復は、 `tolerationSeconds` の後に発生します。

自己修復

失敗したポッドがスタンバイの場合、ポッドは `-r` サービスおよび `-ro` サービスから削除されます。ポッドは、使用可能な場合はそのPVCを使用して再起動されます。そうでない場合、現在のプライマリのバックアップから `newpod` が作成されます。 `-r` サービスへと準備ができたときに `-ro` サービスを再度追加することがpodwill。

失敗したポッドがプライマリ場合、演算子はステータスが準備完了でレプリケーションラグが最小のアクティブなポッドをプロモート、 `-rw service` をポイントします。失敗したポッドは、 `-r` サービスおよび `-ro` サービスから削除されサービス。他のスタンバイは、新しいプライマリから複製をスタートます。前のプライマリは、そのPVCが使用可能な場合、 `pg_rewind` を使用して自分自身を新しいものと同期します。そうでない場合、現在のプライマリのバックアップから新しいスタンバイが作成されます。

手動介入

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

!!! Important * このような場合、 `EnterpriseDB` エンジニアリングチームのサポートと支援なしにマニュアルオペレーションを実行しないでください。

Rolling Updates

演算子は、アプリケーションがクラスターに対して実行されている間にクラスターで使用されるPostgreSQLバージョンを変更できます。

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

ローリングアップグレードは、次の場合に開始されます。

- ユーザがクラスター仕様の `imageName` 属性を変更します。
- 演算子が更新された後、Podが最新のインスタンスを実行保証にします マネージャ;

PostgreSQL構成の変更によりリスタートが必要な場合 適用されます。

演算子は、すべてのレプリカのアップグレード処理を開始します。一度に1つのポッドが、シリアルが最も高いレプリカから開始されます。

プライマリは、アップグレードされる最後のノードです。このオペレーションは構成可能で、 `primaryUpdateStrategy` オプションによって管理され、次の2つの値を受け入れます。

- `unsupervised` : ローリング更新プロセスはKubernetesによって管理されます そして* `switchover` *操作で完全に自動化されています オペレーション すべてのレプリカがアップグレードされたら開始
- `supervised` : ローリング更新プロセスはすぐに中断されます すべてのレプリカがアップグレードされ、完了のみ可能になった後 管理者によってトリガーされるマニュアルスイッチオーバー `kubectl cnp promote [cluster] [pod]` 。プラグインはからダウンロードできます [kubectl-cnp](#) [project page](#) GitHubで。

デフォルトの推奨値は `unsupervised` です。

アップグレードでは、Cloud Native PostgreSQLのIDが保持され、データのクローンは再作成されません。ポッドは削除され、同じPVCで再度作成されます。

ローリング更新プロセスの間、サービスエンドポイントはクラスターのステータスを反映するように移動するため、アプリケーションは更新のノードを無視します。

Backup and Recovery

演算子は、[Barman](#)ツールに基づく継続的なバックアップインフラストラクチャを調整できます。多くのPostgreSQLインスタンスをバックアップするBarmanサーバーで従来のアーキテクチャを使用する代わりに、演算子は `barman-cloud-wal-archive` および `barman-cloud-backup` ツールを使用します。その結果、ベースバックアップは* `tarballs` *になります。ベースバックアップとWALファイルの両方を圧縮および暗号化できます。

これには、 `barman-cli-cloud` がインストールされたイメージが必要です。コミュニティPostgreSQLイメージと`latest barman-cli-cloud` パッケージで構成されているため、このスコープでイメージ `quay.io/enterprisedb/postgresql` を使用できます。

クラウド認証情報

APIがAWS S3と互換性があるサービスのバックアップファイルをアーカイブできます。環境に関する次の情報が必要になります。

- `ACCESS_KEY_ID` : 使用されるアクセスキーのID S3でファイルをアップロードするには
- `ACCESS_SECRET_KEY` : 前のアクセスキーの秘密のパート
- `ACCESS_SESSION_TOKEN` : 必要な場合のオプションのセッショントークン

使用するアクセスキーには、バケットにファイルをアップロードするパーミッションが必要です。そのため、資格情報を使用してk8sシークレットを作成する必要があり、次のコマンドを使用して作成できます。

```
kubectl create secret generic aws-creds \
  * --from-literal=ACCESS_KEY_ID=<access key here> \
  * --from-literal=ACCESS_SECRET_KEY=<secret key here>
# --from-literal=ACCESS_SESSION_TOKEN=<session token here> # if required
```

資格情報はKubernetes内に保存され、インストール時に保存時の暗号化が設定されている場合は暗号暗号化れます。

クラスターの設定

S3

その秘密を考えると、次の例のようにクラスターを構成できます。

```

apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
[...]
spec:
  * backup:
  *   barmanObjectStore:
  *     destinationPath: "<destination path here>"
  *   s3Credentials:
  *     accessKeyId:
  *       name: aws-creds
  *       key: ACCESS_KEY_ID
  *     secretAccessKey:
  *       name: aws-creds
  *       key: ACCESS_SECRET_KEY

```

宛先パスは、インスタンスがWALファイルをアップロードできるフォルダーを指すすべてのURL（eg `s3://BUCKET_NAME/path/to/folder` など）にすることができます。

他のS3互換オブジェクトストレージプロバイダー

MinIOやLinode Object StorageなどのS3互換オブジェクトストレージを使用している場合、デフォルトのS3を使用する代わりにエンドポイントを指定できます。

この例では、region `us-east1` のLinodeの `bucket` バケットを使用します。

```

apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
[...]
spec:
  * backup:
  *   barmanObjectStore:
  *     destinationPath: "<destination path here>"
  *     endpointURL: bucket.us-east1.linodeobjects.com
  *   s3Credentials:
  *     [...]

```

MinIOゲートウェイ

オプションで、S3、GCS、Azureなどの他のクラウドストレージソリューションにバックアップオブジェクトをリレーする共通インタフェースとしてMinIO Gatewayを使用できます。詳細については、[MinIO official documentation](#)を参照してください。

具体的には、Cloud Native PostgreSQLクラスターは、以前に作成された資格情報とサービスを使用して、エンドポイントとして `localMinIO`ゲートウェイを直接指すことができます。

MinIOシークレットはPostgreSQLクラスターとMinIOインスタンスの両方で使用されるため、同じネームスペースで作成する必要があります。

```

kubectl create secret generic minio-creds \
  * --from-literal=MINIO_ACCESS_KEY=<minio access key here> \
  * --from-literal=MINIO_SECRET_KEY=<minio secret key here>

```

この場合、!!! NOTE "Note" * クラウドオブジェクトストレージの認証情報は、MinIOゲートウェイでのみ使用されます。

PostgreSQLはMinIOゲートウェイに到達できるようにするオーダーに!!! Important * は、MinIOゲートウェイインスタンスにバインドされたポート `9000` に `ClusterIP` サービスを作成する必要があります。

例：

```

apiVersion: v1
kind: Service
metadata:
  * name: minio-gateway-service
spec:
  * type: ClusterIP
  * ports:
  *   - port: 9000
  *     targetPort: 9000
  *     protocol: TCP
  * selector:
  *   app: minio

```

!!! Warning * この文書の執筆時点では、Kubernetesの公式[MinIO Operator](#)はゲートウェイ機能をサポートしていません。そのため、代わりに `deployment` を使用します。

MinIO展開では、クラウドストレージの認証情報を使用してオブジェクトをリモートバケットにアップロードし、バックアップファイルを別の場所にリレーします。

クラウドオブジェクトストレージとしてAWS S3を使用する例です。

```
apiVersion: apps/v1
kind: Deployment
[...]
spec:
  containers:
  - name: minio
    image: minio/minio:RELEASE.2020-06-03T22-13-49Z
    args:
    - gateway
    - s3
    env:
    # MinIO access key and secret key
    - name: MINIO_ACCESS_KEY
      valueFrom:
        secretKeyRef:
          name: minio-creds
          key: MINIO_ACCESS_KEY
    - name: MINIO_SECRET_KEY
      valueFrom:
        secretKeyRef:
          name: minio-creds
          key: MINIO_SECRET_KEY
    # AWS credentials
    - name: AWS_ACCESS_KEY_ID
      valueFrom:
        secretKeyRef:
          name: aws-creds
          key: ACCESS_KEY_ID
    - name: AWS_SECRET_ACCESS_KEY
      valueFrom:
        secretKeyRef:
          name: aws-creds
          key: ACCESS_SECRET_KEY
    # Uncomment the below section if session token is required
    # - name: AWS_SESSION_TOKEN
    #   valueFrom:
    #     secretKeyRef:
    #       name: aws-creds
    #       key: ACCESS_SESSION_TOKEN
  ports:
  - containerPort: 9000
```

Cluster definitionで `endpointURL` としてMinIO Gatewayサービスを構成し、次にバケット名前を選択して `BUCKET_NAME` を置き換えます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
[...]
spec:
  backup:
  barmanObjectStore:
  destinationPath: s3://BUCKET_NAME/
  endpointURL: http://minio-gateway-service:9000
  s3Credentials:
  accessKeyId:
  name: minio-creds
  key: MINIO_ACCESS_KEY
  secretAccessKey:
  name: minio-creds
  key: MINIO_SECRET_KEY
  [...]
```

バックアップを実行する前に、`s3://BUCKET_NAME/` でアーカイブされたWALファイルの存在を確認します。

オンデマンドバックアップ

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

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

演算子は、`barman-cloud-backup` を使用して必要なバックアップを取るためにクラスターの調整をスタートします。ブレーンな `kubectl describe backup <name>` コマンドを使用して、バックアップステータスを確認できます。

```
Name:          backup-example
Namespace:     default
Labels:        <none>
Annotations:   API Version:  postgresql.k8s.enterprisedb.io/v1
Kind:          Backup
Metadata:
  * Creation Timestamp:  2020-10-26T13:57:40Z
  * Self Link:           /apis/postgresql.k8s.enterprisedb.io/v1/namespaces/default/backups/backup-example
  * UID:                 ad5f855c-2ffd-454a-a157-900d5f1f6584
Spec:
  * Cluster:
    * Name:  pg-backup
Status:
  * Phase:      running
  * Started At: 2020-10-26T13:57:40Z
Events:        <none>
```

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

```
Name:          backup-example
Namespace:     default
Labels:        <none>
Annotations:   API Version:  postgresql.k8s.enterprisedb.io/v1
Kind:          Backup
Metadata:
  * * Creation Timestamp:  2020-10-26T13:57:40Z
  * * Self Link:           /apis/postgresql.k8s.enterprisedb.io/v1/namespaces/default/backups/backup-example
  * * UID:                 ad5f855c-2ffd-454a-a157-900d5f1f6584
Spec:
  * * Cluster:
    * * Name:  pg-backup
Status:
  * * Backup Id:          20201026T135740
  * * Destination Path:   s3://backups/
  * * Endpoint URL:       http://minio:9000
  * * Phase:              completed
  * * s3Credentials:
    * * Access Key Id:
    * *   Key:  ACCESS_KEY_ID
    * *   Name: minio
    * * Secret Access Key:
    * *   Key:  ACCESS_SECRET_KEY
    * *   Name: minio
  * * Server Name:  pg-backup
  * * Started At:   2020-10-26T13:57:40Z
  * * Stopped At:   2020-10-26T13:57:44Z
Events:            <none>
```

!!!重要この機能は、スーパーユーザとアプリケーションユーザの秘密をバックアップしません。シークレットは、Kubernetes クラスターの標準バックアップ手順のパートとしてバックアップされることになっています。

スケジュールされたバックアップ

`ScheduledBackup` 名前付けのリソースを作成して、バックアップを定期的にスケジュールすることもできます。後者は `Backup` と似ていますが、`schedule` と呼ばれるフィールドが追加されています。

このフィールドは、[Cron](#) スケジュール仕様であり、先頭にフィールドが追加されます。このスケジュール形式は、Kubernetes `CronJobs` で使用されているものと同じです。

これはスケジュールされたバックアップの例です：

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: ScheduledBackup
metadata:
  * name: backup-example
spec:
  * schedule: "0 0 0 * * *"
  * cluster:
    * name: pg-backup
```

提案された仕様では、毎日午前0時にバックアップをスケジュールします。

WALアーカイビング

宛先パスを選択し、クラウド認証情報を設定するとすぐに、WALアーカイビングが有効になります。

必要に応じて、WALファイルをアップロードしたらすぐに圧縮したり、暗号化したりできます：

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
[...]
spec:
  * backup:
    * barmanObjectStore:
      * [...]
      * wal:
        * compression: gzip
        * encryption: AES256
```

バケットで暗号化を直接設定できます。クラスター設定で暗号化を上書きしない限り、オペレータは暗号化を使用します。

回復

オブジェクトストレージにアップロードされたデータを使用して、バックアップから新しいクラスターをブートストラップできます。演算子は、 `barman-cloud-restore` ツールを使用して回復プロセスを調整します。

バックアップが完了すると、対応するKubernetesリソースには、次の例のように、復元に必要なすべての情報が含まれ例。

```
Name:      backup-example
Namespace: default
Labels:    <none>
Annotations: API Version:  postgresql.k8s.enterprisedb.io/v1
Kind:      Backup
Metadata:
  * * Creation Timestamp:  2020-10-26T13:57:40Z
  * * Self Link:          /apis/postgresql.k8s.enterprisedb.io/v1/namespaces/default/backups/backup-example
  * * UID:                ad5f855c-2ffd-454a-a157-900d5f1f6584
Spec:
  * * Cluster:
    * *   Name:  pg-backup
Status:
  * * Backup Id:      20201026T135740
  * * Destination Path: s3://backups/
  * * Endpoint URL:   http://minio:9000
  * * Phase:          completed
  * * s3Credentials:
    * *   Access Key Id:
    * *     Key:  ACCESS_KEY_ID
    * *     Name: minio
    * *   Secret Access Key:
    * *     Key:  ACCESS_SECRET_KEY
    * *     Name: minio
  * * Server Name:  pg-backup
  * * Started At:   2020-10-26T13:57:40Z
  * * Stopped At:   2020-10-26T13:57:44Z
Events:            <none>
```

次のクラスター定義：

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  * name: cluster-restore
spec:
  * instances: 3
  * storage:
  * size: 5Gi
  * bootstrap:
  * recovery:
  * backup:
  * name: backup-example
```

演算子はクラスターの最初のインスタンスにinitコンテナを挿入し、initコンテナはオブジェクトストレージからバックアップの回復をスタートします。

リカバリプロセスが完了すると、演算子はインスタンスをスタートで、復元されたデータディレクトリの整合性に必要なトランザクションログファイルを復旧できるようにします。

リカバリが完了すると、演算子は必要なパスワードをインスタンスに設定します。新しいプライマリインスタンスは通常どおり起動し、残りのインスタンスはレプリカとしてクラスターに結合します。

このプロセスはユーザに対して透過的であり、ポッドで実行されているinstancemanagerによって管理されます。

オプションで `recoveryTarget` を指定して、特定の時点のリカバリを実行できます。指定しない場合、デフォルトのターゲットタイムラインで利用可能な最新のWAL（最大11のPostgreSQLの場合は `current` 、バージョン12以降の場合は `latest` ）までリカバリが続行されます。

PostgreSQL Configuration

PostgreSQLに精通しているユーザは、インスタンスを設定するための次の2つのファイルの存在を認識しています。

- `postgresql.conf` : PostgreSQLのメインの実行時設定ファイル
- `pg_hba.conf` : クライアント認証ファイル

宣言的な構成とPostgreSQLコンテナの不変性の概念により、ユーザはこれらのファイルに直接触れることはできません。the parameters カスタム設定の使用のため `pg_hba` keys.Aリファレンス介してカスタム `postgresql.conf` と `pg_hba.conf` 設定を定義definitionby Cluster リソースの `postgresql` 部を介して可能Configurationis、サンプル、see [cluster-example-custom.yaml](#) に含まれています。

これらの設定は、すべてのインスタンスで同じです。

!!! Warning ** OpenShiftユーザ： ** OpenShiftユーザインタフェースの現在の制限により、YAMLペインからのみPostgreSQL設定を変更できます。

postgresql セクション

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

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

`custom.conf` ファイルには、ユーザ定義の設定が含まれます。 [more information on the available parameters](#)については、PostgreSQLの文書を参照してください。 `custom.conf` のコンテンツは、次のセクションをこのオーダーで適用することにより、オペレーターによって自動的に生成および維持されます。

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

グローバルデフォルトパラメータは次のとおりです。

```
logging_collector = 'off'
max_parallel_workers = '32'
max_replication_slots = '32'
max_worker_processes = '32'
```

** PostgreSQL 13以降のデフォルトパラメータ**は次のとおりです。

```
wal_keep_size = '512MB'
```

** PostgreSQL **のデフォルトパラメータは次のとおりです。

```
wal_keep_segments = '32'
```

以下のパラメーターは固定であり、演算子によって排他的に制御されます：

```
archive_command = '/controller/manager wal-archive %p'
archive_mode = 'on'
archive_timeout = '5min'
full_page_writes = 'on'
hot_standby = 'true'
listen_addresses = '*'
port = '5432'
ssl = 'on'
ssl_ca_file = '/tmp/ca.crt'
ssl_cert_file = '/tmp/server.crt'
ssl_key_file = '/tmp/server.key'
unix_socket_directories = '/var/run/postgresql'
wal_level = 'logical'
wal_log_hints = 'on'
```

固定パラメータは最後に追加されるため、YAML設定を介してユーザーが上書きすることはできません。これらのパラメーターは、正しいWALarchivingとレプリケーションに必要です。

レプリケーション設定

`primary_conninfo` および `recovery_target_timeline` パラメーターは、クラスター内のインスタンスの状態に応じて演算子によって自動的に管理されます。

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

pg_hba セクション

`pg_hba` は、ポッドで使用される `pg_hba.conf` を作成するために使用されるPostgreSQLホストベース認証ルールの一覧です。

最初のマッチングルールが認証に使用されるため、演算子によって生成された `pg_hba.conf` ファイルは3つのセクションで構成されていますと見なすことができます。

1.ルール2を修正しました。ユーザー定義のルール3。デフォルトのルール

固定ルール：

```
local all all peer

hostssl postgres streaming_replica all cert clientcert=1
hostssl replication streaming_replica all cert clientcert=1
```

デフォルトのルール：

```
host all all all md5
```

結果の `pg_hba.conf` は次のようになります。

```
local all all peer

hostssl postgres streaming_replica all cert clientcert=1
hostssl replication streaming_replica all cert clientcert=1

<user defined rules>

host all all all md5
```

[more information on pg_hba.conf](#) については、PostgreSQLの文書を参照してください。

設定の変更

`Cluster` リソースの `postgresql` セクションを編集して、構成の変更を適用できます。

変更後、クラスターインスタンスはすぐに構成をリロードで変更を適用します。変更にリスタートが必要なパラメータが含まれる場合、演算子はローリングアップグレード。

固定パラメーター

一部のPostgreSQL構成パラメーターは、オペレーターのみが管理する必要があります。演算子は、ユーザがwebhookを使用して設定できないようにします。

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

- `allow_system_table_mods`
- `archive_cleanup_command`
- `archive_command`
- `archive_mode`
- `archive_timeout`
- `bonjour_name`
- `bonjour`
- `cluster_name`
- `config_file`
- `data_directory`
- `data_sync_retry`
- `dynamic_shared_memory_type`
- `event_source`
- `external_pid_file`
- `full_page_writes`
- `hba_file`
- `hot_standby`
- `huge_pages`
- `ident_file`
- `jit_provider`
- `listen_addresses`
- `log_destination`
- `log_directory`
- `log_file_mode`
- `log_filename`
- `log_rotation_age`
- `log_rotation_size`
- `log_truncate_on_rotation`
- `logging_collector`
- `port`
- `primary_conninfo`
- `primary_slot_name`
- `promote_trigger_file`
- `recovery_end_command`
- `recovery_min_apply_delay`
- `recovery_target_action`
- `recovery_target_inclusive`
- `recovery_target_lsn`
- `recovery_target_name`
- `recovery_target_time`
- `recovery_target_timeline`
- `recovery_target_xid`
- `recovery_target`
- `restart_after_crash`
- `restore_command`
- `shared_memory_type`
- `ssl_ca_file`
- `ssl_cert_file`

- `ssl_ciphers`
- `ssl_crl_file`
- `ssl_dh_params_file`
- `ssl_ecdh_curve`
- `ssl_key_file`
- `ssl_max_protocol_version`
- `ssl_min_protocol_version`
- `ssl_passphrase_command_supports_reload`
- `ssl_passphrase_command`
- `ssl_prefer_server_ciphers`
- `ssl`
- `stats_temp_directory`
- `synchronous_standby_names`
- `syslog_facility`
- `syslog_ident`
- `syslog_sequence_numbers`
- `syslog_split_messages`
- `unix_socket_directories`
- `unix_socket_group`
- `unix_socket_permissions`
- `wal_level`
- `wal_log_hints`

Storage

ストレージは、データベースワークロードの重要なコンポーネントです。演算子は、 PostgreSQL インスタンスごとに永続ボリュームクレームを作成し、ポッドにマウントします。

PostgreSQL クラスのストレージを設定する簡単な方法は、次の例のように、特定のサイズのストレージを要求すること例。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  * name: postgresql-storage-class
spec:
  * instances: 3
  * storage:
    * size: 1Gi
```

前の構成を使用すると、生成されたPVCはデフォルトのstorageclassで満たされます。ターゲットKubernetesクラスターにデフォルトのストレージクラスがない場合、またはPVCを既知のストレージクラスで満たす必要がある場合は、カスタムリソースに設定できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  * name: postgresql-storage-class
spec:
  * instances: 3
  * storage:
    * storageClass: standard
    * size: 1Gi
```

カスタムPVCテンプレートを使用する

生成されたPVCをさらにカスタマイズするには、次の例のように、カスタムリソース内にPVCテンプレートを提供できます。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  * name: postgresql-pvc-template
spec:
  * instances: 3
  * storage:
  * pvcTemplate:
  * accessModes:
  *   - ReadWriteOnce
  * resources:
  *   requests:
  *     storage: 1Gi
  * storageClassName: standard
  * volumeMode: Filesystem
```

インスタンスに使用されるストレージサイズの拡張

Kubernetesには[expanding PVCs](#)を許可するAPI、これはデフォルトで有効になっていますが、基礎となる `StorageClass` でサポートする必要があるニーズ。

特定の `StorageClass` がボリューム拡張をサポートしているかどうかを確認するには、ストレージクラスの `allowVolumeExpansion` フィールドを読み取ることができます。

```
$ kubectl get storageclass -o jsonpath='{$.allowVolumeExpansion}' premium-storage
true
```

ボリューム拡張Kubernetes機能を使用する

ストレージクラスがボリューム拡張をサポートしている場合、`Cluster` のサイズ要件を変更できます。演算子はすべてのPVCに変更を適用します。

`StorageClass` が[online volume resizing](#)をサポートしている場合、変更はすぐにポッドに適用されます。基になるストレージクラスがそれをサポートしていない場合、サイズ変更をトリガーするにはポッドを削除する必要があります。

続行する最善の方法は、レプリカから開始し、各Podがバックアップされるのを待機して、一度に1つのPodを削除することです。

ストレージ

ストレージクラスがボリューム拡張をサポートしていないとします。その場合でも、ストレージを増やした新しいPVCを割り当てて、そこにデータベースを移動することにより、異なるPVCでクラスターを再生成できます。このオペレーションは、クラスターに複数のノードが含まれる場合にのみ実行可能です。

それを行う間、次の例のように、`resizeInUseVolumes` フラグを無効にして、演算子が既存のPVCを変更しないようにする必要があります。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  * name: postgresql-pvc-template
spec:
  * instances: 3
  * storage:
  *   storageClass: standard
  *   size: 1Gi
  *   resizeInUseVolumes: False
```

クラスター全体を別の格納領域に移動するには、すべてのPVCとすべてのPodを再作成する必要があります。次の例のように、3つのレプリカを持つクラスターがあるとします。

```
$ kubectl get pods
NAME                READY   STATUS    RESTARTS   AGE
cluster-example-1   1/1     Running   0           2m37s
cluster-example-2   1/1     Running   0           2m22s
cluster-example-3   1/1     Running   0           2m10s
```

異なるPVCを使用してクラスターを再作成するには、クラスター定義を編集してdisable `resizeInUseVolumes` にしてから、すべてのインスタンスを異なるPVCに再作成します。

例として、`cluster-example-3` のストレージを再作成するには、次のことができます。

```
$ kubectl delete pvc cluster-example-3 --wait=false
$ kubectl delete pod cluster-example-3 --wait=false
```

それを行った後、演算子は、サイズ変更されたPVCを使用して別のレプリカの作成を調整します。

```
$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
cluster-example-1                   1/1     Running   0           5m58s
cluster-example-2                   1/1     Running   0           5m43s
cluster-example-4-join-v2bfg        0/1     Completed 0           17s
cluster-example-4                   1/1     Running   0           10s
```

Configuration Samples

このセクションでは、PostgreSQL Cluster をセットアップするための構成ファイルの例を見つけることができます。

- `cluster-example.yaml` : デフォルトのストレージクラスを使用する Cluster の基本的な例。デモンストレーションおよび実験目的の“Quickstart”で説明されているように、MinikubeまたはKindを含む個人用Kubernetesクラスター上。
- `cluster-example-custom.yaml` : `postgresql.conf` のデフォルトのストレージクラスとカスタムパラメータを使用する Cluster の基本的な例 `pg_hba.conf` ファイル
- `cluster-storage-class.yaml` : 指定されたストレージクラスを使用する Cluster の基本的な例。
- `cluster-pvc-template.yaml` : 永続ボリューム要求テンプレートを使用する Cluster の基本的な例。
- `cluster-example-full.yaml` : 使用可能なオプションのほとんどを設定する Cluster の例。

利用可能なオプションのリストについては、“[API Reference](#)” pageを参照してください。

Monitoring

PostgreSQLインスタンスごとに、演算子はポート8000でHTTPを介して、[Prometheus](#)のメトリックのエクスポートを提供します。演算子には、事前定義されたメトリックのセットと、1つ以上の ConfigMap オブジェクトを介して追加のクエリを定義するための高度に構成可能でカスタマイズ可能なシステムが付属しています-そして、将来のバージョン、 Secret も。

エクスポートには次のようにアクセスできます。

```
curl http://<pod ip>:8000/metrics
```

すべてのモニタリングクエリは次のとおりです。

- トランザクションアトミック（クエリーごとに1つのトランザクション）
- `pg_monitor` ロールで実行

`pg_monitor` ロールの詳細については、“[Default roles](#)” section in [PostgreSQL documentation](#)を参照してください。

ユーザー定義のメトリック

ユーザーは、演算子が提供する使用可能なインターフェースを介してメトリックを定義できます。このインターフェースは現在* beta *状態にあり、PostgreSQL Prometheus Exporterの[queries.yaml file](#)に触発されたYAMLファイルを使用して、ConfigMap および Secret オブジェクトとしてのカスタムクエリの定義をサポートしています。

次の例のように、Cluster 定義の `monitoring` セクションで被参照されるように、クエリを ConfigMap で定義する必要があります。

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  * name: cluster-example
spec:
  * instances: 3
  * storage:
  * size: 1Gi
  * monitoring:
  * customQueriesConfigMap:
  *   - name: example-monitoring
  *     key: custom-queries
```

具体的には、`monitoring` セクションは、名前が示すとおり、カスタムクエリのソースとして使用される ConfigMap キー参照のリストをニーズとする`name customQueriesConfigMap` の配列を探しキー。

例：

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  * namespace: default
  * name: example-monitoring
data:
  * custom-queries: |
  *   pg_replication:
  *     query: "SELECT CASE WHEN NOT pg_is_in_recovery()
  *              THEN 0
  *              ELSE GREATEST (0,
  *                            EXTRACT(EPOCH FROM (now() - pg_last_xact_replay_timestamp())))
  *              END AS lag"
  *     primary: true
  *     metrics:
  *       - lag:
  *         usage: "GAUGE"
  *         description: "Replication lag behind primary in seconds"
```

オブジェクトは名前を持っており、上記のクエリーが出力をthefollowingで、 `primary` ノード上で実行されることを `Cluster` .Noteと同じ名前空間でなければなりません。

```
# HELP custom_pg_replication_lag Replication lag behind primary in seconds
# TYPE custom_pg_replication_lag gauge
custom_pg_replication_lag 0
```

このフレームワークにより、カスタムメトリックの定義により、 PostgreSQLクラスター内のデータベースまたはアプリケーションをモニタできます。

Exposing Postgres Services

このセクションでは、NGINX Ingress Controllerを使用して、Kubernetesクラスターの外部からPostgreSQLデータベースにアクセスできるように、PostgreSQLサービスを外部に公開する方法について説明します。

を実行した場合、`default` 名前空間の（プライマリ）および（読み取り専用）サービスを介してクラスター内でアクセスできるデータベースができています。両方のサービスはポート `5432` を使用します。

ポート `5432` の外部アクセスからプライマリインスタンスにアクセスできるmakeにしたいとします。Kubernetesインフラストラクチャに移行する場合の典型的なユースケースは、実際には、レガシーアプリケーションで表されるものであり、簡単または持続的に「コンテナ化」することはできません。賢明な回避策は、仮想マシンまたは物理的サーバーにある可能性が高いアプリケーションが、同じネットワーク内のKubernetesクラスター内のPostgreSQLデータベースにアクセスできるようにすることです。

!!! Warning * 悪意のあるユーザーからの潜在的な攻撃にデータベースを公開することができ、パブリックネットワークからデータベースへのアクセスを許可します。外部アクセスを許可する前にデータベースを保護するか、Kubernetesクラスターがプライベートネットワークからのみ到達可能であることを確認してください。

この例では、Kubernetesプロジェクトによって直接管理され、すべてのKubernetesクラスターに設定できるため、使用します。他の多くのコントローラーが利用可能です（包括的なリストを参照）。

次のことを想定しています。

- NGINX Ingressコントローラーが展開され、正常に動作する
- クラスターに `LoadBalancer` タイプのサービスを作成することが可能です

!!! Important * Ingressは、HTTPおよびHTTPSトラフィックを公開するためにのみ必要です。NGINX Ingressコントローラーは可能ですが、すべてのIngressオブジェクトが任意のポートまたはプロトコルを公開できるわけではありません。

最初のステップは、`tcp-services` `ConfigMap` を作成することです。このデータフィールドには、外部に公開されたポートと、内部的に指す名前空間、サービス、ポートに関する情報が含まれます。

```
apiVersion: v1
kind: ConfigMap
metadata:
  * name: tcp-services
  * namespace: ingress-nginx
data:
  * 5432: default/cluster-example-rw:5432
```

次に、ドキュメントで提案されているようにNGINX Ingress Controllerをインストールした場合、`ingress-nginx` サービスが必要で

す。5432ポートを `ingress-nginx` サービスに追加して公開する必要があります。イングレスは、ポート5432の着信接続をデータベースにリダイレクトします。

```
apiVersion: v1
kind: Service
metadata:
  * name: ingress-nginx
  * namespace: ingress-nginx
  * labels:
  *   app.kubernetes.io/name: ingress-nginx
  *   app.kubernetes.io/part-of: ingress-nginx
spec:
  * type: LoadBalancer
  * ports:
  *   - name: http
  *     port: 80
  *     targetPort: 80
  *     protocol: TCP
  *   - name: https
  *     port: 443
  *     targetPort: 443
  *     protocol: TCP
  *   - name: postgres
  *     port: 5432
  *     targetPort: 5432
  *     protocol: TCP
  * selector:
  *   app.kubernetes.io/name: ingress-nginx
  *   app.kubernetes.io/part-of: ingress-nginx
```

`cluster-expose-service.yaml` を使用し、`kubectl` を使用して適用できます。

!!! Warning * このファイルを直接適用すると、イングレスの `ConfigMap` および `Service` の以前の変更を上書きします

これで、Kubernetesクラスターの外部からPostgreSQLクラスターに到達できるようになります。

!!! Important * イングレスからの接続を許可するように `pg_hba` を設定してください。

Minikubeでのテスト

Minikubeでは、実行中のイングレスコントローラをセットアップできます。

```
minikube addons enable ingress
```

次に、`tcp-service` `ConfigMap`にパッチを適用して、イングレスのポート5432のプライマリtheconnectionsにリダイレクトします。

```
kubectl patch configmap tcp-services -n kube-system \
  * --patch '{"data":{"5432":"default/cluster-example-rw:5432"}}'
```

次に、デプロイメントにパッチ、ポート5432でのアクセスを許可します。次のコンテンツを含む `patch.yaml` というファイルを作成します。

```
spec:
  * template:
  *   spec:
  *     containers:
  *       - name: nginx-ingress-controller
  *         ports:
  *           - containerPort: 5432
  *             hostPort: 5432
```

`nginx-ingress-controller deployment` に適用します：

```
kubectl patch deployment nginx-ingress-controller --patch "$(cat patch.yaml)" -n kube-system
```

次を実行しているマシンからプライマリにアクセスできます。

```
psql -h $(minikube ip) -p 5432 -U postgres
```

Client SSL Connections

Cloud Native PostgreSQLは現在、認証局（CA）永久クラスターを作成しています。このCAは、証明書に署名してクライアントに提供し、クライアントとの安全な接続を作成するために使用されます。

SSLを使用してポッドに接続する

SSLを使用してクラスターに接続する

```
psql postgresql://cluster-example-rw:5432/app?sslmode=require
```

これにより、cluster `cluster-example` の `rw` サービスとの安全な接続が生成されます。

Kubernetes Upgrade

Kubernetesクラスターは常に更新する必要があります。Kubernetesクラスター、特にベアメタルを自己管理している場合、これはさらに重要になります。

定期的な更新の計画と実行は、メンテナンス上の理由でクラスターから一時的にノードを取り出すダウンタイムのKubernetesインフラストラクチャへの序文にもかかわらず、組織が技術的負債をクリーンアップし、ビジネスリスクを軽減する方法です（推奨読書：サイト信頼性エンジニアリングから）本）。

例、KubernetesがインストールされているLinuxservers上のセキュリティ更新を適用する必要があるかもしれませんが、あるいは、RAM、CPU、またはRAIDコントローラ、またはKubernetesの最新バージョンにしてもupgradetheクラスタとしてmalfunctioninghardwareコンポーネントを交換します。

通常、クラスター内のメンテナンス操作は、一度に1つのノードで実行されます。

1.更新するノードからワークロードを削除します（`drain`）2。実際のオペレーション（例、システム更新）の実行3。ノードをクラスターに再度参加させる（`uncordon`）

上記のプロセスでは、アップグレードの全期間にわたってワークロードを停止するか、別のノードに移行する必要があります。

最新のケースは、Kubernetesのサービスの信頼性と自己修復機能の観点から予想されるケースですが、一時的に劣化したクラスターで動作し、アップグレードされたノードが再び起動するのを待つことが推奨される場合があります。

具体的には、あなたのPostgreSQLのクラスタは、**ノードローカルストレージに依存している場合- PostgreSQL>は、*.nodeファイル・ローカル・ストレージ（または単にローカルストレージ）が実行されているwherethe Kubernetesワーカーノードに対してローカルである*ストレージがあることパフォーマンスを向上させるために使用されます。

!!! Note * データベースファイルがネットワーク上の共有ストレージにある場合、メンテナンスウィンドウを定義する必要はありません。ポッドで現在使用されているボリュームが、ドレイン後に異なるノードで実行されているポッドで再利用できる場合、演算子のデフォルトの自己修復動作は正常に機能します（このセクションの残りをスキップできます）。

PostgreSQLにローカルストレージを使用する場合は、`nodeMaintenanceWindow` オプションを使用してクラスターをメンテナンスモードで一時的に配置して、例、物理的ノードのパーティションを拡大したり、ノードを更新したりする標準的な自己修復手順を開始することをお勧めします自分自身。

!!! Warning * メンテナンスウィンドウの期間を可能な限り短い時間に制限します。このフェーズでは、Kubernetesの予想される動作の一部が無効になっているか、自己修復、ローリング更新、Pod中断予算などの制限付きで実行されてい更新。

クラスターの `nodeMaintenanceWindow` オプションには、さらに2つの設定があります。

`inProgress` : ノードのメンテナンスウィンドウが現在進行中かどうかを示すブール値。デフォルトでは、`off` に設定されています。メンテナンス期間中、以下の `reusePVC` オプションは演算子によって評価されます。

`reusePVC` : メンテナンスオペレーション中に既存のPVCを再利用するかどうかを定義するブール値。デフォルトでは、`on` に設定されています。有効の場合、Kubernetesはノードが再び起動するのを待ってから、既存のPVCを再利用します。

`PodDisruptionBudget` policyは一時的に削除されます。無効の場合、Kubernetesは、PostgreSQLの物理的ストリーミングレプリケーションに依存することにより、新しいPVCを使用して別のノードでthePodを強制的に再作成します。通常、このシナリオは、データベースのサイズが小さく、新しいPostgreSQLインスタンスの再クローン作成に待機するよりも時間がかかる場合を除き、お勧めできません。

!!! Note * `kubectl drain` コマンドを実行する場合、`--delete-local-data` オプションを追加する必要があります。恐れてはいけません: PostgreSQLデータディレクトリではなく、演算子によって内部的に使用される別のボリュームを参照します。

End-to-End Tests

Cloud Native PostgreSQL演算子は、一連の** End-to-end（E2E）テスト**を介して、各コミット後に自動的にテストされます。これにより、演算子がPostgreSQLクラスターを正しく展開および管理できます。

さらに、各コミットについて次のKubernetesバージョンがテストされ、開発プロセスの初期段階で障害とバグの検出が保証されます。

- 1.20
- 1.19
- 1.18
- 1.17
- 1.16

以下のPostgreSQLバージョンがテストされています。

- PostgreSQL 13
- PostgreSQL 12
- PostgreSQL 11
- PostgreSQL 10

KubernetesおよびPostgreSQLの各テストバージョンについて、[kind](#)を使用してKubernetesclusterが作成され、そのクラスターで次の一連のE2Eテストが実行されます。

- 演算子のインストール。
- Cluster の作成。
- データストレージ用の永続ボリュームの使用。
- 読み取り専用を含むサービスを介した接続。
- Cluster のスケールアップ。
- Cluster の縮小。
- フェイルオーバー。
- スイッチオーバー;
- PostgreSQL構成の変更を管理します。
- PostgreSQLイメージを変更する際のローリング更新。
- バックアップとScheduledBackupsの実行。
- 同期レプリケーション。
- バックアップからの復元。
- NodeSelector を使用したポッドアフィニティ。
- メトリックコレクション。
- フェイルオーバーが10秒より小さい場合のプライマリエンドポイントスイッチ。
- 20秒より小さいでスイッチオーバーが発生した場合のプライマリエンドポイントスイッチ。
- 60秒より小さいで劣化状態から回復します。

E2Eテストスイートは、次のサービスで作成されたクラスター上のOpenShift 4.6および最新のKubernetesおよびPostgreSQLリリースでも実行されます。

- Google GKE
- Amazon EKS
- Microsoft Azure AKS

Cloud Native PostgreSQL Plugin

Cloud Native PostgreSQLは、Kubernetesでクラスターを管理するための `kubect1` のプラグインを提供します。このプラグインは、OpenShift環境の `oc` でも機能します。

インストール

以下を使用して、システムにプラグインをインストールできます。

```
curl -sSfL \
  * https://github.com/EnterpriseDB/kubect1-cnp/raw/main/install.sh | \
  * sudo sh -s -- -b /usr/local/bin
```

使用

プラグインをインストールしてデプロイしたら、次のように使用をスタートできます。

```
kubect1 cnp <command> <args...>
```

ステータス

`status` コマンドは、クラスターの現在のステータスの概要を提供します。

```
kubectl cnp status cluster-example
```

```
Cluster in healthy state
Name:          cluster-example
Namespace:     default
PostgreSQL Image: quay.io/enterprisedb/postgresql:13
Primary instance: cluster-example-1
Instances:     3
Ready instances: 3

Instances status
Pod name      Current LSN  Received LSN  Replay LSN  System ID      Primary  Replicating  Replay paused
-----
cluster-example-1 0/6000060
cluster-example-2      0/6000060    0/6000060    6927251808674721812  x      ✓      x
cluster-example-3      0/6000060    0/6000060    6927251808674721812  x      ✓      x
```

`--verbose` または単に `-v` を追加することで、より詳細なバージョンのステータスを取得することもできます

```
kubectl cnp status cluster-example --verbose
```

```
Cluster in healthy state
Name:          cluster-example
Namespace:     default
PostgreSQL Image: quay.io/enterprisedb/postgresql:13
Primary instance: cluster-example-1
Instances:     3
Ready instances: 3

PostgreSQL Configuration
archive_command = '/controller/manager wal-archive %p'
archive_mode = 'on'
archive_timeout = '5min'
full_page_writes = 'on'
hot_standby = 'true'
listen_addresses = '*'
logging_collector = 'off'
max_parallel_workers = '32'
max_replication_slots = '32'
max_worker_processes = '32'
port = '5432'
ssl = 'on'
ssl_ca_file = '/tmp/ca.crt'
ssl_cert_file = '/tmp/server.crt'
ssl_key_file = '/tmp/server.key'
unix_socket_directories = '/var/run/postgresql'
wal_keep_size = '512MB'
wal_level = 'logical'
wal_log_hints = 'on'
* PostgreSQL HBA Rules
# Grant local access
local all all peer

# Require client certificate authentication for the streaming_replica user
hostssl postgres streaming_replica all cert clientcert=1
hostssl replication streaming_replica all cert clientcert=1

# Otherwise use md5 authentication
host all all all md5
* Instances status
Pod name      Current LSN  Received LSN  Replay LSN  System ID      Primary  Replicating  Replay paused
-----
cluster-example-1 0/6000060
cluster-example-2      0/6000060    0/6000060    6927251808674721812  x      ✓      x
cluster-example-3      0/6000060    0/6000060    6927251808674721812  x      ✓      x
```

このコマンドは、`yaml` および `json` フォーマットの出力もサポートしています。

プロモート

このコマンドの意味は、クラスター内のポッドをプライマリに `promote` することです。したがって、メンテナンス作業をスタートし

たり、クラスターのスイッチオーバーパシチューエーションをテストしたりできます。

```
kubectl cnp promote cluster-example cluster-example-2
```

証明書

Cloud Native PostgreSQL演算子を使用して作成されたクラスターは、CAと連携してTLS認証証明書に署名します。

証明書を取得するには、storethe資格情報、クラスタ名前への秘密の名前し、この証明書をユーザに提供する必要があります

```
kubectl cnp certificate cluster-cert --cnp-cluster cluster-example --cnp-user appuser
```

シークレットが作成されたら、 kubectl を使用して取得できます。

```
kubectl get secret cluster-cert
```

そして、次のコマンドを使用したプレーンテキストでの同じコンテンツ：

```
kubectl get secret cluster-cert -o json | jq -r '.data | map(@base64d) | .[]'
```

License and License Keys

演算子が作業するには、ライセンスキーが常に必要です。

唯一の例外は、コミュニティPostgreSQLで演算子を実行する場合です。この場合、ライセンスキーが設定されていない場合、クラスターはdefaulttrialライセンスで開始されます。これは30日後に自動的に期限切れになります。

!!! Important * ライセンスの有効期限が切れると、演算子はクラスターでの調整試行を停止し、そのステータスを管理するために事実上停止します。ポッドとデータは引き続き利用可能です。

会社レベルのライセンスキー

ライセンスキーを使用すると、インストール環境でPostgreSQLclusterを無制限に作成できます。

ライセンスキーは、演算子が展開されているのと同じネームスペースの ConfigMap で利用可能であるニーズがあります。

Kubernetesでは、演算子はデフォルトで postgresql-operator-system 名前空間にデブロイされます。代わりにOLMが使用されると（つまりOpenShift上で）、演算子はデフォルトで openshift-operators 名前空間にインストールされます。

名前空間名前とライセンスキーを指定すると、次のコマンドで構成マップできます。

```
kubectl create configmap -n [NAMESPACE_NAME_HERE] \
* postgresql-operator-controller-manager-config \
* --from-literal=EDB_LICENSE_KEY=[LICENSE_KEY_HERE]
```

次のコマンドを使用して、構成マップをリロードできます。

```
kubectl rollout restart deployment -n [NAMESPACE_NAME_HERE] \
* postgresql-operator-controller-manager
```

ライセンスキーの有効性は、クラスタステータス内で確認できます。

```
kubectl get cluster cluster_example -o yaml
[...]
status:
  * * [...]
  * * licenseStatus:
  * *   licenseExpiration: "2021-11-06T09:36:02Z"
  * *   licenseStatus: Trial
  * *   valid: true
  * *   isImplicit: false
  * *   isTrial: true
[...]
```

クラスターレベルのライセンスキー

各 Cluster リソースの定義には `licenseKey` パラメータがあります。クラスターステータスで、有効期限とライセンスに関する詳細情報を確認できます。

```
kubectl get cluster cluster_example -o yaml
[...]  
status:  
  * * [...]
  * * licenseStatus:  
  * *   licenseExpiration: "2021-11-06T09:36:02Z"  
  * *   licenseStatus: Trial  
  * *   valid: true  
  * *   isImplicit: false  
  * *   isTrial: true  
[...]
```

有効期限を延長したり、クラスターを稼働ライセンスに移動したりするために、クラスターライセンスキーはいつでも新しいものに更新できます。

Cloud Native PostgreSQLは、enterprisedb.com/limited-use-licenseで入手可能なEnterpriseDB Limited Usage LicenseAgreementの下で配布されます。

クラウドネイティブPostgreSQL： Copyright（C）2019-2021 EnterpriseDB。

Container Image Requirements

KubernetesのCloud Native PostgreSQL演算子は、次の要件に準拠する、PostgreSQLの互換性のあるコンテナイメージで動作するように設計されています。

- バスに存在する必要があるPostgreSQL 10以降の実行可能ファイル：
 - `initdb`
 - `postgres`
 - `pg_ctl`
 - `pg_controldata`
 - `pg_basebackup`
- バスに含まれている必要があるBarman Cloud実行可能ファイル：
 - `barman-cloud-wal-archive`
 - `barman-cloud-wal-restore`
 - `barman-cloud-backup`
 - `barman-cloud-restore`
 - `barman-cloud-backup-list`
- 賢明なロケール設定

CloudNative PostgreSQLはインスタンスマネージャでそれをオーバーライドするため、イメージ定義にエントリーポイントやコマンドは必要ありません。

!!! Warning * アプリケーションコンテナイメージは、マルチプル/オプションのホットスタンバイサーバーアーキテクチャを備えたプライマリでのみCloud Native PostgreSQLによって使用されます。

EnterpriseDBは、Cloud NativePostgreSQLのパブリックコンテナイメージを提供およびサポートし、[Quay.io](https://quay.io)で公開します。

イメージタグの要件

イメージ名前はDockerで有効なものであれば何でもかまいませんが、Cloud NativePostgreSQL演算子は* イメージタグ*に依存して、イメージによって実行されるPostgres `majorversion`を検出します。

イメージ>は、任意のドットとパッチ>が続く有効なPostgreSQLのメジャーバージョン（例えば9.6or 12）でスタートしなければなりません。

プレフィックスの後には、Dockerタグで有効で受け入れられる有効な文字の組み合わせを、ドット、アンダースコア、またはマイナス符号を前にタグ続けることができます。

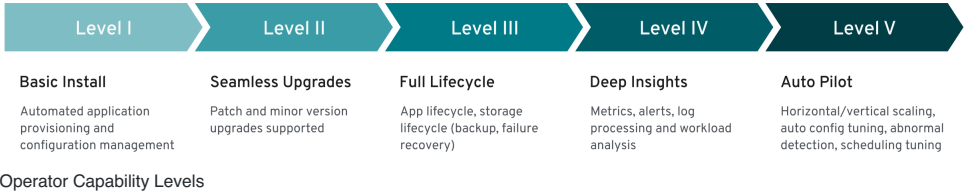
受け入れられるイメージタグの例：

- `9.6.19-alpine`
- `12.4`
- `11_1`
- `13`
- `12.3.2.1-1`

!!! Warning * `latest` は、イメージの有効なタグとは見なされません。

Operator Capability Levels

このセクションでは、“[Operator SDK definition of Capability Levels](#)”frameworkを使用して分類されたCloud Native PostgreSQLによって実装される機能の概要を説明します。



各機能レベルは、演算子が提供する管理機能の特定のセットに関連付けられています。

1.基本インストール2. シームレスなアップグレード3. フルライフサイクル4. 深い洞察5. オートパイロット

!!! Note * このフレームワークは、演算子の将来の作業と実装のガイドと考えています。

レベル1-基本インストール

機能レベル1には、オペレーターのインストールおよび構成が含まれます。このカテゴリには、ユーザーがオペレータやPostgreSQLクラスター設定と対話する方法の改善など、使いやすさとユーザーエクスペリエンスの向上が含まれます。

!!! Important * このレベルの情報セキュリティパートを検討します。

宣言的な設定によるオペレーターの展開

演算子は、3つの `CustomResourceDefinition` オブジェクト、`Cluster`、`Backup`、`ScheduledBackup` を定義するKubernetesマニフェストを使用して宣言的にインストールされます。

宣言的な設定によるPostgreSQLクラスターの展開

PostgreSQLクラスター（オペランド）は、完全に宣言的な方法で `Cluster` カスタムリソースを使用して定義されます。PostgreSQLのバージョンは、CRで定義されているオペランドコンテナイメージによって決定され、要求されたレジストリ自動的に取得されます。オペランドをデプロイするとき、演算子は次のリソース自動的に作成します：`Pod`、`Service`、`Secret`、`ConfigMap`、`PersistentVolumeClaim`、`PodDisruptionBudget`、`ServiceAccount`、`RoleBinding`、`Role`。

CRDを介したオペランドイメージのオーバーライド

演算子は、任意のオペランドイメージwithPostgreSQL inside.Byのデフォルトをサポートするように設計され、演算子>の互換性のある任意のイメージを使用することができますPostgreSQLCommunityによってサポートされており、EnterpriseDBの。あなたによってQuay.ioに公開される最新の安定（stable）メジャーバージョンの最新の利用可能MINORVERSIONを使用していますCRで `imageName` 属性を設定して、プライマリ/スタンバイアーキテクチャを直接サポートします。演算子は、プライベートコンテナレジストリにアクセスするための `imagePullSecretsNames` もサポートしています。

自己完結型のインスタンスマネージャ

PatroniやStolonなどの外部ツールに依存してKubernetesクラスターポッド内のPostgreSQLインスタンスを調整する代わりに、オペレーターは各ポッド内で、`/controller/manager` という名前のファイルで演算子実行可能ファイルを注入します。アプリケーションはunderlyingPostgreSQLインスタンスを制御し、PostgreSQLのクラスタポッドにitselfbasedインスタンスとポッドの状態を調整するために使用されます。インスタンスマネージャは、プロープ用に `kubelet` によって呼び出されるWebサーバーも起動します。 `kubelet` によって呼び出されたUnixシグナルは、インスタンスマネージャによってフィルター処理され、必要に応じて、外部イベントに対する高速で制御された反応のために `postgres` プロセスに転送されます。インスタンスマネージャはGoで記述されており、外部依存関係はありません。

ストレージ構成

ストレージは、データベースワークロードの重要なコンポーネントです。ストレージに関して、Kubernetesのネイティブ機能とリソースを活用することで、オペレーターは、基盤となるKubernetes環境が提供できるものに基づいて、ワークロード要件に適したストレージを選択するのに十分な柔軟性をユーザーに提供します。これは、パブリッククラウド環境で特定のストレージクラスを選択するか、CRの `storage` パラメータでPVCテンプレートを介して生成されたPVCを微調整することを意味します。

レプリカ構成

演算子は、`instances` という単一のパラメータ、クラスター内のレプリカを自動的に検出します。 `1` に設定すると、クラスターはレプリカなしの単一のプライマリPostgreSQLインスタンスで構成されます。 `1` よりも高い場合、演算子は、自動フェイルオーバーによ

る高可用性やスイッチオーバー操作によるローリング更新など、 `instances -1` レプリカを管理し更新。

データベース設定

演算子は、単一のデータベースでPostgreSQLクラスターを管理するように設計されています。演算子は、透過的に同じ名前を持つ通常のPostgresのユーザが所有し、デフォルトで自動的にプロビジョニングおよび読み取り専用writeand読み取り専用のコンフィギュレーション・アプローチの上に規則をworkloads.Using、演算子はadatabaseと呼ばれる `app` を作成するために管理Kubernetesサービスthroughtwoデータベースへのアクセスを管理します。必要に応じて、データベース名とユーザ名前の両方を指定できます。クラスターの実行に構成は必要ありませんが、ユーザーは、CRの `postgresql` セクションでPostgreSQL実行時構成とPostgreSQL Host-BasedAuthenticationルールの両方をカスタマイズできます。

ポッドセキュリティポリシー

InfoSec要件の場合、演算子は、コンテナの実行と、演算子とオペランドの両方のボリュームへのアクセスに特権モードを必要としません。

ライセンスキー

演算子は、キーがない場合のデフォルトの動作をプログラムで定義できるライセンスキーをサポートしていいキー。CloudNative PostgreSQLは、デプロイされたクラスターごとに暗黙の30日間トライアルライセンスを作成するようにプログラムされています。ライセンスキーは署名された文字列で、演算子は非対称キー技術を使用して検証できキー。コンテンツは、製品、クラスター識別子（名前空間と名前）、インスタンスの数、有効期限、および必要に応じて、保護されたイメージからイメージをプルダウンするために演算子が秘密として使用する資格情報を含むJSONオブジェクトですcontainerregistry。有効期限を過ぎると、演算子はライセンスキーが復元されるまで調整プロセスを停止します。

クラスターの現在のステータス

演算子は、CRの状態セクションをクラスターの監視状態で継続的に更新します。 PostgreSQLクラスターステータス全体は、各ポッドで実行されているインスタンスマネージャによって継続的に監視されます：インスタンスマネージャは、必要な変更を制御されたPostgreSQLインスタンスに適用して、クラスターの必要なステータスに収束する責任があります（例、クラスターステータスがポッド `-1` であると報告した場合プライマリ、ポッド `-1` は自分自身をプロモートニーズが、他のポッドはポッド `-1` をフォローする必要があります。同じステータスは、OpenShiftダッシュボードなどの詳細を提供するためにKubernetesクライアントアプリケーションによって使用されます。

オペレーターの認証権限

演算子は、自動的.ITを作成し、演算子の権限との兆候リーフがtheKubernetes APIサーバと演算子自分自身の間で安全な通信を確保する保証に、ウェブフックサーバによって使用されるcertificateto自分自身のために証明権限を作成します。

クラスターの証明権限

演算子は、自動的に【選択演算子は、すべてのcluster certification権限を署名する証明機関を使用する問題に使用し、ストリーミングレプリケーションスタンバイサーバおよびアプリケーション（代わりのパスワード）をストリーミングauthenticationof用のTLS証明書を更新されているすべてのPostgreSQLcluster、のための権限を作成します。

TLS接続

演算子は、TLS / SSL接続を透過的かつネイティブにサポートして、クラスターの証明権限を使用してセキュリティを強化するためにクライアント/サーバ通信を暗号化します。

ストリーミングレプリケーションの証明書認証

演算子は、パスワード（およびそのための秘密）に頼るのではなく、TLSクライアント証明書認証に依存して、スタンバイサーバからのストリーミングレプリケーション接続を承認します。

継続的な構成管理

演算子は、ユーザーがPostgreSQL構成の `cluster` リソースYAMLセクションに変更を適用できるようにし、構成オプションに応じてすべてのインスタンスが適切にリロードまたは再起動されるようにします。*現在の制限*： `ALTER SYSTEM` による変更は検出されません。実施されない； `max_connections` や `max_wal_senders` など、 `hot standby sensitive parameters`では適切なリスタートオーダーが実装されていません。

複数のインストール方法

演算子は `kubectlapply` を介してKubernetesマニフェストからインストール、パブリックおよびプライベートクラウド環境での従来のKubernetesインストールで使用できます。さらに、OperatorHubを介してOpenShiftContainerプラットフォームにデプロイできます。

設定より規約

演算子は、構成パラダイムに関する規則をサポートし、デフォルト値を決定すると同時に、ユーザーがそれらをオーバーライドしてカスタマイズできるようにします。いくつかのYAMLコード行で `cluster` CRDを使用して、 PostgreSQLクラスターのデプロイメントを

指定できます。

レベル2-シームレスなアップグレード

機能レベル2とは、演算子と実際のワークロード（この場合はPostgreSQLサーバー）の更新を有効にすることです。これには、** PostgreSQLマイナーリリース更新（通常はセキュリティおよびバグ修正） およびメジャーオンラインアップグレード**が含まれます。

演算子のアップグレード

新しい展開として演算子をシームレスにアップグレードできます。演算子の変更は、オペランドの変更を必要としません-`instancemanager`のインジェクションのおかげです。演算子は、古いバージョンのオペランドを管理できます。

管理対象ワークロードのアップグレード

オペランドは、CR、特に `imageName` パラメータの変更の一環として、宣言的な構成アプローチを使用してアップグレードできます。オペレーターは、PostgreSQLのメジャーアップグレードを防止し、メジャーバージョン内のマイナーPostgreSQLリリースの両方の方向に進むことを可能にします（更新とロールバックを有効にし更新）。

スタンバイサーバーが存在する場合、演算子は既存のポッドをドロップし、基になるストレージを再利用する新しい要求されたオペランドイメージで新しいポッドを作成することで、レプリカからローリング更新を実行します。`primaryUpdateStrategy` の値に応じて、演算子はスイッチオーバーを続行します以前のプライマリ（`unsupervised`）を更新するか、ユーザがスイッチオーバープロシージャ（`supervised`）を手動で発行するのを待ちます。実際のデータベースに基づいて、オペレーションがアプリケーションのダウンタイムを数秒から生成する可能性があるため、使用する設定はビジネス要件に依存しますワークロード。

アップグレード中にクラスターの可用性ステータスを表示する

いつでも、例 `OK`、`Failover in progress`、`Switchover in progress`、`Upgrade in progress`、`or Upgrade failed` など、クラスターの高可用性ステータスを伝えます。

レベル3-完全なライフサイクル

機能レベル3では、演算子がビジネス継続性およびスケーラビリティの側面を管理する必要があります。災害リカバリは、データベースのバックアップとリカバリの両方が正しく機能することを必要とするビジネス継続性コンポーネントです。出発点としての目標はRPO <5分を達成することですが、長期的な目標はRPO = 0バックアップソリューションを実装することです。高可用性は、PostgreSQLのネイティブ物理レプリケーションとホットスタンバイレプリカを介して、演算子がフェールオーバーおよびスイッチオーバー操作を実行できるようにするビジネス継続性のもう1つの重要なコンポーネントです。この領域には、次の機能強化が含まれます。

- 同期レプリケーションなどのPostgreSQL物理的レプリケーションの制御、（カスケード）レプリケーションクラスターなど。
- コネクションプーリング、を介してパフォーマンスと制御を改善します `pgBouncer`によるコネクションプーリングレイヤー。

PostgreSQLバックアップ

演算子は、物理ベースバックアップと連続WALアーカイピングに基づくPostgreSQLのネイティブ連続バックアップテクノロジーを使用して、アプリケーションレベルのバックアップを提供するように設計されています。具体的には、演算子は現在、AWS S3またはS3互換のオブジェクトストアおよびMinIOなどのゲートウェイでのバックアップのみをサポートしています。

WALアーカイピングとベースバックアップは、S3プロトコルの宛先URL（例、AWS S3バケット内の特定のフォルダーを指す）を指定することにより、クラスター定義の `backup` パラメータを介してクラスターレベルで宣言定義され、オプションで汎用エンドポイントURL。連続バックアップの前提条件であるWALアーカイピングは、ユーザさらなるアクションを必要としません。演算子は、`archive _command` を自動的かつ透過的に設定し、`barman-cloud-wal-archive` に依存して、定義されたエンドポイントにWALfilesを送信します。ユーザーは圧縮アルゴリズムを決定できます。

ベースバックアップは、オンデマンド（`Backup custom`リソース定義）またはスケジュール（`cron`のような構文を使用して `Scheduled Backup customer`リソース定義を使用）の2つの方法で定義できます。これらは両方とも、ジョブの `on barman-cloud-backup`（`applicationcontainer` イメージのパートとして配布）に依存して、WALファイルと同じエンドポイントでバックアップをリレーします。

`barman-cloud-wal-restore` と `barman-cloud-backup` の両方は、GNU GPL 3の条件の下でアプリケーションコンテナイメージに配布されます。

バックアップからの完全な復元

この演算子により、ユーザーは、`barman-cloud-backup` を使用して取得された既存のアクセス可能なバックアップから開始して、新しいクラスター（およびその設定）をブートストラップできます。ブートストラップが完了すると、`operatorinitiates` リカバリモードとリブレイでは、インスタンス使用可能なすべてのWALのfilesfrom指定されたアーカイブ、リカバリを終了し、.Subsequentlyプライマリとして開始するには、演算子は、プライマリスタンバイinstancesfromの要求された数のクローンを作成します。

バックアップからのポイントインタイムリカバリ（PITR）

この演算子により、ユーザーは、タイムスタンプ、ラベルまたはトランザクションIDで定義された特定の時点に既存のバックアップを回復することにより、新しいPostgreSQLクラスターを作成できます。この機能は完全な`restoreone`の上に構築されており、PostgreSQL

for [PITR](#)で使用可能なすべてのオプションをサポートしています。

同期レプリケーションによるゼロデータ損失クラスター

クォラムベースの同期レプリケーションサポートにより、ローカルの高可用性クラウドネイティブPostgreSQLclusterでゼロデータ損失（RPO = 0）を達成します。演算子は、いつでも利用可能な予想される同期的スタンバイレプリカの最小数と最大数を制御する2つの構成オプションを提供します。演算子は、クラスター内の使用可能な準備が整ったPostgreSQLインスタンスの数に基づいて、次の式を使用してそれに応じて反応します。

```
0 <= minSyncReplicas <= maxSyncReplicas < instances
```

活性と準備のプロープ

演算子は、PostgresContainersの活性プロープと準備プロープを定義します。これらのプロープは、kubectletによって呼び出されます。これらは、インスタンスマネージャによって直接管理されるウェブサーバの `/healthz` および `/readyz` エンドポイントにそれぞれマッピングされます。どちらもGoを使用してクラスターに接続し、シンプルクエリー（`;`）を発行して、サーバが接続を受け入れる準備ができていることを確認します。

ローリング展開

演算子はローリング展開をサポートしてダウンタイムを最小限に抑え、PostgreSQLクラスターが公開されている場合、サービスは初期化または更新中に利用可能なポッドのみに読み取り専用トラフィックの負荷を分散します。

レプリカのスケールアップとスケールダウン

この演算子を使用すると、ユーザーはPostgreSQLクラスター内のインスタンスの数を増減できます。新しいレプリカはプライマリサーバから自動的に起動され、クラスターのHAインフラストラクチャに参加します。CRDは、ユーザが `kubectlet scale` コマンドを使用できるようにする「スケール」サブリソースを宣言します。

KubernetesノードのメンテナンスウィンドウとPodDisruptionBudget

演算子は `PodDisruptionBudget` リソースを作成して、同時中断の数を1つに制限します。この構成により、メンテナンス操作がクラスター内のすべてのポッドを削除することを防ぎ、指定された数のインスタンスを作成できます。`PodDisruptionBudget`は、ノードのドレインオペレーション中に適用され、クラスターサービスの中断を防ぎます。

この戦略は、ストレージがすべてのワーカーノード間で共有されるKubernetesクラスターには適切ですが、ローカルストレージを使用するクラスターやプライベートクラウドにインストールされたクラスターには最適なソリューションではない場合があります。演算子は、ユーザーがメンテナンスウィンドウを指定し、基になるノードの立ち退きに対する反応を設定できるようにします。メンテナンスウィンドウセクションの `ReusePVC` オプションを使用すると、使用する戦略を指定できます。削除されたインスタンスに別のPVCに新しいストレージを割り当てるか、基になるノードが再び使用可能になるまで待機します。

ポッドでの永続ボリュームストレージの再利用

演算子のニーズがKubernetesのメンテナンスオペレーションによって追い出されユーザorhasによって削除されたポッドを作成するとき利用可能な場合、それはneedto再クローンをプライマリからのデータを避け、the `PersistentVolumeClaim` を再利用します。

CPUとメモリのリクエストと制限

演算子により、管理者は、マニフェストの `resources` セクションを使用して、クラスターのポッドによるリソース使用を制御および管理できます。特定の `requests` と `limits` の値は、CPUとRAMの両方に設定できます。

レベル4-深い洞察

機能レベル4は、監視可能性：特に、モニタリング、アラート、トレンド、ログ処理です。これには、Prometheus、Grafana、Fluent Bitなどの外部ツールの使用、およびJSONフォーマットでエラーログを直接出力するためのPostgreSQLエンジンの拡張が含まれる場合があります。

Prometheusエクスポーターインフラストラクチャ

インスタンスマネージャはプラグラブルフレームワークを提供し、独自のWebサーバを介して、Prometheusモニタリングおよびアラートツールのメトリックをエクスポートするエンドポイントを公開します。現在、PostgreSQLの基本的なメトリックと `pg_stat_archiver` システムビューのみが実装されています。

Kubernetesイベント

リソースの作成、ノードの削除、アップグレード処理など、Kubernetes APIが期待するメジャーイベントを記録します。イベントは、`kubectlet describe` および `kubectlet get events` コマンドを使用して表示できます。

レベル5-オートパイロット

能力レベル5は、自動化スケーリング、修復、チューニングに焦点を当てています-可観測性レイヤー出現した異常と洞察の発見を通じて。

自己修復のための自動フェイルオーバー

プライマリで障害が検出された場合、演算子は、最も整合したレプリカを新しいターゲットプライマリとして設定することにより、クラスタのステータスを変更します。その結果、それぞれの生きているポッド内のインスタンス>が必要な手順は、いずれかの新しいプライマリになるか、または元のプライマリが復帰it.Inケースに従うことによって、要求された状態oftheクラスタに自分自身を整列するwillinitiate、同じメカニズムが回避されますアプリケーションからのアクセスを防ぎ、サーバー上で pg_rewind を実行し、スタンバイとして再起動することにより、asplit-brain。

スタンバイの自動作成

スタンバイをホスティングしているポッドが取り外された場合、演算子はスタンバイサーバを再作成するプロシージャを開始します。

API Reference

Cloud Native PostgreSQLは、次のカスタムリソースを定義するKubernetes APIを拡張します。

- [Backup](#)
- [Cluster](#)
- [ScheduledBackup](#)

すべてのリソースは postgresql.k8s.enterprisedb.io/v1 APIで定義されています。

使用例については、ドキュメントの“[Configuration Samples](#)” page "を参照してください。

以下に、定義されたリソースの説明があります。

- [Backup](#)
- [BackupList](#)
- [BackupSpec](#)
- [BackupStatus](#)
- [AffinityConfiguration](#)
- [BackupConfiguration](#)
- [BarmanObjectStoreConfiguration](#)
- [BootstrapConfiguration](#)
- [BootstrapInitDB](#)
- [BootstrapRecovery](#)
- [Cluster](#)
- [ClusterList](#)
- [ClusterSpec](#)
- [ClusterStatus](#)
- [DataBackupConfiguration](#)
- [MonitoringConfiguration](#)
- [NodeMaintenanceWindow](#)
- [PostgresConfiguration](#)
- [RecoveryTarget](#)
- [RollingUpdateStatus](#)
- [S3Credentials](#)
- [StorageConfiguration](#)
- [WalBackupConfiguration](#)
- [ScheduledBackup](#)
- [ScheduledBackupList](#)
- [ScheduledBackupSpec](#)
- [ScheduledBackupStatus](#)

バックアップ

バックアップは、バックアップAPIのスキーマです

Field	Description	Scheme	Required
metadata	NA	metav1.ObjectMeta	false

Field	Description	Scheme	Required
spec	" バックアップの望ましい動作の仕様。詳細： https : /git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status"	BackupSpec	false
status	" バックアップの最新の状態。このデータは最新ではない可能性があります。システムによって設定されます。読み取り専用。詳細： https : /git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status"	BackupStatus	false

BackupList

BackupListにはバックアップのリストが含まれます

Field	Description	Scheme	Required
metadata	" 標準リストメタデータ。詳細： https : /git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds"	metav1.ListMeta	false
items	バックアップのリスト	[]Backup	true

BackupSpec

BackupSpecは、バックアップの望ましい状態を定義します

Field	Description	Scheme	Required
cluster	バックアップクラスター	v1.LocalObjectReference	false

BackupStatus

BackupStatusは、観測されたバックアップの状態を定義します

Field	Description	Scheme	Required
s3Credentials	S3にデータをアップロードするために使用する資格情報	S3Credentials	true
endpointURL	データをクラウドにアップロードするために使用されるエンドポイント。自動エンドポイント検出をオーバーライドします	string	false
destinationPath	バックアップを保存するパス（s3 : // bucket / パス / to / folder）このパスは、異なる宛先フォルダーで、WALとデータに使用されます	string	true
serverName	S3上のサーバー名前。このパラメータが省略された場合、クラスター名前が使用されます。	string	false
encryption	S3 APIに必要な暗号化メソッド	string	false
backupId	BarmanバックアップのID	string	false

Field	Description	Scheme	Required
phase	最後のバックアップステータス	BackupPhase	false
startedAt	バックアップが開始されたとき	*metav1.Time	false
stoppedAt	バックアップが終了したとき	*metav1.Time	false
error	検出されたエラー	string	false
commandOutput	バックアップコマンドの出力	string	false
commandError	バックアップコマンドの出力	string	false

AffinityConfiguration

AffinityConfigurationには、ポッドのアフィニティルールを作成するために必要な情報が含まれています

Field	Description	Scheme	Required
enablePodAntiAffinity	ポッドの非アフィニティをアクティブにします。このフィールドが明示的にfalseに設定されていない限り、演算子はポッドの非アフィニティを定義します	*bool	false
topologyKey	非アフィニティ構成に使用するTopologyKey。詳細については、k8sの文書を参照してください	string	true
nodeSelector	" NodeSelectorは、ポッドを実行できるノードを定義するために使用されるキーと値のペアのマップです。詳細： https : https://kubernetes.io/docs/concepts/configuration/assign-pod-node/"	map[string]string	false

BackupConfiguration

BackupConfigurationは、クラスターのバックアップの取得方法を定義します。現在、サポートされている唯一のバックアップメソッドはbarmanObjectStoreです。詳細と例については、文書の「バックアップ リカバリー」 セクションを参照してください

Field	Description	Scheme	Required
barmanObjectStore	barman-cloudツールの構成	*BarmanObjectStoreConfiguration	false

BarmanObjectStoreConfiguration

BarmanObjectStoreConfigurationには、S3互換オブジェクトストレージに対してBarmanを使用したバックアップ構成が含まれています

Field	Description	Scheme	Required
s3Credentials	S3にデータをアップロードするために使用する資格情報	S3Credentials	true
endpointURL	データをクラウドにアップロードするために使用されるエンドポイント。自動エンドポイント検出をオーバーライドします	string	false
destinationPath	バックアップを保存するパス（s3 : // bucket / パス / to / folder）このパスは、異なる宛先フォルダーで、WALとデータに使用されます	string	true
serverName	S3上のサーバー名前。このパラメータが省略された場合、クラスター名前が使用されます。	string	false
wal	WALストリームのバックアップの構成。定義されていない場合、WALファイルは圧縮されずに保存され、パケットのデフォルトポリシーに従って、オブジェクトストアで暗号化されていない場合があります。	*WalBackupConfiguration	false

Field	Description	Scheme	Required
data	データファイルのバックアップに使用される構成定義されていない場合、ペースバックアップファイルは圧縮されずに格納され、バケットのデフォルトポリシーに従って、オブジェクトストアに暗号化されない場合があります。	* DataBackupConfiguration	false

BootstrapConfiguration

BootstrapConfigurationには、 PostgreSQL クラスターの作成方法に関する情報が含まれています。サポートされているメソッドの中では、1つのブートストラップ方法のみを定義できます。 `initdb` は、指定されないままの場合、ブートストラップメソッドとして使用されます。詳細については、文書のブートストラップページを参照してください。

Field	Description	Scheme	Required
initdb	initdb経由でクラスターをブートストラップします	* BootstrapInitDB	false
recovery	バックアップからクラスターをブートストラップします	* BootstrapRecovery	false

BootstrapInitDB

BootstrapInitDBは、initdbを使用する場合のブートストラッププロセスの構成です。詳細については、文書のブートストラップページを参照してください。

Field	Description	Scheme	Required
database	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： <code>app</code> 。	string	true
owner	アプリケーションで使用するインスタンスデータベースの所有者の名前。デフォルトは <code>database</code> キーの値です。	string	true
secret	ユーザデータベースの所有者の初期資格情報を含むシークレットの名前。空の場合、新しいシークレットが最初から作成されます	*corev1.LocalObjectReference	false
redwood	Redwood互換性を有効/無効にする必要がある場合。EPASが必要で、EPASのデフォルトは真	*bool	false
options	クラスターの作成時にinitdbに渡す必要があるオプションのリスト	[]string	false

BootstrapRecovery

BootstrapRecoveryには、指定された名前でバックアップを復元するために必要な構成が含まれており、スーパーユーザ用に選択したパスワードを変更した後、復元されたプライマリからすべてのインスタンスを複製する完全なクラスターをブートストラップするためにそれを使用します。詳細については、文書のブートストラップページを参照してください。

Field	Description	Scheme	Required
backup	復元する必要があるバックアップ	corev1.LocalObjectReference	true
recoveryTarget	デフォルトでは、一貫した状態に達するとすぐにリカバリが終了します。この場合、バックアップの終了を意味します。このオプションにより、リカバリプロセスを微調整できます。	* RecoveryTarget	false

クラスター

ClusterはPostgreSQL APIのスキーマです

Field	Description	Scheme	Required
metadata	NA	metav1.ObjectMeta	false
spec	クラスターの望ましい動作の仕様。詳細： https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status	ClusterSpec	false

Field	Description	Scheme	Required
status	" クラスターの最近観察されたステータス。このデータは最新ではない可能性があります。システムによって設定されます。読み取り専用。詳細： https : https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status"	ClusterStatus	false

ClusterList

ClusterListにはクラスターのリストが含まれます

Field	Description	Scheme	Required
metadata	" 標準リストメタデータ。詳細： https : https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds"	metav1.ListMeta	false
items	クラスターのリスト	[]Cluster	true

ClusterSpec

ClusterSpecは、クラスターの望ましい状態を定義します

Field	Description	Scheme	Required
description	このPostgreSQLクラスターの説明	string	false
imageName	コンテナイメージの名前	string	false
postgresUID	イメージ内の postgres ユーザのUID、デフォルトは 26	int64	false
postgresGID	イメージ内の postgres ユーザのGID、デフォルトは 26	int64	false
instances	クラスターに必要なインスタンスの数	int32	true
minSyncReplicas	プライマリとの同期レプリケーションに必要なインスタンスの最小数。未定義または0の場合、スタンバイが利用できないときに書き込みを完了できます。	int32	false
maxSyncReplicas	同期レプリケーションクォーラムの目標値。準備完了スタンバイの数がこれより少ない場合は、減らすことができます。未定義または0は、同期レプリケーションを無効にします。	int32	false
postgresql	PostgreSQL サーバーの構成	PostgresConfiguration	false
bootstrap	このクラスターをブートストラップ手順	*BootstrapConfiguration	false
superuserSecret	スーパーユーザのパスワードを含むシークレット。定義されていない場合、ランダムに生成されたパスワードで新しいシークレットが作成されます	*corev1.LocalObjectReference	false
imagePullSecrets	画像のプルに使用されるプルシークレットのリスト。ライセンスキーにプルシークレットが含まれている場合、そのシークレットは自動的に含まれます。	[]corev1.LocalObjectReference	false
storage	インスタンスのストレージの構成	StorageConfiguration	false
startDelay	PostgreSQLインスタンスが正常にスタートするために許可される秒単位の時間（デフォルト30）	int32	false

Field	Description	Scheme	Required
stopDelay	PostgreSQLインスタンスノードが正常にシャットダウンできる時間（秒）（デフォルトは30）	int32	false
affinity	ポッドのアフィニティ/アンチアフィニティルール	AffinityConfiguration	false
resources	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。	corev1.ResourceRequirements	false
primaryUpdateStrategy	すべてのレプリカが正常に更新された後、ローリング更新プロセス中にプライマリサーバーをアップグレードするための戦略：自動化（unsupervised -デフォルト）またはマニュアル（supervised）が可能	PrimaryUpdateStrategy	false
backup	バックアップに使用される構成	* BackupConfiguration	false
nodeMaintenanceWindow	Kubernetesノードのメンテナンスウィンドウを定義する	* NodeMaintenanceWindow	false
licenseKey	クラスターのライセンスキー。空の場合、クラスターは試用モードで動作し、有効期限（デフォルトは30日）が経過すると、演算子は調整の試みを中止します。詳細については、演算子に付属のライセンス契約を参照してください。	string	false
monitoring	このクラスターのモニタリングインフラストラクチャの構成	* MonitoringConfiguration	false

ClusterStatus

ClusterStatusは、クラスターの観測状態を定義します

Field	Description	Scheme	Required
instances	クラスター内のインスタンスの総数	int32	false
readyInstances	クラスター内の準備完了インスタンスの総数	int32	false
instancesStatus	インスタンスのステータス	map[utils.PodStatus][]string	false
latestGeneratedNode	生成された最新のノードのID（ノード名前の衝突を避けるために使用）	int32	false
currentPrimary	現在のプライマリインスタンス	string	false
targetPrimary	ターゲットプライマリインスタンス。これは、スイッチオーバーまたはフェイルオーバー中の以前のインスタンスとは異なります。	string	false
pvcCount	このクラスターによって作成されたPVCの数	int32	false
jobCount	このクラスターによって作成されたジョブの数	int32	false
danglingPVC	このクラスターによって作成され、ポッドに接続されていないまだ使用可能なすべてのPVCのリスト	[]string	false
initializingPVC	このクラスターによって初期化されているすべてのPVCのリスト	[]string	false
licenseStatus	ライセンスの状態	licensekey.Status	false
writeService	現在の書き込みポッド	string	false
readService	読み取りポッドの現在のリスト	string	false
phase	クラスターの現在のフェーズ	string	false
phaseReason	現在のフェーズの理由	string	false

DataBackupConfiguration

DataBackupConfigurationは、データディレクトリのバックアップの構成です。

Field	Description	Scheme	Required
-------	-------------	--------	----------

Field	Description	Scheme	Required
compression	オブジェクトストアにストリーミングしながら、バックアップファイル（テーブルスペースごとのtarファイル）を圧縮します。使用可能なオプションは、空の文字列（圧縮なし、デフォルト）、 <code>gzip</code> または <code>bzip2</code> です。	CompressionType	false
encryption	ファイルの暗号化を強制するとき（バケットがそのためにまだ構成されていない場合）。許可されるオプションは、空の文字列（バケットポリシー、デフォルト）、 <code>AES256</code> および <code>aws:kms</code> です。	EncryptionType	false
immediateCheckpoint	PostgreSQLサーバーの <code>checkpoint_completion_target</code> 設定に従って、バックアップ初期チェックポイントのI/Oワークロードを制限するかどうかを制御します。真に設定すると、即時チェックポイントが使用されます。つまり、PostgreSQLはチェックポイントをできるだけ早く完了します。デフォルトでは <code>false</code> です。	bool	false
jobs	バックアップのアップロードに使用される並列ジョブの数、デフォルトは2	*int32	false

MonitoringConfiguration

MonitoringConfigurationは、特定のクラスターのすべてのモニタリング構成を含むタイプです

Field	Description	Scheme	Required
customQueriesConfigMap	カスタムクエリを含む構成マップのリスト	[]corev1.ConfigMapKeySelector	false
customQueriesSecret	カスタムクエリを含むシークレットのリスト	[]corev1.SecretKeySelector	false

NodeMaintenanceWindow

NodeMaintenanceWindowには、基になるノードのアップグレード処理中に演算子が使用する情報が含まれています。

このオプションは、選択したストレージにより、ポッドがノード間で自由に移動できない場合にのみ役立ちます。

Field	Description	Scheme	Required
inProgress	進行中のノードメンテナンスアクティビティはありますか？	bool	true
reusePVC	既存のPVCを再使用する（ノードが再び起動するのを待つ）または使用しない（他の場所で再作成する）	*bool	true

PostgresConfiguration

PostgresConfigurationはPostgreSQLの構成を定義します

Field	Description	Scheme	Required
parameters	PostgreSQL設定オプション（ <code>postgresql.conf</code> ）	map[string]string	false
pg_hba	PostgreSQLホストベース認証ルール（ <code>pg_hba.conf</code> ファイルに追加される行）	[]string	false

RecoveryTarget

RecoveryTargetを使用すると、リカバリプロセスが停止する瞬間を構成できます。TargetTLIを除くすべてのターゲットオプションは相互に排他的です。

Field	Description	Scheme	Required
targetTLI	ターゲットタイムライン（ <code>*latest</code> 、 <code>current</code> または正の整数）	string	false
targetXID	ターゲットトランザクションID	string	false
targetName	ターゲット名前（ <code>pg_create_restore_point</code> で以前に作成される）	string	false

Field	Description	Scheme	Required
targetLSN	ターゲットLSN（ログシーケンス番号）	string	false
targetTime	PostgreSQLで許可されている明確な表現形式でのターゲット時間	string	false
targetImmediate	一貫した状態に達するとすぐにリカバリを終了します	*bool	false
exclusive	ターゲットを排他的に設定します（デフォルトは真）	*bool	false

RollingUpdateStatus

RollingUpdateStatusには、更新中のインスタンスに関する情報が含まれます

Field	Description	Scheme	Required
imageName	ポッドに入れたイメージ	string	true
startedAt	更新が開始されたとき	metav1.Time	false

S3Credentials

S3Credentialsは、ファイルをS3にアップロードするために使用される資格情報のタイプです

Field	Description	Scheme	Required
accessKeyId	アクセスキーIDへのリファレンス	corev1.SecretKeySelector	true
secretAccessKey	シークレットアクセスキーへのリファレンス	corev1.SecretKeySelector	true

StorageConfiguration

StorageConfigurationは、 PostgreSQLインスタンスのストレージの構成です

Field	Description	Scheme	Required
storageClass	データベースデータに使用する StorageClass（ PGDATA ）。使用可能な場合、PVCテンプレートを評価した後に適用されます。指定しない場合、生成されたPVCはデフォルトのストレージクラスによって満たされます。	*string	false
size	ストレージのサイズ。PVCテンプレートでまだ指定されていない場合は必須です。このフィールドへの変更は、作成されたPVCに自動的に再適用されます。サイズを小さくすることはできません。	string	true
resizeInUseVolumes	既存のPVCのサイズを変更、デフォルトは真	*bool	false
pvcTemplate	永続的ボリューム要求の生成に使用されるテンプレート	*corev1.PersistentVolumeClaimSpec	false

WalBackupConfiguration

WalBackupConfigurationは、WALストリームのバックアップの構成です。

Field	Description	Scheme	Required
compression	オブジェクトストアに送信する前に、WALファイルを圧縮します。使用可能なオプションは、空の文字列（圧縮なし、デフォルト）、 gzip または bzip2 です。	CompressionType	false
encryption	ファイルの暗号化を強制するとき（バケットがそのためにまだ構成されていない場合）。許可されるオプションは、空の文字列（バケットポリシー、デフォルト）、 AES256 および aws:kms です。	EncryptionType	false

ScheduledBackup

ScheduledBackupは、scheduledbackups APIのスキーマです

Field	Description	Scheme	Required
metadata	NA	metav1.ObjectMeta	false
spec	ScheduledBackupの望ましい動作の仕様。詳細： https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status	ScheduledBackupSpec	false
status	" ScheduledBackupの最新の監視ステータス。このデータは最新ではない可能性があります。システムによって設定されます。読み取り専用。詳細：https : https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status">https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status"	ScheduledBackupStatus	false

ScheduledBackupList

ScheduledBackupListには、ScheduledBackupのリストが含まれています

Field	Description	Scheme	Required
metadata	" 標準リストメタデータ。詳細：https : https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds">https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds"	metav1.ListMeta	false
items	クラスターのリスト	[]ScheduledBackup	true

ScheduledBackupSpec

ScheduledBackupSpecは、ScheduledBackupの望ましい状態を定義します

Field	Description	Scheme	Required
suspend	このバックアップが中断されていない場合	*bool	false
schedule	" Cronフォーマットのスケジュールは、https : //en.wikipedia.org/wiki/ Cronを参照してください。"	string	true
cluster	バックアップクラスター	v1.LocalObjectReference	false

ScheduledBackupStatus

ScheduledBackupStatusは、ScheduledBackupの監視状態を定義します

Field	Description	Scheme	Required
lastCheckTime	スケジュールの最新時刻	*metav1.Time	false
lastScheduleTime	バックアップが正常にスケジュールされた最後の時間である情報。	*metav1.Time	false

Credits

Cloud Native PostgreSQL（Operator for Kubernetes / OpenShift）は、EnterpriseDB Cloud Nativeチームによって設計、開発、テストされています。

- ガブリエーレ・バルトリニ
- ジョナサン・パッティアト
- フランチェスコ・カノバイ
- レオナルド・チェッキ
- ヴァレリオ・デル・サルト
- ニコロ・フェイ
- ジョナサン・ゴンザレス
- デンマークのカーン
- アナンド・ネドゥール
- マルコネンシアリーニ
- ガブリエーレ・クアレシマ
- ジテンドラワドル
- アダム・ライト

[page]