

EDB Postgres AI Hybrid Manager

リリース *1.3.4*

EnterpriseDB Corporation

2026 年 01 月 27 日

目 次

第 1 章 One platform, full control **3**
1.1 あなたにできること . . . 3
1.2 ここから始めます . . . 4

第 2 章 Hybrid Manager Release Notes **5**
2.1 Hybrid Manager 1.3.4 release notes . . . 5
2.2 Hybrid Manager 1.3.3 release notes . . . 6
2.3 Hybrid Manager 1.3.2 release notes . . . 6
2.4 Hybrid Manager 1.3.1 release notes . . . 7
2.4.1 HM コンソール . . . 7
2.4.2 高度な高可用性と分散高可用性 PGD . . . 8
2.4.3 バックアップ、リカバリー、ストレージ . . . 8
2.4.4 コアプラットフォーム機能と API . . . 8
2.5 Hybrid Manager 1.3 release notes . . . 9
2.5.1 リリースハイライト . . . 9
2.5.2 新機能と機能強化 . . . 10
2.5.3 の修正 . . . 12
2.6 Hybrid Manager 1.2 release notes . . . 13
2.6.1 リリースハイライト . . . 14
2.6.2 展開オプション . . . 14
2.6.3 新機能と機能強化 . . . 15
2.6.4 可観測性、モニタリング、インサイト . . . 16
2.6.5 サステナビリティ ESG レポートプレビュー . . . 16
2.6.6 プラットフォームの改善 . . . 16
2.6.7 リソース . . . 17

第 3 章 Known issues **19**
3.1 マルチ DC . . . 19
3.1.1 アップグレード後のマルチ DC 構成の損失 . . . 19
3.1.2 マルチ DC セットアップの場所ドロップダウンリストが空です . . . 19
3.2 コアプラットフォームとリソース . . . 21
3.2.1 `upm-beacon-agent` 複雑な環境ではメモリ制限が不十分です . . . 21
3.3 データベースクラスターエンジン . . . 21
3.3.1 EDB Postgres 分散 PGD クラスターに表示される誤ったデータベース名 . . . 21
3.3.2 PGD-X クラスターの作成が「PGD - アプリケーションユーザーの調整」フェーズでスタックする . . . 21
3.3.3 `max_connections` がデフォルト以外の場合、3 ノード PGD クラスターの作成に失敗する . . . 21
3.3.4 2 番目のデータグループを作成または複製するときに PGD データベース設定が複製されない . . . 22

3.3.5 AHA Witness ノードのリソースがオーバープロビジョニングされています 22
3.3.6 HA クラスターは、ストリーミングレプリケーション証明書の認証に verify-full の代わりに verify-ca を使用します 22
3.3.7 2 番目のノードが遅すぎるため、大規模な HA クラスターに参加できません 22
3.3.8 自己管理クラスターのナレッジベースを照会できません 23
3.4 バックアップ リカバリー 23
3.4.1 リージョンをまたがってボリュームスナップショットリカバリーを使用する場合、レプリカクラスターの作成が失敗する 23
3.4.2 デフォルトの並列構成が原因で WAL アーカイブが遅い 23
3.4.3 ボリュームスナップショットクラスターは WAL ファイルを自動的にプルーニングしません 23
3.5 AI ファクトリーとモデル管理 24
3.5.1 プロファイルキャッシュを使用した nim-nvidia-nvclip モデルの展開の失敗 . . . 24
3.5.2 モデルプロファイルのオブジェクトストレージパスが空の場合、AI モデルクラスターの展開が停止する 25
3.5.3 モデル名が間違っていると、LLM をリモートで呼び出すときに 404 エラーが発生します 26
3.6 HM コンソールと可観測性 26
3.6.1 アクティブなモデルクラスターのタグはモデルの詳細画面に表示されません 26
3.6.2 チャットモデルクラスターメトリックが Grafana モデル概要ダッシュボードにありません 26
3.6.3 ユーザー作成の Grafana ダッシュボードは、プラットフォームの再展開/アップグレード後に持続しない 26
3.7 マイグレーション 26

第 4 章 EDB Postgres AI Hybrid Manager **27**
4.1 なぜハイブリッドマネージャーを使うのか？ 27
4.2 主要な機能 27
4.3 サポートされている環境 28
4.4 統合ソリューション 28
4.5 詳細はこちら 28
4.6 はじめる 29
4.6.1 Hybrid Manager architecture 29
4.6.2 Supported database types 31
4.6.3 Supported Postgres distributions 39

第 5 章 Installing Hybrid Manager **41**
5.1 インストールする前に 41
5.2 プラットフォーム別のインストールガイド 41
5.2.1 RedHat OpenShift RHOS 41
5.2.2 Rancher RKE2 42
5.2.3 アマゾン ウェブ サービス EKS 42
5.2.4 Google Cloud PlatformGKE 42
5.3 プリインストールされたハードウェアソリューションを検討していますか? 42
5.3.1 Hybrid Manager installation methods for Rancher RKE2 42
5.3.2 Install Hybrid Manager on Rancher RKE2 59
5.3.3 Install Hybrid Manager on AWS EKS 82

5.3.4 Install Hybrid Manager on GCP GKE . . . 123
5.3.5 Customizing your installations . . . 139

第 6 章 Using Hybrid Manager 169
6.1 どこから始めるべきか . . . 169
6.2 ユーザー向け . . . 169
6.3 Kubernetes 管理者の場合 . . . 170
6.3.1 Hybrid Manager Terminology . . . 170
6.3.2 Hybrid Manager console . . . 175
6.3.3 For Users . . . 184
6.3.4 For Kubernetes Administrators . . . 466

第 7 章 AI Factory in Hybrid Manager 587
7.1 構築できる AI ソリューションの例 . . . 588
7.2 ラーニングパス . . . 588
7.3 ユースケースとソリューション . . . 589
7.3.1 Gen AI ワークロード . . . 589
7.3.2 モデルの管理と提供 . . . 589
7.3.3 ベクトルエンジン . . . 589
7.4 Hybrid Manager の学習コンテンツ . . . 589
7.5 AI Factory がハイブリッドマネージャーにどのように適合するか . . . 589
7.6 はじめる . . . 590
7.6.1 Getting Started . . . 590
7.6.2 Manual . . . 611
7.6.3 How to Use Gen AI Builder . . . 628
7.6.4 Setup . . . 629
7.6.5 How to Use Models . . . 640
7.6.6 Operate and Observe . . . 641

第 8 章 Analytics in Hybrid Manager 651
8.1 ハイブリッドマネージャーでアナリティクスを使用する理由 . . . 651
8.2 コアコンセプトと機能 . . . 651
8.2.1 Apache Iceberg ® . . . 651
8.2.2 デルタレイク . . . 652
8.2.3 PGD を使用した階層テーブル . . . 652
8.2.4 レイクハウスクラスター . . . 652
8.3 これらの機能がどのように連携するか . . . 653
8.4 はじめに: クイックスタートチェックリスト . . . 653
8.5 さらに探索する . . . 653
8.5.1 Analytics Concepts in Hybrid Manager . . . 654
8.5.2 Delta Lake in Hybrid Manager . . . 656
8.5.3 Apache Iceberg ® in Hybrid Manager . . . 658
8.5.4 EDB Postgres Lakehouse Clusters on Hybrid Manager . . . 661
8.5.5 Tiered Tables in Hybrid Manager . . . 663

第 9 章 Sovereign Data and AI Factory 667
9.1 ハードウェアベースのシステムを選択する理由は何ですか? . . . 667
9.2 中身は何ですか . . . 667

9.3 サポートされているワークロード 668
9.4 ハードウェア構成 668
9.5 Explore 機能 668
9.5.1 Concepts and architecture 668
9.5.2 Frequently asked questions 671
9.5.3 Glossary and key terms 672
9.5.4 System specifications 674
9.5.5 Common workloads and use cases 678
9.5.6 What to expect 680

EDB Postgres AI Hybrid Manager HM は、 EDB Postgres AI プラットフォームのコントロールプレーンです。Kubernetes が実行されている場所ならどこでも実行される**ソブリン AI、分析、および Postgres 管理ソリューション**です。データベースの展開、GenAI ワークロードの実行、および **クラウド**、 **オンプレミス** 、または **ハイブリッド環境**全体で Postgres ネイティブ分析のスケーリングのための統一エクスペリエンスを提供します。

現在、オンプレミスの **Red Hat OpenShift RHOS** 、 **Rancher RKE2** 、およびインクラウドの場合 **Amazon EKS** 、および **Google GKE の次の Kubernetes ディストリビューションでサポートされています。** 。

第1章　One platform, full control

HM は、最新の Postgres ベースのデータプラットフォームをサポートするために必要なインフラストラクチャ、データ、およびアプリケーションレイヤーをまとめます。

- **ソブリン AI** —独自のインフラストラクチャを使用して、AI 搭載ワークロードを構築および提供し、GPU リソースと GenAI パイプラインを完全に制御します。
- **Postgres ネイティブ分析** Postgres の互換性と制御を維持しながら、Iceberg や Delta Lake のようなオープンデータ形式にオフロードします。
- **オペレーションエクセレンス** —Postgres クラスターのバックアップ、スケーリング、モニタリング、およびクラスターライフサイクル管理を簡素化します。

クラウドで展開するかオンプレミスで展開するかにかかわらず、ハイブリッドマネージャーは、データと AI スタック全体で**可観測性、ガバナンス、および復元力** を保証します。

1.1 あなたにできること

Area	Capabilities
Database provisioning	コンソール、テンプレート、または宣言的マニフェストを使用して、高可用性の PostgreSQL または分散 Postgres クラスターを展開します。
AI integration	sovereign GenAI pipelines を実行し、KServe でモデルを提供し、インフラストラクチャによって管理される AI ナレッジベースを構築します。
Analytics	コールドデータを Delta Lake or Iceberg にオフロードし、Postgres 構文で照会し、ハイブリッドトランザクション/分析処理を有効にします。
Monitoring and observability	Prometheus、Grafana、Loki を活用する統合ダッシュボードを使用して、クエリの状態、リソースの使用状況、システムアラートに関する詳細な可視性を取得します。
Security and access management	ID プロバイダーと統合し、プロジェクトとクラスターの RBAC を定義し、データベースとサービスへの安全なアクセスを管理します。
Database migration	外部データベース Oracle、自己管理の Postgres から HM 管理クラスターにスキーマとデータを移行します。移行とスキーマ評価、データ転送スナップショット、および継続的レプリケーションストリーミングのための統合サービスを活用して、最小限のダウンタイムの移行を実現します。
	NA
	NA

1.2 ここから始めます

- *Install Hybrid Manager on AWS EKS* → Red Hat OpenShift RHOS、Rancher RKE2、EKS、または GKE。
- **ソブリンデータと *AI* ファクトリーハードウェアアプライアンス** → ソブリンデータと AI Factory ハードウェア
- *Using Hybrid Manager* → バックアップ/リストア、モニタリング、移行、クラスター作成、マルチ DC 展開など
- *AI Factory Architecture on Hybrid Manager* → ソブリン GenAI パイプライン、モデル提供、および取得システム
- *Analytics Concepts in Hybrid Manager* → デルタレイク、氷山、階層化ストレージ、およびレイクハウスパターン
- *Hybrid Manager 1.2 release notes*

第2章　Hybrid Manager Release Notes

Hybrid Manager 1.3.3 release notes	15 Dec 2025
Hybrid Manager 1.3.2 release notes	27 Nov 2025
Hybrid Manager 1.3.1 release notes	28 Oct 2025
Hybrid Manager 1.3 release notes	10 Oct 2025
Hybrid Manager 1.2 release notes	17 Jun 2025

2.1 Hybrid Manager 1.3.4 release notes

発売日 2026 年 1 月 15 日

Hybrid ManagerHM 1.3.4 は、バグ修正、内部オペランドとサードパーティの依存関係の更新、プラットフォーム全体の安定性を向上させる更新を含むメンテナンスリリースです。

Type	Description
Enhancement	Postgres クラスターの自動 WAL 保持とクリーンアップの向上。先行書き込みログ WAL がオブジェクトストレージから正しくプルーニングされるように、HM は、新しく作成される高度な高可用性および分散高可用性 PGD クラスターのデフォルトのスケジュールされたバックアップ方法として barman-cloud を利用するようになりました。既存のクラスターと、新しく作成された単一ノード、高可用性 PSR、または分析クラスターの場合、HM はボリュームスナップショットとともに定期的な Barman バックアップを自動的に実行するようになりました。
Bug fix	無効な名前スペースを含むモデルクラスターを作成すると、汎用 500 エラーが発生する問題を修正しました。 HM コンソールは、入力中にモデルクラスター名を検証して、スペースを防止するようになりました。さらに、無効な名前が指定された場合、HM は説明的なメッセージを含む 400 エラーを正しく返すようになりました。
Bug fix	エアギャップ環境でプライベート Red Hat Quay レジストリを使用する場合、イメージの検出が失敗した問題を修正しました。以前は、`upm-beacon-agent`は、正しいレジストリ URL と認証トークンで構成されている場合でも、イメージを正常に取得できませんでした。この修正により、イメージ検出が、制限されたネットワーク環境での安全なプライベートレジストリと安全でないプライベートレジストリの両方で正しく動作するようになりました。

次のページに続く

表 2 – 前のページからの続き

Type	Description
Bug fix	Persistent Volume ClaimsPVC が誤ってポッドにマッピングされる`cnp_pg_replication_in_recovery`アラートの問題を修正しました。以前は、アラートロジックが PVC のポッド名を再利用しようとしたため、`-wal`サフィックスを使用する WAL ボリュームで障害が発生しました。モニタリングロジックは`cnpg_instance_id`を正しく使用してメトリックを結合し、正確なレプリケーションリカバリステータスとすべてのボリュームタイプのアラートを保証します。
Bug fix	`upm-griptape`コンポーネントの「missing module」エラーを修正しました。以前は、`upm-griptape`サービスのバージョン v1.5.16 および v1.7.6 は、Python 依存関係、特に`fastapi`モジュールが欠落しているため、起動に失敗していました。この修正では、依存関係の競合を解決し、必要なすべてのモジュールがサービスイメージに正しくバンドルされるようにします。

!!!note Postgres は、EDB でサポートされているすべてのディストリビューションを指します。- PostgreSQL - EDB Enterprise Postgres-以前の EDB Postgres Extended ServerPGE - EDB Enterprise PostgresOracle 互換-以前の EDB Postgres Advanced Server !!!

2.2 Hybrid Manager 1.3.3 release notes

発売日 2025 年 12 月 15 日

Hybrid Manager 1.3.3 は、バグ修正と安定性の更新を含むメンテナンスリリースです。

Type	Description
Bug fix	Barman がデフォルトのバックアップソリューションとして構成されている場合でも、[Restore Cluster] ボタンを正常にクリックできるようになりました。
Bug fix	VPC エンドポイントが使用中の場合、AWS STS 要求はデフォルトのグローバルエンドポイントではなく、リージョンエンドポイントに正しくルーティングされるようになりました。
Bug fix	クイックサインイン URL を導入して、ユーザーが ID プロバイダー IdP に直接移動できるようにします。
Bug fix	すべての内部 gRPC API エンドポイントの認証と承認を強化して、不正なアクセスを防止することにより、 CVE-2025-14038 に対するプラットフォームのセキュリティが向上しました。

2.3 Hybrid Manager 1.3.2 release notes

発売日 2025 年 11 月 27 日

Hybrid Manager 1.3.2 は、バグ修正と安定性の更新を含むメンテナンスリリースです。

Type	Description
Bug fix	インストール中にバックアップの保持期間と圧縮設定を変更できなかった問題を修正しました。ユーザーは、インストール中に `internal_db_backup_compression`および`internal_db_backup_retention`デフォルト30 日を定義できるようになり、制御されていない内部ストレージの消費とデータライフサイクルポリシーに関する問題を防止できます。
Bug fix	`beacon-agent`構成を修正して、欠落しているリリースアノテーションを含めます。 `org.enterprisedb.operandset.version`が`beacon-server`に適切に送信され、リリースレポートの不正確さを修正するようになりました。
Bug fix	`ImageLibrary`が不要なイメージをプルし、パフォーマンスの低下をもたらした問題を修正しました。フィルタリングロジックを追加し、Helm チャートの構成を修正して、インストール中にこれらの制限が適切に適用されることを確認します。
Bug fix	RHOS 展開で kubelet との接続エラーを発生させた`spire-agent`コンポーネントに修正を追加しました。
Bug fix	`kube-state-metrics`に修正を追加して、サービスがカスタムメトリックをエクスポートできないようにしました。
Bug fix	`beacondb`バックアップが失敗する原因となる修正を追加しました。
Bug fix	RHOS 展開で問題を発生させた`barman-plugin-v0-8-0-migrator`プラグインの修正を追加しました。
Bug fix	カスタム証明書発行者を使用するようにハイブリッドマネージャーを構成するときにエラーが発生した問題を修正しました。
Bug fix	AI モデルのソート動作を修正し、ユーザーが名前の降順でソートできるようにしました。
Bug fix	**Estate** テーブルのフィルタリングの問題を解決し、PostgreSQL データベースにフィルタを適用する際に正確な結果を保証します。
Bug fix	移行ポータルの **Delete** および **Dismiss** ボタンの可視性を解決し、これらのアクションがユーザーの権限に基づいて正しく制限されるようになりました。
Bug fix	PGD ナレッジベースが GenAI Builder インターフェイスに表示されない問題を解決しました。

2.4 Hybrid Manager 1.3.1 release notes

発売日 2025 年 10 月 28 日

このリリースには、バグ修正とセキュリティ更新が含まれています。

2.4.1 HM コンソール

- HM コンソールホームページの Postgres クラスターの合計数は、個々のクラスターノード EFM または CNP レプリカを適切に除外して、合計の膨らみや不正確な合計を防ぐようになりました。
- HM コンソールの **Connect** タブのデータベース接続情報に、高度な高可用性および分散高可用性 PGD クラスターに必要な’bdrdb’の代わりに’edb_admin’が誤って表示される問題を修正しました。
- クラスターの詳細ページの **モニタリング**タブのデータベースサイズに、自己管理クラスターのストレージメトリックが正しく表示されるようになりました。

- HM コンソールの **展開の詳細**ページに、アクティブなクラスターのタグが正しく表示されるようになりました。

2.4.2 高度な高可用性と分散高可用性 PGD

- `max_worker_processes` の検証ロジックを修正して、レプリカグループの GUC 値がプライマリグループよりも低いと誤って検証されないようにします。
- 空の GUC 構成ペイロードを送信すると、既存の設定が保持されず、代わりにデフォルト値へのリセットが誤ってトリガーされる問題を修正しました。
- データグループを複製するときに、GUC `max_connections` などがペイロードに適切に含まれることを確認し、不要な検証の失敗を防ぎます。
- 新しい PGD グループの正常な追加を妨げたバグを解決しました。
- リコンサイルフェーズ中に PGD-X クラスターの作成が停止する可能性がある問題を修正しました。
- 新しい PGD データグループを追加する場合、データベース設定が最初のデータグループから正しく複製され、一貫性が維持されるようになりました。
- PGD クラスターステータスの可視性は、クラスター、クラスターラッパー、および pgdgroup リソースの新しいコントロールプレーンリソース CR ステータスパネルで強化されました。
- コンソールは、HM コンソールの 8 GB の実効 `max_wal_size` を正しく反映するようになり、コンソールが 1024 MB と表示される不一致を解決しました。
- PGD に必要な `shared_preload_libraries` 設定に pgaa が正しく含まれるようになりました、または設定が正しくない場合、HM コンソールは適切な表示を提供します。

2.4.3 バックアップ、リカバリー、ストレージ

- 保持期間後にクラスターメタデータが完全に削除されず、S3 /オブジェクトストレージに孤立した Barman ファイルが残ったバグを修正しました。
- 完了としてマークされた WAL ファイルは、構成された制限を超えて誤って保持されなくなりました。
- upm-beacon エージェントコンポーネントの問題を解決して、バックアップ削除プロセスを安定させました。
- クロスロケーションクラスターリストア機能は、方法として pg_basebackup を正しく使用するようになりました。

2.4.4 コアプラットフォーム機能と API

- 無効なポッド名の形式が原因で `accm-server` のクラッシュを引き起こしたバグを修正しました。
- EKS の上限制限が原因でクラスターの作成を妨げる可能性がある問題を解決しました。
- セカンダリロケーションビーコンディスパッチャーランナーがプライマリビーコンサーバーに接続できないバグを修正しました。

- HM コンソールの **Model Profile Paths** フィールドは、構成中にオプショナルとして正しくマークされるようになりました。
- `status.backupStatus` などの特定のステータスフィールドが空に表示される API のバグを修正しました。
- プラットフォーム全体でクラスターメタデータの可用性が向上し、信頼性が向上します。
- LLM をリモートで呼び出すときにエラーが発生した問題を修正し、外部 API 通信の成功を保証します。
- ビーコンエージェントとサーバーのデフォルトのメモリ割り当てを増加し、ブートストラップ値で制限を構成可能にしました。
- プロセスグループが 1 時間後に接続の再試行を停止し、接続と安定性の継続的な試行を保証するバグを修正しました。

2.5 Hybrid Manager 1.3 release notes

発売日 2025 年 10 月 10 日

EDB Postgres AI Hybrid Manager v1.3 のリリースを発表できることを嬉しく思います。

2.5.1 リリースハイライト

- **アップグレードの信頼性**サポートされているすべてのマイナーリリースとパッチリリースでプラットフォームのアップグレードを強化することに成功しました。これには、既存の EDB Cloud Service 旧 Big Animal の顧客がハイブリッドマネージャーに移行するための完全に検証されたセルフサービスパスが含まれています。
- **重要なバックアップおよびリカバリー機能の保護**マルチロケーションのディザスターリカバリーを導入し、複雑なスイッチオーバーと HA 管理を簡素化します。
- **統合されたオブザーバビリティ**自己管理のベアメタルおよび CloudNativePG クラスターコミュニティ CNPG を含むを監視するための機能パリティを達成し、すべての EDB インスタンスに真の単一ペインを提供します。
- **プラットフォームの安定性と柔軟性** 検証された EDB Cloud Service 旧 Big Animal からハイブリッドマネージャーへの移行、SUSE Rancher への展開サポートの拡張、および重要なバックアップとリカバリフローの保護。
- **移行用のデータソブリン AI** 移行 AI コパイロットは、ハイブリッドマネージャー内でホストされるモデルを使用できるようになり、移行クエリの安全なプラットフォーム境界内を保証します。
- **安全な AI モデルアクセス**新しい動的プロキシレイヤーと一元化された AI モデルカタログ管理を介して、展開された KServe モデルへの安全な認証された外部アクセスが有効になりました。

2.5.2 新機能と機能強化

プラットフォームの管理と信頼性

SUSE Rancher プラットフォームのサポート Hybrid Manager は、SUSE Rancher 環境での展開が認定およびサポートされるようになりました。

EDB Postgres DistributedPGD 6.1 のサポートハイブリッドマネージャーは、EDB Postgres DistributedPGD バージョン 6.1 を実行しているクラスターのプロビジョニングと管理をサポートするようになりました。このアップグレードにより、お客様は、統合されたハイブリッドマネージャープラットフォーム内の最新の PGD リリースが提供する最新の機能、パフォーマンスの最適化、およびセキュリティ強化を利用できます。

分離されたオペランドのライフサイクル Postgres オペランドリリースを、メインの Hybrid Manager リリースサイクルから分離しました。この根本的な改善により、データベースエンジンコンポーネントのより速く、より頻繁に、中断の少ない更新が可能になることにより、プラットフォームの俊敏性が向上します。

エンジニアドシステム展開のローカルストレージのサポートエンジニアドシステム内のワーカーノード上のローカルディスクから高性能、低レイテンシーの永続ボリュームを直接プロビジョニングするサポートを導入しました。これは、LVM Storage Operator TopoLVM を使用し、共有外部ストレージを利用しないオンプレミス展開の堅牢なデータの永続性を有効にします。

管理者ブロードキャストメッセージ管理者プロジェクト所有者を含むは、ハイブリッドマネージャー内のすべてのユーザーに重要なプラットフォーム全体またはプロジェクト固有のメッセージをブロードキャストできるようになりました。この機能は、計画的なメンテナンス、停止、またはその他の重要な運用の更新を通知するのに最適です。これらの強制通知は、受信ボックスと情報ペインを介してユーザーインターフェイスに表示され、すべてのユーザーに通知されます。

ディザスターリカバリーと高可用性

マルチロケーションの高可用性とディザスターリカバリー DR ハイブリッドマネージャーは、CloudNativePG の分散トポロジ機能を活用して、復元力の高い Postgres アーキテクチャ

across distinct Kubernetes clusters (e.g., across regions) の構築をサポートするようになりました。これにより、ユーザーはネイティブの Postgres ストリーミングレプリケーションと WAL アーカイブを使用して、マルチロケーションクラスターを構成および管理し、堅牢な DR と高可用性を実現できます。

バックアップ頻度ハイブリッドマネージャーコントロールプレーンのバックアップ頻度をカスタマイズする機能を導入しました。お客様は、Helm 値ファイルを介して Velero バックアップスケジュールを直接定義できるようになったため、目標復旧ポイント RPO を正確に制御し、特定の運用要件を満たすことができます。

ボリュームスナップショット復元の検証ハイブリッドマネージャーには、互換性のないストレージ構成が原因で復元操作が停止して失敗するのを防ぐための強化された検証が含まれるようになりました。新しいクラスターのストレージ構成たとえば、単一のデータボリュームから分割データ/WAL ボリュームへの変更など、ソーススナップショットと互換性がない場合、ボリュームスナップショットからクラスターを復元しようとすると、システムはブロックまたは明確な警告を発行するようになりました。

RHOS でのクロス名前空間ボリュームスナップショットリストアのサポート私たちは、以前は別の名前空間にボリュームスナップショットを復元することを妨げていた Red Hat OpenShift のデフォルトのセキュリティコンテキスト制約 SCC を克服するソリューションを設計することに成功しました。Hybrid Manager は、必要なセキュリティ設定を自動的に構成して、クロス名前空間のリストア操作を許可するようになり、

OpenShift 環境に展開されたクラスターのディザスターリカバリーとマルチテナント機能を大幅に強化します。これにより、任意のターゲット名前空間へのデータの移行またはリカバリーのために、ボリュームスナップショットバックアップを確実に復元できます。

RHOS のバックアップサポート ハイブリッドマネージャーは、Red Hat OpenShiftRHOS およびエンジニアドシステムに展開されたクラスターのボリュームスナップショットベースのバックアップおよびリカバリー操作をサポートするようになりました。これにより、これらの重要な環境に、より高速なプラットフォームネイティブの B&R エクスペリエンスを提供します。

アップグレードエクスペリエンス

シームレスなハイブリッドマネージャーのアップグレードサポートされているすべてのマイナーリリースとパッチリリースの間で upgrades の信頼性を高めるための基礎作業を実装しました。これにより、プラットフォームの安定性が強化され、回帰のリスクが最小限に抑えられ、ユーザーに明確な期待とプラットフォーム移行の信頼できるパスが提供されます。

EDB Cloud Service からの移行 既存の Cloud Service 旧 Big Animal の Bring Your Own Account BYOA の顧客が簡単かつ確実にハイブリッドマネージャーに移行できるように、無停止のセルフサービスパスをサポートするようになりました。

可観測性、モニタリング、および洞察

統合されたオブザーバビリティハイブリッドマネージャーは、自己管理のベアメタル Postgres インスタンス、および自己管理の CloudNativePGCNP および EDB Community CloudNativePG インスタンスからの重要なテレメトリと状態メトリックの収集をサポートし、それらを同等にします HM 管理対象クラスター。新しい metrics には、ホストメトリック、WAL アクティビティ、ストレージ使用率、IOPS 統計などの重要な操作データが含まれています。これにより、Postgres インスタンスが展開されている場所に関係なく、Hybrid Manager コンソール内の統合された包括的な可観測性が可能になります。

Postgres 構成の推奨事項ハイブリッドマネージャーは、Postgres 設定の調整、データベースパフォーマンスの向上、リソース使用の最適化に役立つ推奨事項を提供します。これらの提案は、クラスターの詳細ビューの**推奨事項** タブからアクセスできます。

AI Factory を使用した GenAI ワークロード

展開された AI モデルへの安全な外部アクセスハイブリッドマネージャーは、 EDB Postgres AI プラットフォームの外部から展開された KServe モデルへの安全な認証アクセスをサポートするようになりました。これは、動的プロキシを実装し、堅牢な認証と承認 AuthN / AuthZ のために Istio を活用することにより実現されます。この機能強化により、開発者は、完全なプラットフォームのガバナンスを維持しながら、展開された AI モデル OpenAI 互換エンドポイントを含む外部アプリケーションに安全に統合できます。

一元化された AI モデルカタログハイブリッドマネージャーには、組織の AI モデルの中央リポジトリとして機能する専用のモデルカタログが含まれるようになりました。このカタログは、許可されたユーザーに、AI ガバナンスと迅速な展開に必要なツールを提供します。ユーザーは、AI モデルのインベントリ全体をリスト、作成、インポート、編集、および表示できるようになりました。カタログは KServe と直接統合されているため、ユーザーはハイブリッドマネージャーインターフェイスからワンクリックで AI モデルをモデルサービスインフラストラクチャに展開できます。

AI モデルサービングクラスター KServe の専用管理ハイブリッドマネージャーは、基になる AI モデルサービングインフラストラクチャの集中管理を提供するようになりました。ユーザーと管理者は、展開された KServe クラスターとモデルを統合した可視性と制御を取得できるようになりました。

- モデルクラスターの概要新しいユーザーインターフェイスには、実行中のすべてのモデルサーバークラスターのリストと、監視と管理のための専用の詳細ページが表示されます。
- REST API エンドポイント実行中のモデルクラスターをリストし、詳細なクラスター情報を取得するために専用の REST API を利用でき、外部統合と自動化を促進します。

マイグレーション

移行ポータル AI コパイロットのデータソブリン AI AI コパイロットは、ハイブリッドマネージャー内で configured to exclusively utilize AI Factory Models ホストできるようになりました。これにより、機密の移行クエリとデータがプラットフォームの境界内に残り、サードパーティベンダーに送信されません。

コンソールでの Postgres 移行評価の詳細ハイブリッドマネージャーの移行評価機能が拡張され、サポートされているソースデータベースとして Postgres が含まれます。既存の migration assessment details page は、Postgres 固有の情報を提供するように更新され、Oracle 評価と併せて Postgres の移行の準備状況の統合された詳細なビューをユーザーに提供します。

エージェントスキーマの選択とフィルタリングユーザーは、`beacon-agent.yaml` 構成の新しい databases.schema.filter オプションを使用して、移行エージェントがソースデータベースから取り込むスキーマを正確に制御できるようになりました。これにより、プラットフォームユーザーは含めるまたは除外するスキーマのサブセットを定義できます。

更新されたデータ移行サービスエージェントデータ移行の基になるコンポーネント、以前は EDB DMS リーダーおよびライターとして知られていましたが、統合され、EDB DMS Agent に更新されました。 Hybrid Manager 1.3 で有効なこの変更は、すべてのデータの取り込みと移行のワークフローに合理化された統合エージェントを提供します。

データベース移行ドキュメントの強化構成と明確性を向上させるために、すべてのデータベース移行コンテンツを大幅にオーバーホールしました。すべてのコンテンツが Migrating databases セクションの下で統合され、明確な PLANNING、MIGRATING、および REFERENCE カテゴリにグループ化されました。

2.5.3 の修正

- **データベース移行ページの UI 500 エラーを解決**データベース項目の数がデフォルトのページネーションサイズを超えると、Migration Databases ビューが誤って HTTP 500 エラーを返す問題を修正しました。
- **解決されたクロスバージョンの復元の問題**メジャーバージョンのアップグレードを試行した後の復元操作中に、ユーザーが正しい Postgres イメージバージョンを選択することを以前にブロックした問題を修正しました。ユーザーは元の Postgres バージョンイメージ v16 などを正常に選択し、新しいクラスターにアップグレード前のバックアップを復元できるようになりました。メジャーアップグレードが失敗した場合、または放棄する必要がある場合に信頼できるロールバックパスを提供します。
- **ビーコンエージェントサイドカーのデフォルトのログレベルの調整** Postgres ポッドの `beacon-agent` サイドカーのデフォルトログレベルが不必要に `DEBUG` に設定されていたため、特に GKE デプロイメントで、過度にノイズの多いログが発生しました。デフォルトのログレベルが `INFO` に変更されて、ログの混乱を削減し、関連する操作情報に焦点を当てました。

- **CNP フェールオーバー後の重複メトリックの修正**外部マルチノード CloudNativePG クラスターをハイブリッドマネージャーに接続すると、プライマリノードのフェールオーバーまたはプロモーションに続いて pg_node レベルのチャートに重複したメトリックシリーズが表示される問題を修正しました。これらの外部クラスターのメトリック収集は、トポロジの変更後に正確に報告されるようになりました。
- **PGD / BDR 内部テーブルの誤った推奨事項の排除**クエリ推奨機能が内部 EDB Postgres DistributedPGD または BDR 管理テーブルの提案を誤って表示する問題を修正しました。基本的なメカニズム `query_advisor` 拡張機能が更新され、内部システムスキーマを除外できるようになり、推奨事項がエンドユーザーアプリケーションテーブルにのみ焦点を当て、内部インフラストラクチャの状態に関する誤ったアラームを防ぎます。
- **クラスターの状態の初期バックアップ**クラスタープロビジョニングワークフローが改善され、最初の必須バックアップが正常に完了するまで、新しく展開されたクラスターがハイブリッドマネージャーで「展開済み」または「正常」としてマークされないように。この変更により、即時復元ポイントのない一見正常なクラスターに依存する可能性のある状況が防止されます。
- **EDB Postgres 分散 PGD の最小リソース要件の増加** PGD 展開の安定性と信頼性を向上させるために、最小必要ストレージとメモリの割り当てを増加しました。最小ディスクサイズが 1GB から 10GB に増加し、デフォルトの最小メモリが 2GB に設定されました。この変更は、低メモリ環境での RAFT コンセンサス失敗などの重要な運用上の問題を防止し、監視ノードを含むすべての PGD コンポーネントに一貫して適用されます。
- **ロケーション固有のネットワークパラメーターの API 検証** `allowedIpRanges` などのネットワークパラメーターが IP `allowlisting` をサポートしていない展開場所に指定されている場合、クラスター作成要求を拒否するためにプロビジョニング API に検証を実装しました。これにより、ユーザーは無効な構成に関するフィードバックをすぐに受け取れるようになり、ロケーション機能の一貫した使用が促進されます。
- **エステートページのタグの可視性の復元** Postgres クラスターに割り当てられたタグが、メインの **エステート**ページの **タグ**列に表示されない問題を修正しました。タグが API から正しく取得され、関連するすべてのユーザーインターフェイスビューに一貫して表示されるようになりました。
- **Kubernetes 1.33 でのインストール障害の解決**基になるクライアントライブラリの非互換性のため、Kubernetes バージョン 1.33 で `transporter` コンポーネントがインストールに失敗した問題を修正し、最新の K8s 環境でのインストールの成功を保証します。

2.6 Hybrid Manager 1.2 release notes

発売日 2025 年 6 月 17 日

EDB Postgres AI Hybrid Manager v1.2 の最初の一般公開リリースを発表できることを嬉しく思います。

EDB Postgres AI Hybrid Manager は、ソブリン AI、分析、および Postgres 管理ソリューションである EDB Postgres AI プラットフォームのコントロールプレーンです。このリリースでは、高度な分析、生成 AI、および運用管理を単一の凝集プラットフォームに統合することにより、データの操作方法に革命をもたらします。

2.6.1 リリースハイライト

- **ソブリン AI ＆分析プラットフォーム**ハイブリッドマネージャーは、最新のデータと AI アプリケーションを構築するための完全なプラットフォームになりました。Postgres ネイティブのレイクハウスと、統合されたガバナンスを備えたソブリン GenAI ワークロードをすべて独自のインフラストラクチャ内で展開します。
- **新しい展開オプションソブリンデータと AI ファクトリー**外部を使用せずに独自のデータセンターにプロダクショングレードの Postgres および AI ワークロードを展開するための、EDB と Supermicro ®が共同設計した完全に統合されたハードウェアおよびソフトウェアアプライアンスを紹介します。依存関係。
- **Postgres-Native Lakehouse** 画期的な分析スタックがビルトインされました。**レイクハウスクラスター**を展開して、オブジェクトストレージ内の **Apache Iceberg ®** や **Delta Lake フォーマット**のようなオープンデータフォーマットに対して直接高速のベクトル化クエリを実行します。
- **GenAI ワークロード** の AI ファクトリー ローコード **GenAI Builder** 、GPU アクセラレーションを使用して提供する統合モデル、データベース内**セマンティック検索**、および包括的な Python を使用して、強力な AI アプリケーションを構築して提供します。SDK
- **統合管理とプロアクティブな診断**完全に再設計されたモニタリングインターフェイス、新しいクラスターレベルのスコアカード、**プロアクティブなクエリの推奨事項**、および詳細なクエリレイテンシー統計を体験してください。

2.6.2 展開オプション

EDB Postgres AI Hybrid Manager v1.2 は 2 つの方法で利用でき、運用ニーズに最大限の柔軟性を提供します。

ソブリンデータと AI ファクトリーハードウェアアプライアンス

最大限の制御とゼロの外部依存関係を必要とする組織にとって、**ソブリンデータおよび AI ファクトリー**は、完全に統合され、事前に組み立てられたハードウェアおよびソフトウェアプラットフォームです。ラックマウントシステムとしてデータセンターに提供され、データ主権、セキュリティ、ライフサイクルの信頼性が最優先される環境に理想的な選択です。

このエンジニアドシステムはスケール向けに最適化されており、インストールした瞬間に高度な Postgres、vector、AI パイプラインを実行する準備が整います。これには、検証された Supermicro サーバー、ライフサイクル管理される Kubernetes、および AI 推論用のオプションの GPU ノードを備えた完全なハイブリッドマネージャーソフトウェアスイートが含まれています。

ソフトウェアのみの展開

既存のインフラストラクチャを持つ組織の場合、EDB Postgres AI Hybrid Manager は、優先する Kubernetes ディストリビューションにソフトウェアのみのソリューションとして展開できます。**Amazon EKS** 、**Google GKE** 、および **Red Hat OpenShift RHOS** に対して認定されています。

2.6.3 新機能と機能強化

ソブリン AI ＆分析プラットフォーム

Hybrid Manager v1.2 は、Postgres プラットフォームで直接次世代の分析と AI ワークロードを構築および管理するための強力な統合ツールスイートを紹介します。

Postgres ネイティブアナリティクス

Postgres の能力とオブジェクトストレージのスケールを組み合わせた、最新の費用対効果の高いデータアーキテクチャを構築します。Hybrid Manager は、分析スタック全体を管理するツールを提供します。

- **レイクハウスクラスター**オブジェクトストレージでスケーラブルなベクトル化されたクエリ実行を提供する専用のコンピューティングクラスターをプロビジョニングおよび管理します。これらは、Postgres ネイティブのレイクハウスのクエリエンジンとして機能します。
- **Apache Iceberg ®サポート** Iceberg 形式で大規模な分析テーブルをネイティブに管理します。 EDB Postgres 分散 PGD で **階層テーブル**を使用して、コールドデータパーティションを Iceberg に自動的にオフロードし、ストレージコストを削減しながら、レイクハウスクラスターを介してクエリ可能にします。
- **デルタレイクフォーマットの統合**既存のデータレイクとシームレスに統合します。Lakehouse クラスターは、Delta Lake オープンフォーマットを使用してテーブルを直接照会できるため、Spark のような他のビッグデータパイプラインによって生成されたデータの高速分析が可能になります。
- **PGD を使用した階層テーブル**テーブルを自動的にパーティション化し、コールドパーティションを Iceberg 形式のオブジェクトストレージにオフロードすることにより、大規模な時系列データセットを効率的に管理します。1 つのクエリで、Postgres のホットトランザクションデータとデータレイクのコールド履歴データの両方にまたがることができます。

AI Factory を使用した GenAI ワークロード

EDB Postgres AI Hybrid Manager v1.2 では、エンタープライズグレードの展開用に設計された包括的な GenAI 機能スタックである **AI Factory** が導入されています。

- **GenAI Builder** 単純なワンショットプロンプトから複雑な RAG パイプラインに至るまで、AI アプリケーションを迅速にプロトタイプ、反復、および展開するための Web ベースのローコードインターフェイス。
- **モデルサービス**可用性と自動スケーリングのビルトインサポートを備えた、GPU アクセラレーションインフラストラクチャでの AI 推論用の **NVIDIA NIM** コンテナを展開および管理します。
- **セマンティック検索** Postgres に組み込まれているベクトル検索機能を使用して、GenAI アプリケーションの取得を強化します。テキスト、HTML、PDF、または画像を取り込んで埋め込み、類似性検索を介して照会します。
- **AI 開発用の Python SDK** EDB AI Builder Framework SDK を使用すると、開発者は Python を使用して AI ワークフローを組み立てることができます。より少ないコード行で、取得、再ランキング、埋め込み、および生成応答を統合します。

2.6.4 可観測性、モニタリング、インサイト

v1.2 は、拡張されたメトリック、持続可能性の洞察、パフォーマンス診断によりプラットフォームの可観測性を強化します。

- **再デザインされたモニタリングダッシュボード**改良された UI、改善されたレイアウト、およびメトリックとしきい値のより詳細な制御により、クラスターの状態に関するより深い洞察が得られます。
- **クエリレイテンシーメトリック** `pg_stat_monitor` 拡張機能を介して収集されたクエリレイテンシー統計を [**Query Diagnostics**] タブで直接視覚化します。
- **プロアクティブなクエリの推奨事項**実際のワークロードパターンに基づいて、新しいインデックスの提案や古い統計アラートを含む最適化のヒントを受信します。
- **クラススコアカード**各クラスターの診断、推奨事項、およびリスク指標の統合された概要を 1 か所に表示します。
- **自己管理の Postgres サポート**外部 Postgres インスタンスをハイブリッドマネージャーに接続して、クエリ診断、推奨事項、およびモニタリングメトリックを収集します。

2.6.5 サステナビリティ ESG レポートプレビュー

EDB Hybrid Manager には、 **ESG ダッシュボードの実験サポート**が含まれ、組織がデータインフラストラクチャのエネルギー使用量と炭素排出量を追跡するのに役立ちます。

- **カーボンおよびエネルギーメトリック**構成可能な排出係数を使用して、CPU ベースのエネルギー使用量と推定 CO_2 排出量を視覚化します。
- **リージョンアウェアなレポート**展開地域に基づいて、ローカルエネルギーコスト$/kWh と排出率 kg CO_2/kWh を使用してダッシュボードをカスタマイズします。
- **統合された Grafana ダッシュボード** 新しい**電力使用量とカーボンフットプリント**ダッシュボードにコンソールから直接アクセスします。
- **プレビューフェーズの目標** ESG オブザーバビリティの需要を検証し、関心のあるユーザーからフィードバックを収集します。

2.6.6 プラットフォームの改善

Area	Highlights
Deployment options	New Sovereign Data and AI Factory hardware appliance.
Analytics	Lakehouse clusters with Iceberg and Delta Lake query support.
AI application tooling	GenAI Builder, model serving, semantic search, and SDK.
Monitoring	Refreshed console, scorecard view, extended query diagnostics.
Postgres coverage	Supports external/self-managed Postgres instances.
Sustainability	Preview of ESG metrics and dashboards for power and carbon tracking.

2.6.7 リソース

- *Installing Hybrid Manager on AWS EKS*
- *Using Hybrid Manager*
-

AI Factory Overview

- Analytics Lakehouse Architecture
-

Power & ESG Reporting

- Release Notes Archive

リリース日 2025 年 6 月 17 日

第3章　Known issues

これらは、Hybrid Manager 1.3 リリースで特定された現在既知の問題と制限です。必要に応じて、これらの問題の影響を軽減するために役立つ回避策を含めました。これらの問題は積極的に追跡され、将来のリリースで解決される予定です。

3.1 マルチ DC

3.1.1 アップグレード後のマルチ DC 構成の損失

説明マルチ DC セットアップスクリプトクラスター間通信用に適用された構成は、ハイブリッドマネージャープラットフォームのアップグレードまたはオペレーターの調整後、持続しません。

回避策ハイブリッドマネージャーをアップグレードまたはコンポーネントをバージョンアップするたびに、マルチ DC セットアップスクリプトを再度実行して、必要な構成を再適用する必要があります。

3.1.2 マルチ DC セットアップの場所ドロップダウンリストが空です

説明マルチデータセンター環境では、利用可能な場所を取得する API 呼び出しが gRPC メッセージサイズエラー 429 / 4.3MB 制限を超過して失敗します。これは、API 応答に大量の画像セット情報が含まれるため、コンソールに空のロケーションリストが発生するためです。

回避策この高度な回避策には、API から返されるイメージセット情報の量を制限するクラスター管理者権限が必要です。これには、`upm-image-library` および `upm-beacon` ConfigMap のイメージ検出タグルールの変更と、それに続く関連ポッドの再起動が含まれます。

回避策の詳細

回避策は、画像ライブラリとビーコンコンポーネントが使用する正規表現タグルールを変更して、インデックス付けされる画像タグの数を一時的に制限します。これにより、API 応答サイズが削減され、ロケーションをロードできるようになります。

1. `upm-image-library` ConfigMap を見つけます。

```
kubectl get configmaps -n upm-image-library | grep upm-image-library
# Example Output: upm-image-library-ttkt29fmf7 1 5d3h
```

1. 前の手順で見つけた ConfigMap を編集し、各イメージ検出ルール `edb-postgres-advanced` 、`edb-postgres-extended` 、`postgresql` の下の `tags` ルールを変更します。既存の正規表現を制限的な正規表現に置き換えます。

```
# Snippet of the YAML you will edit in the ConfigMap
"imageDiscovery": {
  "rules": {
    "(^|.*/)edb-postgres-advanced$": {
      "readme": "EDB postgres advanced server",
      "tags": [
        "^(?P<major>\d+)\.(?P<minor>\d+)-2509(?P<day>\d{2})(?P<hour>\d{2})(?P<minute>
↪\d{2})    (?:-(?P<pgdFlavor>pgdx|pgds))?(?:-(?P<suffix>full))?$"
      ]
    },
    # ... repeat for edb-postgres-extended and postgresql ...
  }
}
```

!!!note マルチ DC セットアップを実行している場合、この手順はプライマリハイブリッドマネージャークラスターで実行する必要があります。

1. イメージライブラリーポッドを再起動します。

```
kubectl rollout restart deployment upm-image-library -n upm-image-library
```

1. upm-beacon ConfigMap を取得して、エージェント構成を変更します。

```
kubectl get configmaps -n upm-beacon beacon-agent-k8s-config
```

1. ConfigMap beacon-agent-k8s-config を編集し、各 postgres_repositories エントリー edb-postgres-advanced 、edb-postgres-extended 、postgresql の下の tag_regex ルールを変更します。

```
# Snippet of the YAML you will edit in the ConfigMap
postgres_repositories:
  - name: edb-postgres-advanced
    description: EDB postgres advanced server
    tag_regex: "^(?P<major>\d+)\.(?P<minor>\d+)-2509(?P<day>\d{2})(?P<hour>\d{2})(?
 P<minute>\d{2})(?:-(?P<pgdFlavor>pgdx|pgds))?(?:-(?P<suffix>full))?$"
  # ... repeat for edb-postgres-extended and postgresql ...
```

1. エージェントポッドを再起動します。

```
kubectl rollout restart deployment -n upm-beacon upm-beacon-agent-k8s
```

これらの手順を完了すると、画像データサイズが削減されたため、ロケーション API 呼び出しが成功し、ロケーションが Hybrid Manager コンソールに正しく表示されるはずです。

3.2 コアプラットフォームとリソース

3.2.1 upm-beacon-agent 複雑な環境ではメモリ制限が不十分です

説明多くのデータベースとバックアップがある環境では、upm-beacon-agent ポッドのデフォルトの 1GB メモリ割り当てでは不十分であり、頻繁な OOMKill またはクラッシュループの問題が発生する可能性があります。このリソース制限は、現在標準の Helm 値または HybridControlPlane CR を介して構成できません。

回避策ユーザーは Kubernetes 展開に手動でパッチを適用して、upm-beacon-agent ポッドのメモリリソース制限を増やす必要があります。

3.3 データベースクラスターエンジン

3.3.1 EDB Postgres 分散 PGD クラスターに表示される誤ったデータベース名

!!!tip HM バージョン 1.3.1 以降で解決されました。

説明 Hybrid Manager コンソールの **Connect** タブと接続文字列は、PGD クラスターのデフォルトのデータベース名が edb_admin と誤って表示されます。PGD クラスターには、 bdrdb データベースへの接続が必要です。

回避策 PGD クラスター接続情報には、次の信頼できるソースのいずれかを使用します。 .PGPASS BLOB 、.PG_SERVICE.CONF ファイル、またはクラスターの詳細ページの完全な接続文字列。

3.3.2 PGD-X クラスターの作成が「PGD - アプリケーションユーザーの調整」フェーズでスタックする

説明 PGD-X クラスターの作成、特に監視専用リージョンが関係する場合、-グローバル RAFT リーダーシップが予期せず監視専用ノードによって保持されている、または-サブグループの enable_routing が無効になっていることが原因でストールする場合があります。

RAFT リーダーシップ問題の回避策データグループ内のノードへのグローバル RAFT リードの転送を手動でトリガーします。PGD クラスターの bdrdb に接続し、実行します。

```
bdr.raft_leadership_transfer(node_name:=<target node>, wait_for_completion:=true,␣
↪node_group_name:=world);
```

enable_routing 問題の回避策サブグループのルーティングを手動で有効にします。bdrdb に接続して実行します。

```
SELECT bdr.alter_node_group_option(<subgroup name>,enable_routing,true);
```

3.3.3 max_connections がデフォルト以外の場合、3 ノード PGD クラスターの作成に失敗する

説明初期クラスタープロビジョニング中に構成パラメーター max_connections がデフォルト以外の値に設定されている場合、3 データノード EDB Postgres 分散 PGD クラスターの作成は失敗します。

回避策 デフォルトの max_connections 値を使用して PGD 3 データノードクラスターを作成し、クラスターが正常にプロビジョニングされた後に値を更新します。

3.3.4 2 番目のデータグループを作成または複製するときに PGD データベース設定が複製されない

!!!tip HM バージョン 1.3.1 以降で解決されました。

説明 PGD クラスターで 2 番目のデータグループを作成または複製する場合、Postgres 設定 max_connections 、max_worker_processes などは最初のデータグループから自動的にコピーされません。レプリカグループ設定はプライマリグループより低くすることはできないため、これにより、一貫性のない設定とクラスターの状態の問題が発生する可能性があります。

回避策 2 番目の PGD グループの構成を手動で編集して、プロビジョニングの前にデータベース設定が最初のデータグループと同じであることを確認します。

3.3.5 AHA Witness ノードのリソースがオーバープロビジョニングされています

説明 監視ノードを使用する Advanced High AvailabilityAHA クラスターの場合、監視ノードは大規模なデータノードの CPU、メモリ、およびディスク構成を誤って継承し、不必要なリソースのオーバープロビジョニングが発生します。

回避策 pgdgroup YAML 構成を手動で更新して、監視ノードに必要な最小限のリソースを指定および構成します。

3.3.6 HA クラスターは、ストリーミングレプリケーション証明書の認証に verify-full の代わりに verify-ca を使用します

説明レプリカクラスターは、ストリーミングレプリケーション認証に、推奨される最も安全な verify-full の代わりに、それほど厳密ではない verify-ca 設定を使用します。基になる CloudNativePGCNP クラスターが、特定の環境 GKE ロードバランサーなどで verify-full に必要な IP サブジェクト代替名 IP SAN をサポートしていないため、これは現在必要です。

回避策 なし。修正は、IP SAN をサポートする基になる CNP コンポーネントに依存しています。

3.3.7 2 番目のノードが遅すぎるため、大規模な HA クラスターに参加できません

説明大規模なクラスターの場合、2 番目のノードスタンバイが HA クラスターに参加するために使用する pg_basebackup プロセスが遅すぎます。これにより、スタンバイノードの参加に失敗する可能性があり、単一ノードを HA にスケーリングできず、クラスターを HA 構成に直接復元するときに問題が発生します。

回避策単一のノードにデータをロードして HA にスケーリングするベストプラクティスを避けます。代わりに、最初から HA クラスターにデータを直接ロードします。大規模なクラスターを HA 構成に復元するための回避策はありません。

3.3.8 自己管理クラスターのナレッジベースを照会できません

説明 自己管理クラスターのナレッジベースを照会することはできません。

回避策 該当なし

3.4 バックアップ リカバリー

3.4.1 リージョンをまたがってボリュームスナップショットリカバリーを使用する場合、レプリカクラスターの作成が失敗する

!!!tip HM バージョン 1.3.1 以降で解決されました。

説明ボリュームスナップショットリカバリーはクロスリージョンの復元をサポートしていないため、別のリージョンにある 2 番目の場所でのレプリカクラスターの作成は `InvalidSnapshot.NotFound` エラーで失敗します。

回避策最初にプライマリクラスターから Barman バックアップを手動でトリガーし、次にその Barman バックアップをボリュームスナップショットの代わりに使用して、クロスリージョンレプリカクラスターをプロビジョニングします。

3.4.2 デフォルトの並列構成が原因で WAL アーカイブが遅い

説明 `wal.maxParallel` のデフォルト設定は制限が多すぎるため、大量のデータのロード中に WAL アーカイブが遅くなります。これにより、アーカイブの準備ができた WAL ファイルのバックログが発生し、ディスクがフル状態になる可能性があります。このパラメーターは、HM コンソールからはまだ設定できません。

回避策特定のバックアップオブジェクトストアの `objectstores.barmancloud.cnpg.io` Kubernetes リソースを手動で編集し、`wal.maxParallel` 値を 20 などに増やして、アーカイブを高速化します。

3.4.3 ボリュームスナップショットクラスターは WAL ファイルを自動的にプルーニングしません

!!!tip HM バージョン 1.3.4 以降で解決されました。

説明現在の制限のため、自動ログ先行書き込み WAL 保持ポリシーは VolumSnapshot では機能せず、barman-cloud バックアップ方法とのみ互換性があります。VolumeSnapshot は、サポートされているすべての環境のデフォルトのバックアップ方法であるため、VolumeSnapshot を使用して作成されたクラスターは、古い WAL ファイルを自動的にプルーニングせず、オブジェクトストレージに WAL ファイルが蓄積されます。

回避策ストレージの蓄積を防止し、WAL リテンションポリシーを呼び出すには、少なくとも 1 週間に 1 回、VolumeSnapshot で作成したクラスターのオンデマンド Barman バックアップを実行またはスケジュールする必要があります。

3.5 AI ファクトリーとモデル管理

3.5.1 プロファイルキャッシュを使用した nim-nvidia-nvclip モデルの展開の失敗

説明展開プロセス中にプロファイルキャッシュが利用されると、AI ファクトリー内で nim-nvidia-nvclip モデルのモデル作成は失敗します。

回避策回避策では、管理者が NVIDIA レジストリからローカルマシンに必要なモデルプロファイルを手動でダウンロードする必要があります。次に、プロファイルファイルをハイブリッドマネージャーのオブジェクトストレージパスに直接アップロードし、最後に、特定の環境変数を使用して Kubernetes InferenceService YAML にパッチを適用して、失敗したネットワークダウンロードを試行する代わりに事前キャッシュされたファイルを使用するように強制することにより、モデルを展開する必要があります。。

回避策の詳細

1. NGC API キーを使用して、NVIDIA Container Registry nvcr.io にログインします。

```
docker login nvcr.io -u $oauthtoken -p $NGC_API_KEY
```

1. Docker イメージをローカルマシンにプルします。

```
docker pull nvcr.io/nim/nvidia/nvclip:latest
```

1. ダウンロードしたプロファイル用のローカルディレクトリを準備します。

```
mkdir -p ./model-cache
chmod -R a+w ./model-cache
```

1. ターゲット GPU のプロファイルを選択します。

例、A100 GPU プロファイル 9367a7048d21c405768203724f863e116d9aeb71d4847fca004930b9b9584bb6

1. コンテナを実行してプロファイルをダウンロードします。コンテナは CPU 専用モード NIM_CPU_ONLY=1 で実行され、ダウンロードマシンでの GPU 固有の初期化問題が発生します。

```
export NIM_MANIFEST_
↪PROFILE=9367a7048d21c405768203724f863e116d9aeb71d4847fca004930b9b9584bb6
export NIM_CPU_ONLY=1

docker run -v ./model-cache:/opt/nim/.cache -u $(id -u) -e NGC_API_KEY -e NIM_CPU_
↪ONLY -e NIM_MANIFEST_PROFILE --rm nvcr.io/nim/nvidia/nvclip:latest
```

このコンテナは終了しません。ログに「Health Method denied」という行が表示されたら、実行を手動で停止する必要があります Ctrl+C 。これは、プロファイルのダウンロードが完了したことを確認します。

1. ローカルマシンからハイブリッドマネージャー展開で使用されるオブジェクトストレージバケットにプロファイルをアップロードします。

```
gcloud storage cp -r ./model-cache gs://uat-gke-edb-object-storage/model-cache/nim-
↪nvidia-nvclip
```

!!!note gs:// パスを調整して、展開の構成済みオブジェクト保存場所と一致します。

1. HM コンソールを使用してモデル nim-nvidia-nvclip を作成し、Model Profiles Path フィールドを以前の場所/model-cache/nim-nvidia-nvclip などとして指定します。展開は、最初は失敗するか、スタックします。

2. Hybrid Manager Kubernetes クラスターから InferenceService YAML をエクスポートします。

3. 必要な環境変数 NIM_IGNORE_MODEL_DOWNLOAD_FAIL を、エクスポートした YAML の spec.predictor.model ブロックの env セクションに追加します。このフラグは、ローカルで利用可能なキャッシュアップロードしたファイルを使用し、ネットワークダウンロードの失敗を無視するように NIM コンテナに指示します。

```
# --- Snippet of the modified InferenceService YAML ---
spec:
  predictor:
    minReplicas: 1
    model:
      modelFormat:
        name: nim-nvidia-nvclip
      name: ""
      env:
      - name: NIM_IGNORE_MODEL_DOWNLOAD_FAIL  # <-- ADD THIS LINE
        value: "1"                            # <-- ADD THIS LINE
      resources:
        # ... resource requests/limits ...
      runtime: nim-nvidia-nvclip
      storageUri: gs://uat-gke-edb-object-storage/model-cache/nim-nvidia-nvclip
# -------------------------------------------------
```

1. kubectl を使用して変更した YAML を適用して、事前ダウンロードされたプロファイルを使用するように展開を強制します。

```
kubectl apply -f <modified-inference-service-file.yaml> -n <model-cluster-namespace>
```

これで、ポッドは、オブジェクトストレージを介して手動で提供したモデルプロファイルを使用して、正常に起動するようになりました。

3.5.2 モデルプロファイルのオブジェクトストレージパスが空の場合、AI モデルクラスターの展開が停止する

説明 AI モデルクラスターを作成し、[モデルプロファイルパスフィールド] でオブジェクトストレージパスを指定すると、指定されたパスにコンテンツが含まれていない場合、つまりモデルプロファイルにまだ存在します)。

回避策モデルクラスター展開を開始する前に、[モデルプロファイルパスフィールド] で指定されたオブジェクトストレージパスに、正しい有効なプロファイルが含まれていることを確認します。

3.5.3 モデル名が間違っていると、LLM をリモートで呼び出すときに 404 エラーが発生します

!!!tip HM バージョン 1.3.1 以降で解決されました。

説明 API エンドポイントを介して展開された NVIDIA NIM モデルを呼び出すと、モデルカードに表示されるモデル名が API で必要とされる正しい名前ではない場合があります。これにより、「404 Not Found error」が発生します。

回避策 API 呼び出しに必要な正確なモデル名 `nvidia/llama-3.3-nemotron-super-49b-v1` などを見つけるには、最初に`/v1/models` API エンドポイントを照会します。

3.6 HM コンソールと可観測性

3.6.1 アクティブなモデルクラスターのタグはモデルの詳細画面に表示されません

!!!tip HM バージョン 1.3.1 以降で解決されました。

説明 [**モデルクラスターの詳細**] 画面に特定のモデルを利用するアクティブなモデルクラスターを表示するテーブルで、**タグ** フィールドは空です。

回避策 モデルクラスタータグは、専用の **モデルクラスターの詳細**ページで引き続き正しく表示できます。

3.6.2 チャットモデルクラスターメトリックが Grafana モデル概要ダッシュボードにありません

!!!tip HM バージョン 1.3.1 以降で解決されました。

説明展開されたチャットモデルクラスターのメトリックは Grafana モデル概要ダッシュボードに表示されず、これらの特定の AI コンポーネントの可観測性に影響します。

3.6.3 ユーザー作成の Grafana ダッシュボードは、プラットフォームの再展開/アップグレード後に持続しない

説明ユーザーが Grafana アプリケーション内で直接作成したダッシュボードは、永続ストレージに保存されません。これらは、Grafana ポッドが更新、再展開、または再起動されると消えますたとえば、EKS の自動更新またはハイブリッドマネージャーのアップグレード中。

回避策 カスタムダッシュボードは、

exporting the dashboard as JSON によって外部でバックアップする必要があります。アップグレードの後、それらは Grafana に手動でインポートして戻す必要があります。

3.7 マイグレーション

HM 移行ポータル、データ移行ワークフロー、およびスキーマ取り込みワークフローに関連する既知の問題は、 Migrating databases ドキュメント内の専用ページで維持されています。これにより、ドキュメントの関連セクション内で集中的なコミュニケーションが可能になります。完全なリストについては、Known issues, limitations, and notes を参照してください。

第4章　EDB Postgres AI Hybrid Manager

EDB Postgres AI Hybrid ManagerHM は、Kubernetes ベースのインフラストラクチャ全体での Postgres、ソブリン AI、および分析ワークロードのコントロールプレーンです。オンプレミス、クラウド、またはハイブリッド環境に展開するかにかかわらず、PostgreSQL クラスターをプロビジョニング、操作、および監視する一貫した統一方法を提供します。

ハイブリッドマネージャーは、データベースの展開から AI 分析に至るエンドツーエンドのデータワークフローを有効にし、すべてインフラストラクチャ内で管理されます。

4.1 なぜハイブリッドマネージャーを使うのか？

最新のデータプラットフォームでは、データベースのプロビジョニング以上のものが必要です。以下をサポートする必要があります。

- Kubernetes ネイティブのスケーラビリティとオーケストレーション
- 復元力のある高可用性の展開
- AI と GenAI の統合
- オープンフォーマットを使用した Postgres ネイティブ分析
- ガバナンス、可観測性、およびアクセス制御

Hybrid Manager は、これらの要求を満たすために作成されました。ビルトインの Postgres の専門知識で Kubernetes 機能を拡張し、コンテナ化されたインフラストラクチャの柔軟性を犠牲にすることなくステートフルワークロードの操作を簡単にします。

4.2 主要な機能

Area	Description
Database provisioning	コンソール、テンプレート、または宣言的マニフェストを使用して、高可用性の PostgreSQL または分散 Postgres クラスターを展開します。
AI integration	sovereign GenAI pipelines を実行し、KServe でモデルを提供し、インフラストラクチャによって管理される AI ナレッジベースを構築します。
Analytics	コールドデータを Delta Lake or Iceberg にオフロードし、Postgres 構文で照会し、ハイブリッドトランザクション/分析処理を有効にします。
Monitoring and observability	Prometheus、Grafana、Loki を活用する統合ダッシュボードを使用して、クエリの状態、リソースの使用状況、システムアラートに関する詳細な可視性を取得します。

次のページに続く

表 1 – 前のページからの続き

Area	Description
Security and access management	ID プロバイダーと統合し、プロジェクトとクラスターの RBAC を定義し、データベースとサービスへの安全なアクセスを管理します。
Database migration	外部データベース Oracle、自己管理の Postgres から HM 管理クラスターにスキーマとデータを移行します。移行とスキーマ評価、データ転送スナップショット、および継続的レプリケーションストリーミングのための統合サービスを活用して、最小限のダウンタイムの移行を実現します。

4.3 サポートされている環境

Hybrid Manager は Kubernetes で実行され、次の一般的なエンタープライズクラウドプラットフォームをサポートします。

- Amazon Elastic Kubernetes Service EKS
- Google Kubernetes EngineGKE

HM は、オンプレミスホスティングプラットフォームもサポートしています。

- Red Hat OpenShift RHOS
- SUSE Rancher RKE2

クラウドネイティブとオンプレミスの両方のアーキテクチャをサポートし、主権、コンプライアンス、柔軟なデータインフラストラクチャを構築できます。

4.4 統合ソリューション

Hybrid Manager は、EDB Postgres AI エコシステム全体のコア機能のロックを解除します。それは次のもので構成されています。

- *AI Factory Architecture on Hybrid Manager* – プライベートデータを使用して GenAI ワークロードをビルドおよび実行します。
- *Analytics Concepts in Hybrid Manager* – Iceberg、Delta Lake、および階層化オフロードを使用してオープンフォーマット分析を有効にします。
- **モデルイメージの移行** – プラットフォーム内の移行ツールを使用して Postgres の採用を加速します。
- **オペレーションモニタリング** – Kubernetes とデータベースの洞察を備えたビルトインの可観測性スタックを使用して、オペレーションを監視します。
- Cluster lifecycle – テンプレートの作成、アップグレードの実行、HA 設定の構成、およびデータの復元。

4.5 詳細はこちら

- *AI Factory Architecture on Hybrid Manager*
- *Supported Postgres distributions*

- *Supported database types*

4.6 はじめる

- *Install Hybrid Manager on AWS EKS*
- Use Hybrid Manager

4.6.1 Hybrid Manager architecture

図の最も外側のコンテナは、EKS on AWS や GKE on GCP などのクラウドサービスプロバイダーのインフラストラクチャ、またはオンプレミスインフラストラクチャで実行されている Kubernetes クラスターを示しています。この文脈での「オンプレミス」には、組織のインフラストラクチャ内でホストされる物理サーバーベアメタル、仮想マシン、またはプライベートクラウドでの展開が含まれます。

図の内部に向かって作業すると、3 つの主な論理グループは次のとおりです。

1. **Kubernetes コントロールプレーン** Kubernetes コントロールプレーンは、コア Kubernetes コンポーネントで構成され、2 つまたは 3 つの Kubernetes 制御ノードによって実装されます。コアコンポーネントの例は、`kube-apiserver`、`etcd`、`kube-scheduler`、`kube-controller-manager`、およびクラウド設定の場合は `cloud-controller-manager` です。

2. **制御コンピューティンググループ** HM の制御コンピューティンググループは、2 または 3 つの Kubernetes ワーカーノードで実行されている 70 以上のポッドの論理グループです。Grafana、Loki、Thanos、オブザーバビリティの Prometheus、証明書管理の Cert Manager、ネットワーキングの Istio、ソフトウェアリリースを保護するためのトラストマネージャーなど、HM のコア管理コンポーネントを実装しています。

コンポーネントの 70 以上のポッドは、`kube-scheduler` がそれらを配置する方法に従って、ワーカーノード全体に分散されます。これは、図に示されているほど単純ではありません。

必要に応じて、ワーカーノードを追加して、より多くの Database-as-a-Service リソースをサポートできます。

3. **データコンピューティンググループ** HM のデータコンピューティンググループは、名前空間によって編成された Postgres クラスターの論理グループです。`kube-scheduler` が配置すると、少なくとも 3 つの Kubernetes ワーカーノードに分散された多数のポッドで実行されますが、図ほど単純ではない場合があります。

HM を使用してデータベースが追加されると、ワーカーノードの数を 3 つから増やすことができます。ただし、これを自動的に行うには、ワーカーノードを手動で追加するか、インフラストラクチャとともに自動スケーリングを構成する必要がある場合があります。

制御コンピューティンググループのワーカーノードとデータコンピューティンググループのワーカーノードは両方とも、スナップショット SAN を使用するオンプレミスまたは EKS で EBS を使用するオンプレミスまたは Barman for the cloud を使用する S3 互換オブジェクトストアにバックアップされます。

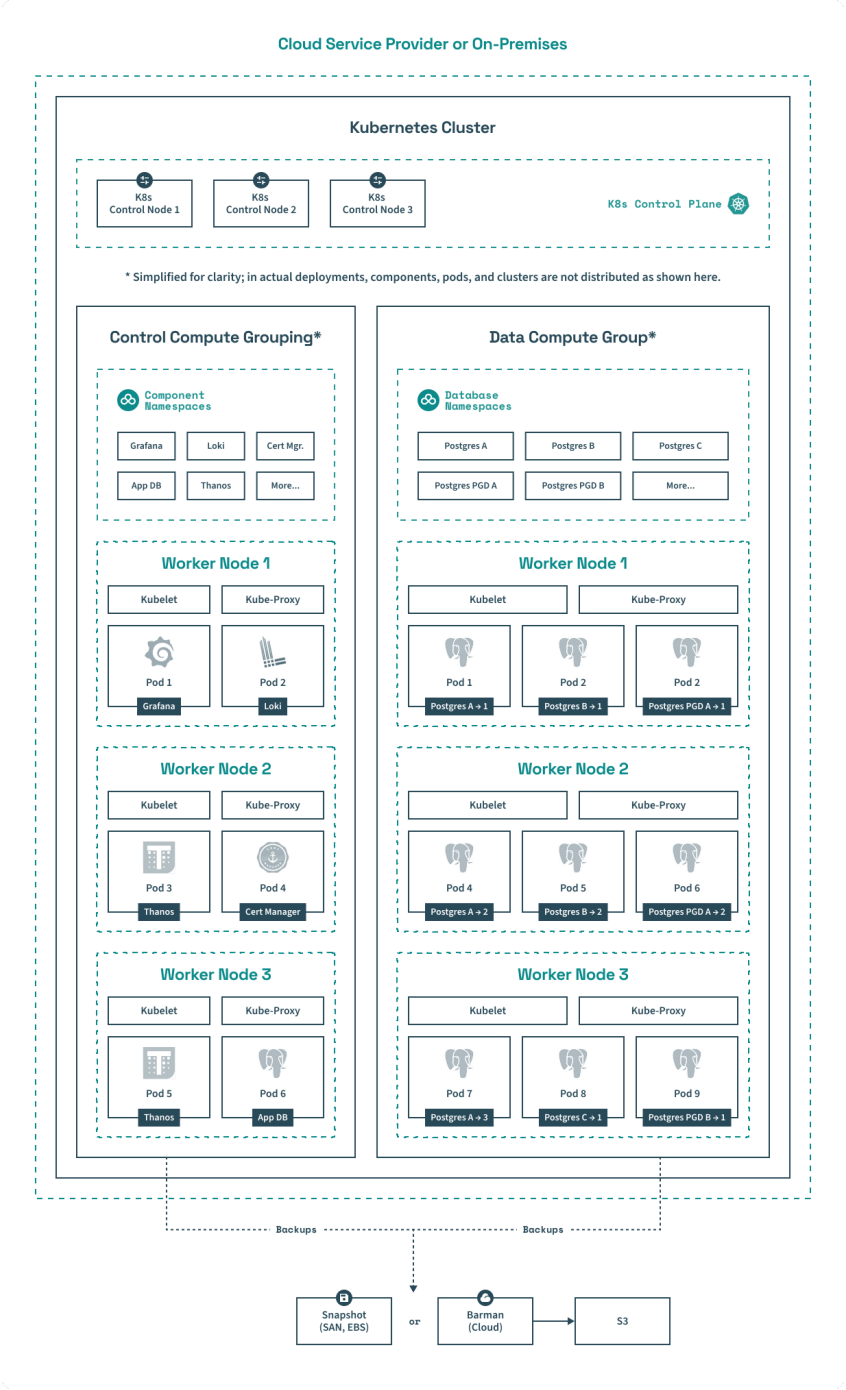


図 1: High-level diagram of Hybrid Manager’s architecture

スケーラビリティ

スケーラビリティをサポートするために、HM は Kubernetes の自動スケーリング機能と統合しています。これは、AWS のようなプラットフォームで、データコンピューティンググループのワーカーノードを、AWS のマネージドサービス Auto Scaling groups を使用して、必要に応じてクラスターに自動的に追加できることを意味します。ただし、HM の Kubernetes 自動スケーリング機能を無効にして、コストを制御できます。さらに、オンプレミス展開の自動スケーリング機能を有効にするには、追加のインフラストラクチャ構成が必要であることに注意してください。これは、AWS の Auto Scaling グループの場合と同様に、オンプレミス展開にはすぐに使える自動スケーリング機能がないためです。

included Grafana dashboards または using Kubernetes `kubectl commands <Tracing clusters back to their Kubernetes resources>` を使用して、データベースクラスターを実装するポッドとワーカーノードの分散とリソース使用量をモニタできます。

バックアップアーキテクチャ

HM は、データの完全性と復旧可能性を保証するための堅牢なバックアップオプションを提供します。データコンピューティンググループ内の各 HM ワーカーノードは、デフォルトで夜間バックアップを実行し、Barman クラウド用またはスナップショットバックアップを使用して保存されます。

スナップショットバックアップは、復旧時間を短縮するために利用できますが、ストレージレイヤーに互換性のある CSI ドライバーが必要ですクラウドベースのスナップショットとローカルディスクスナップショットの両方。論理ボリュームマネージャー LVM セットアップの制約を考慮すると、スナップショットはローカルリカバリに最適ですが、サイト間の共有アクセスが欠如しているため、マルチサイトまたはグローバルリカバリシナリオのセットアップで使用する場合、制限に直面する場合があります。

より堅牢なマルチアベイラビリティーゾーンのリカバリーのために、Barman クラウドバックアップは、マルチゾーンレプリケーションをネイティブにサポートする AWS S3 のようなオブジェクトストアに保存できます。または、ローカルオブジェクトストアでは、バックアップがリモートサイトにレプリケートされるように追加の構成が必要になる場合があります。ストレッチクラスターは、地理的に分散した場所全体でデータのレプリケーションと冗長性を有効にすることにより、これらの制限を緩和します。

高可用性と復元力

HM は、Kubernetes の実績のあるテクノロジーを活用して、自動フェイルオーバーメカニズムを介して高可用性を保証します。これらのメカニズムは、ストレッチクラスター内、アベイラビリティーゾーン全体、または同じリージョン内のゾーン間で機能します。このアーキテクチャは、スナップショットバックアップを介したより高速なバックアップと、大規模なデータセットのリカバリをサポートしていますが、マルチサイトのリカバリ機能は、基になるストレージインフラストラクチャに依存します。複数のラックまたはデータセンターにまたがるストレージエリアネットワーク SAN などの高度なストレージ構成を使用しているお客様の場合、HM はインフラストラクチャを活用して復元力を強化し、迅速なフェイルオーバーを保証できます。ただし、これらの構成はすべてのユーザーで共通ではない場合があります。

4.6.2 Supported database types

Hybrid ManagerHM で新しいクラスターを作成するときに適切な Postgres データベースクラスタータイプを選択することは、データベース環境を最適化するために重要です。EDB は、基本的なシングルノード展開から復元力の高い分散ソリューションに至るまで、さまざまなニーズに合わせて調整された幅広いソリューションを提供します。

分散高可用性

分散高可用性 DHA クラスターは、 EDB Postgres 分散 PGD を搭載し、マルチマスター論理レプリケーションを活用します。このアーキテクチャにより、マルチプルのリージョンにわたって真のアクティブ/アクティブソリューションが可能になり、最高レベルの可用性と復元力を提供します。

次の場合にこのオプションを選択します。

- 組織は、超高可用性とほぼゼロのダウンタイムを必要としています。
- アプリケーションは、ユーザーの低遅延アクセスを備えた分散展開を必要とします。
- ビジネスには、マルチプルのアクティブノードにわたる継続的なオペレーションとデータの一貫性が最重要であるミッションクリティカルなアプリケーションがあります。
- シナリオには、可用性に影響を与えることなく、中断のないオンラインデータベースのメンテナンスとインプレースメジャーバージョンのアップグレードが必要です。

高度な高可用性

高度な高可用性 AHA は、基本的なプライマリ/スタンバイセットアップを超えた堅牢な HA ソリューションを提供することに焦点を当てており、より高いアップタイム保証とより洗練されたリカバリメカニズムを目指しています。これには、さまざまな強化された HA 構成が包含され、多くの場合 EDB の専用ツールと専門知識を使用して、優れた復元力を提供します。DHA クラスターとは異なり、AHA クラスターは単一のリージョンを使用し、そのリージョン内で高可用性を提供するように設計されています。

次の場合にこのオプションを選択します。

- チームは、高度な EDB Postgres 分散機能を活用し、将来の完全な DHA への移行を可能にする DHA よりもシンプルなセットアップを好みます。
- ビジネスでは、複雑さと高いアップタイムのバランスが必要で、完全なマルチマスター分散アーキテクチャなしでより強力な復元力を提供します。
- 企業は、標準のプライマリ/スタンバイ構成を超える高可用性の強化が必要です。
- 組織は、より柔軟な展開オプションを使用して、計画的なメンテナンスと計画外の停止のダウンタイムを最小限に抑えようとしています。

高可用性

高可用性 HA プライマリ/スタンバイ構成には、通常、プライマリデータベースサーバーと 1 つ以上のスタンバイレプリカサーバーが関係します。スタンバイサーバーは、プライマリからデータの変更を継続的に受信し、プライマリに障害が発生した場合に引き継ぎます。このセットアップでは、物理的ストリーミングレプリケーションを使用します。

次の場合にこのオプションを選択します。

- アプリケーションはデータの耐久性と最小限のダウンタイムでの継続的なサービスを必要とし、厳密なアクティブ-アクティブマルチマスターは要件ではありません。
- ユースケースには、読み取り中心のワークロードがあり、クエリをホットスタンバイレプリカにオフロードすることでメリットを得ることができます。

単一ノード

シングルノード展開は、スタンバイまたはレプリカサーバーなしで実行される単一の PostgreSQL データベースインスタンスで構成されます。これは最も単純な構成であり、クリティカルでないワークロードまたは開発環境に適しています。

このオプションは次の場合に選択します。

- 高可用性が懸念されない開発およびテスト環境。
- メンテナンスまたはアップグレード、または障害の場合のダウンタイムを許容できる非クリティカルアプリケーション。
- 何よりもシンプルさと最小限のインフラストラクチャコストを優先するユースケース。
- トラフィックとデータ量の少ない小規模なアプリケーション。

クラスタータイプのアーキテクチャ

- *Distributed high availability*
- *Advanced high availability*
- **高可用性の考慮事項**
- **単一ノード**

Distributed high availability

分散高可用性クラスターは、 EDB Postgres Distributed を搭載しています。マルチマスター論理レプリケーションを使用して、物理レプリケーションベースのシステムよりも高度なクラスター管理を提供します。分散高可用性クラスターを使用すると、複数のリージョンまたは単一のリージョンにクラスターを展開できます。リージョン全体の高可用性が要件であるユースケースの場合、分散高可用性を備えたクラスター展開は、2 つの異なるリージョンに最大 2 つのデータグループと、3 番目の別のリージョンの監視グループを提供できます。

この構成は、各データグループが書き込み操作を受け入れるように構成されているため、真のアクティブ-アクティブソリューションを提供します。

分散高可用性クラスターは、EDB Postgres Advanced Server と EDB Postgres Extended Server データベースディストリビューションの両方をサポートしています。

分散高可用性クラスターには、1 つまたは 2 つのデータグループが含まれます。データグループには、3 つのデータノード、または 2 つのデータノードと 1 つの監視ノードを含めることができます。任意の時点で、各グループのこれらのデータノードの 1 つがリーダーであり、書き込みを受け入れますが、残りは

shadow nodes と呼ばれます。非同期 commit scopes を使用する場合を除き、本番では 2 つのデータノードと 1 つの監視ノードを使用しないことをお勧めします。

PGD Connection Manager は、すべてのアプリケーショントラフィックを書き込みリーダーノードにルーティングできます。各グループには独自の書き込みリーダーがあり、分散データの競合が削減されます。PGD は、分散コンセンサスモデルを活用して、クラスター内のデータノードの可用性を判断します。リーダーが失敗または使用不可になると、PGD は新しい書き込みリーダーを選択し、アプリケーショントラフィックをリダイレクトします。EDB Postgres 分散のコア機能と合わせて、アプリケーショントラフィッ

クをリーダーノードにルーティングするこのメカニズムにより、高速なフェイルオーバーとスイッチオーバーが可能になります。

監視ノード/Witness グループはデータをホストしませんが、管理のために存在します。リージョンに障害が発生した場合など、コンセンサスを必要とする操作をサポートします。

!!!note 分散高可用性クラスターに対する操作は、計画的なライフサイクル操作中に 1 秒未満の中断を提供する EDB Postgres Distributed set-leader 機能を活用します。

単一のデータの場所

単一のデータの場所を持つ構成には、1 つのデータグループと次のいずれかがあります。

- 1 つのリード、1 つのシャドウ、および監視ノードを備えた 2 つのデータノード、それぞれが別のアベイラビリティーゾーンにある

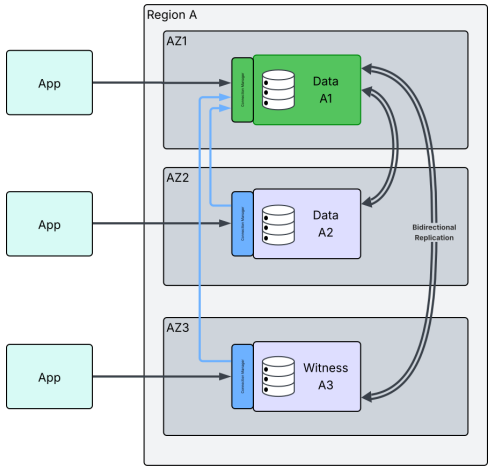


図 2: region(2 data + 1 witness)

- 1 つのリードノードと 2 つのシャドウノードを備えた 3 つのデータノード、それぞれが別のアベイラビリティーゾーンにある

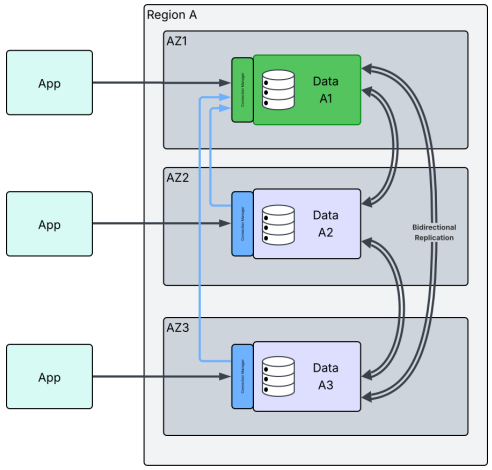


図 3: region(3 data)

監視ノードを使用する複数のグループ

複数のデータの場所を持つ構成には、各グループに 3 つのデータノードを含む 2 つのデータグループがあります。-1 つのグループ内のデータノードと 2 つのシャドウノード-別のグループの同じ構成-3 番目のグループの監視ノード

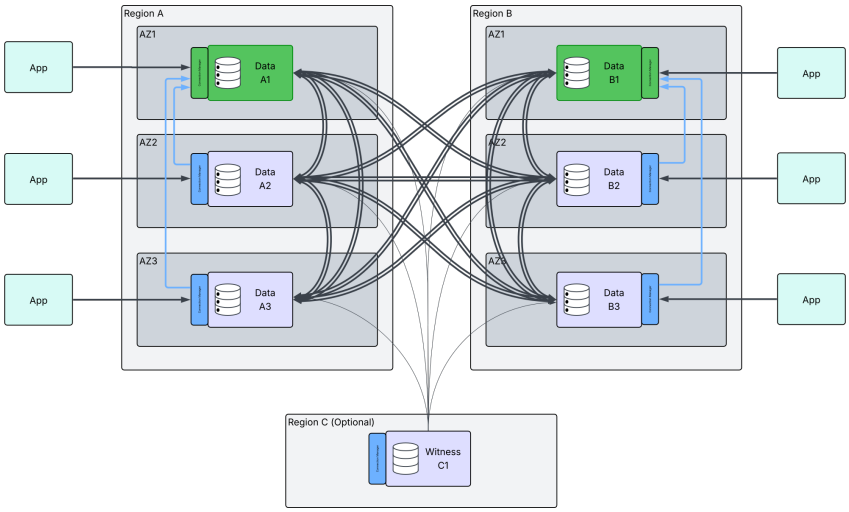


図 4: groups(2 x 3 data + 1 witness)

グループは現在同じリージョンクラスターが存在するリージョンにありますが、これは将来的に変更される可能性があります。

アドバイザリーロック

アドバイザリロックは Postgres ノード間でレプリケートされないため、シャドウノードで取得されたアドバイザリロックは、リーダーで取得されたアドバイザリロックと競合しません。アドバイザリーロックに依存するアプリケーションは、これらのトランザクションで読み取り専用のワークロードを使用しないことをお勧めします。

詳細については

HM CLI を使用して分散高可用性クラスターを作成、取得、および管理する手順については、Using the HM CLI を参照してください。

Advanced high availability

Advanced 高可用性 AHA クラスターは EDB Postgres Distributed を搭載し、PGD の Essential エディションをベースとしています。*Distributed high availability* とは異なり、高度な高可用性クラスターは単一のリージョンを使用し、そのリージョンで高可用性を提供するように設計されています。

これらのクラスターは、分散高可用性クラスターよりも簡単なセットアップと少ない構成オプションを必要とするユーザー向けに設計されています。同時に、これらのユーザーは、EDB Postgres Distributed の高度な機能と、将来的に分散高可用性クラスターに簡単に移行できる機能の恩恵を受けます。

AHA クラスターは、マルチリージョン展開の複雑さなしで堅牢なフェイルオーバー機能を必要とするアプリケーションに最適です。高度な高可用性クラスターの基になるアーキテクチャは、分散高可用性クラスターのアーキテクチャと似ていますが、ユーザーが PGD 接続マネージャーすべてのノードに埋め込みを

介して接続して、接続が現在の書き込みリーダーノードに送られるようにする必要があります。これにより、クラスター全体の競合を管理する必要がなくなり、アプリケーション開発が簡単になります。

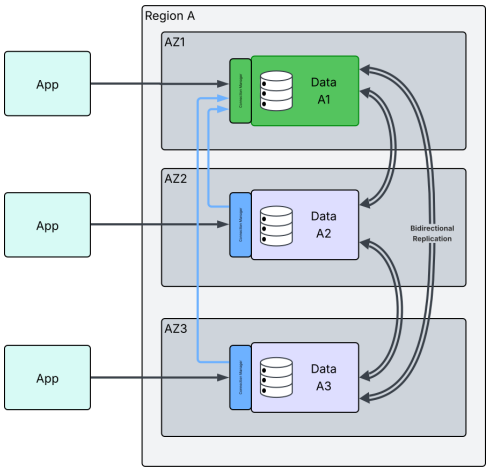


図 5: Advanced high-availability cluster architecture

この図は、接続マネージャーポートを介して PGD クラスターに接続するアプリケーションを示しています。接続マネージャーは、読み取りおよび書き込みトランザクションをグループ内の適切なノードにルーティングします。書き込みリーダーは、すべての書き込みトランザクションを処理する責任を負い、AZ1 の上部に緑色で表示されます。

グループ内の他のノードは、書き込みリーダーからデータを複製する読み取り専用ノードです。読み取り専用ノードの接続マネージャー読み取り/書き込みポートに接続しているアプリケーションは、クエリと変更が書き込みリーダーにルーティングされます。常に、ノードは互いに通信し、データを複製して同期していることを確認します。

AHA クラスターは、EDB Postgres Advanced Server と EDB Postgres Extended Server データベースディストリビューションの両方をサポートしています。コミュニティ PostgreSQL ディストリビューションはサポートされていません。

High availability

プライマリ/スタンバイの高可用性により、障害が発生した場合のダウンタイムを最小限に抑えます。プライマリ/スタンバイの高可用性クラスター 1 つの**プライマリ** と 1 つまたは 2 つの **スタンバイレプリカ**は、自動的に構成されます。スタンバイレプリカは、物理的ストリーミングレプリケーションを介して最新の状態に保たれます。

読み取り専用ワークロードが有効になっている場合、スタンバイレプリカは読み取り専用ワークロードを提供します。2 ノードクラスターでは、単一のスタンバイレプリカが読み取り専用のワークロードを処理します。3 ノードクラスターでは、両方のスタンバイレプリカが読み取り専用のワークロードを処理します。接続は、2 つのスタンバイレプリカへの接続ごとにランダムに作成されます。

アベイラビリティーゾーンを備えたクラウドリージョンでは、クラスターがゾーンを超えてプロビジョニングされ、データセンターに障害が発生した場合のフォールトトレランスを提供します。

プライマリが一時的または永続的に使用不可になった場合、スタンバイレプリカがプライマリになります。

着信クライアント接続は、常に現在のプライマリにルーティングされます。プライマリに障害が発生した場合、スタンバイレプリカがプライマリに昇格し、新しい接続が新しいプライマリにルーティングされま

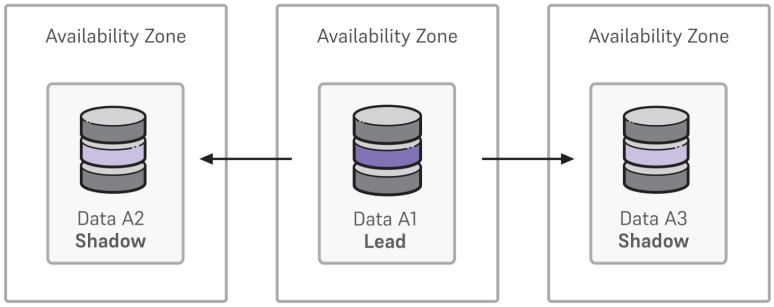


図 6: Primary/standby high availability

す。古いプライマリが復旧すると、スタンバイレプリカとしてクラスターに再参加します。

スタンバイレプリカ

デフォルトでは、レプリケーションは 1 つのスタンバイレプリカと同期し、もう 1 つとは非同期です。つまり、1 つのスタンバイレプリカは、クライアントが成功したコミットの確認を受信する前に、トランザクションレコードがディスクに書き込まれたことを確認する必要があります。

1 つのプライマリと 1 つのレプリカを持つクラスター 2 ノードプライマリ/スタンバイ高可用性クラスターでは、3 ノードと同じレベルの信頼性がないため、クラスターが書き込みに使用できなくなるリスクがあります。クラスター。ハイブリッドマネージャーは、2 ノードクラスターのメンテナンス操作中に同期レプリケーションを無効にして、書き込みの可用性を保証します。2 ノードクラスターのデフォルトの同期レプリケーションから、セッションごとまたはトランザクションごとに非同期レプリケーションに変更することもできます。

PostgreSQL では、`synchronous_commit` は `on` に設定され、`synchronous_standby_names` は `ANY 1 (replica-1, replica-2)` に設定されます。 `SET` または `ALTER` コマンドを使用して、トランザクションごと、セッションごと、ユーザーごと、またはデータベースごとにこの動作を変更できます。

ハイブリッドマネージャーは 1 つのノードにのみ同期してレプリケートするため、3 ノードクラスターの一部のスタンバイレプリカではレプリケーションラグが発生する場合があります。また、ハイブリッドマネージャーの同期レプリケーション構成をオーバーライドすると、スタンバイレプリカに一貫性がなくなります。

Single-node cluster

高可用性が主な関心事ではない非運用ユースケースの場合、高可用性を有効にしていないクラスター展開は、フェールオーバーまたは読み取り専用ワークロードのスタンバイレプリカのないプライマリを 1 つ提供します。

プライマリに回復不能な障害が発生した場合、バックアップからの復元が必要です。



図 7: Single node cluster

4.6.3 Supported Postgres distributions

Hybrid ManagerHM で新しいデータベースクラスターを作成するときに適切な Postgres ディストリビューションを選択することは、データベース環境を最適化するために重要です。EDB は、オープンソースの柔軟性からエンタープライズグレードの機能、Oracle 互換性まで、さまざまなニーズに合わせた幅広いオプションを提供します。

EDB Postgres Advanced Server

EDB Postgres Advanced Server は、EDB で最も包括的な PostgreSQL ディストリビューションです。advanced security 、advanced SQL 、高可用性、マルチマスターレプリケーションなどの機能をサポートします EDB Postgres Distributed およびオプションで Oracle compatibility 。要求の高いエンタープライズワークロードと、コストのかかる Oracle データベースからのシームレスな移行に最適です。

このオプションは次の場合に適しています。

- 移行コストと複雑さを削減するための高い互換性を求める Oracle データベースから移行する企業。
- 高度なセキュリティ、パフォーマンス、および管理ツールを必要とするミッションクリティカルなアプリケーションを持つ組織。
- 強化された監査、SQL インジェクション保護、堅牢な高可用性を含む包括的なエンタープライズグレードの機能を必要とするビジネス。

EDB Postgres Extended Server

EDB Postgres Extended Server は、エンタープライズ要件、特にデータセキュリティにおいて設計された特定の機能強化を提供します transparent data encryption などを介して。また、EDB の強力な高可用性マルチマスターレプリケーションソリューション EDB Postgres Distributed を使用した高度なレプリケーションもサポートしています。

このオプションは次の場合に適しています。

- 強化されたデータセキュリティ機能を必要とする企業。
- EDB の高可用性およびマルチマスターレプリケーションソリューションを活用しようとしている組織。
- コミュニティ PostgreSQL との強い互換性を維持しながら、追加のエンタープライズ機能を備えた PostgreSQL エクスペリエンスを希望するユーザー。

PostgreSQL

オープンソースの PostgreSQL は、強力で信頼性の高い機能が豊富なリレーショナルデータベースシステムです。SQL 標準への強い準拠、堅牢なアーキテクチャ、データの完全性、およびグローバルコミュニティで知られています。

このオプションは次の場合に適しています。

- 費用対効果とオープンソースの自由を優先する組織。
- 高度なセキュリティ機能、マルチマスターレプリケーションソリューション、または Oracle の互換性を必要としないプロジェクト。

- 社内の PostgreSQL の専門知識とリソースとチームして、データベース環境を管理します。

ディストリビューションの比較

機能および機能ごとの使用可能なディストリビューションの比較については、以下を参照してください。

- *Feature support per distribution*
- *Advanced replication*
- *Advanced security*
- *Advanced SQL*
- *Advanced performance*
- *Oracle support*
- *Tools and extensions*
- **展開オプション**

ディストリビューション別のドキュメント

Postgres ディストリビューションを最大限に活用するには、そのドキュメントを参照してください。

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

第5章　Installing Hybrid Manager

Hybrid ManagerHM は、クラウドおよびオンプレミスの展開全体で Postgres、分析、および AI ワークロードのオーケストレーション、オブザーバビリティ、およびライフサイクルオペレーションを提供します。

このセクションでは、サポートされている Kubernetes プラットフォームにわたって、独自のインフラストラクチャに HM を展開するユーザーを対象にインストールガイダンスを提供します。ソブリンデータと *AI* ファクトリーハードウェアアプライアンス のようなターンキーハードウェアシステムを使用している場合、Hybrid Manager はプリインストールされており、セットアップは必要ありません。

5.1 インストールする前に

HM では、ストレージ、ネットワーキング、ノードプールを管理するために Kubernetes 環境とプラットフォーム固有の API へのアクセスが必要です。

HM は次の場所にインストールできます。

- RedHat OpenShift クラスター
- Rancher RKE2 クラスター
- クラウド管理 Kubernetes EKS、GKE

特定のプラットフォームにインストールする前に、次のことを確認してください。

- システムリソースとストレージ要件を確認します
- 組織のアクセスと RBAC 要件を定義する

5.2 プラットフォーム別のインストールガイド

ターゲットプラットフォームを選択して、適切なインストール手順に従います。

5.2.1 RedHat OpenShift RHOS

HM は、OpenShift クラスターへの展開について認定されています。オンプレミスまたは Red Hat OpenShift on AWSROSA の両方。

- Install HM on RHOS

5.2.2 Rancher RKE2

HM は、オンプレミスの Rancher RKE クラスターでの展開でサポートされています。

- Install HM on Rancher RKE2

5.2.3 アマゾン ウェブ サービス EKS

Amazon Elastic Kubernetes Service EKS に HM をインストールします。

- Install HM on EKS

5.2.4 Google Cloud PlatformGKE

GKE 管理の Kubernetes クラスターに HM をインストールします。

- Install HM on GKE

5.3 プリインストールされたハードウェアソリューションを検討していますか?

HM とそのサポートインフラストラクチャのフルマネージド展開を探している場合は、EDB が提供するターンキーハードウェアソリューションである ソブリンデータと *AI* ファクトリーハードウェアアプライアンス を検討してください。

このシステムには以下が含まれます。

- ハイブリッドマネージャーと Redhat Openshift のプリインストールされたエンドツーエンドのサポート
- ハードウェアとソフトウェアの完全なライフサイクル管理
- Postgres、ベクターワークロード、および GPU 駆動の推論のビルトインサポート

詳細については、 Engineered Systems overview をご覧ください。 —

5.3.1 Hybrid Manager installation methods for Rancher RKE2

Red Hat Open Shift installation prerequisites

Red Hat Open SHIft には、 EDB Postgres AI Hybrid ManagerHM をインストールする前に満たす前提条件があります。インストールプロセスを開始する前に、前提条件を満たす必要があることを確認します。

前提条件

- *A system from which you can run the installation process*
- *A RHOS cluster configured for use with HM*
- *Necessary environment variables set*
- *Sync container images to your registry*

次のステップ

これらの要件を満たしたら、*Installation phase of Hybrid Manager on Rancher RKE2* に進むことができます。

Install system

インストールプロセスを実行できるシステムが必要です。このシステムには、次のソフトウェアをインストールする必要があります。

-

OpenShift CLI (oc)

- Kubernetes クラスターを管理するための kubectl
- yq Hybrid Manager インストールファイルの準備

リンクされた手順、または以下にあるオペレーティングシステムの手順を使用して、これらのツールをインストールできます。

macOS

```
brew install kubectl yq
```

Linux

```
sudo apt install kubectl
sudo snap install yq
```

```
sudo yum install kubectl
sudo wget https://github.com/mikefarah/yq/releases/latest/download/yq_linux_amd64 -O /
↪usr/bin/yq && sudo chmod +x /usr/bin/yq
```

```
sudo apt-get install kubectl
sudo wget https://github.com/mikefarah/yq/releases/latest/download/yq_linux_amd64 -O /
↪usr/bin/yq && sudo chmod +x /usr/bin/yq
```

Windows

```
winget install kubectl --source winget
winget install yq --source winget
```

1. kubectl をインストールします。
2. yq をインストールします。

次のステップ

必要なソフトウェアをインストールしたら、 *setting up aa RHOS cluster* に進むことができます。

Creating an Red Hat OpenShift cluster for use with Hybrid Manager

installed your system が完了したら、Hybrid ManagerHM で使用する Red Hat OpenShiftRHOS クラスターを作成する準備が整います。

HM インストールのための OpenShift クラスターの準備

Red Hat OpenShiftRHOS オンプレミスクラスターに HM を正常にインストールするには、特定の前提条件と構成を満たしている必要があります。

事前構成のクラスター要件

- HM は現在サポートしています
    - OpenShift 4.17 with Kubernetes 1.30

or:

- OpenShift 4.18 with Kubernetes 1.31

ステップ 1 ネットワークを構成する

- ネットワークポリシーを定義します。Kubernetes コントロールプレーン、ワーカーノード、外部システム間のトラフィックを管理するために、適切なネットワークポリシーが設定されていることを確認します。
- イングレス/エグレスルールを定義します。
- クラスターコンポーネントと HM スタック間の通信が安全であることを確認します。- 入力/出力ネットワークポリシーを検証して、不要な外部アクセスを制限しながら、クラスターコンポーネントが通信できるようにします。
- ロードバランサーのセットアップ外部ロードバランサーを使用する場合、OpenShift API トラフィックとワーカーノード通信を処理するように構成します。
- DNS の構成 HM の内部サービス検出を含む、クラスターに DNS レコードが設定されていることを確認します。
- クラスターにワーカーノード用の適切なプライベートネットワーク分離があることを確認します。OpenShift は通常、ソフトウェア定義のネットワーク SDN 構成を介してこれを処理します。

ステップ 2 ノードサイズとリソースを構成する

HM とその関連コンポーネントをサポートするには、OpenShift クラスター内のノードがリソース要件を満たしている必要があります。

ノードのサイジング

- **マスターノード** OpenShift コントロールプレーンノードは、リソース割り当ての標準要件 CPU、メモリ、ディスクサイズなどを満たしている必要があります。
  - 一般的な推奨事項
    - 3 つのマスターノードそれぞれに次のものが含まれます。
      - CPU
        - 最小値 4 つの vCPU 最大 10 のワーカーノードの場合
        - 推奨 中規模のクラスター 10〜50 のワーカーノード用の 8 vCPU
        - 50 ワーカーノードの場合 16+ vCP Us
      - メモリ
        - 最小 32 GB RAM 最大 10 の場合ワーカーノード)
        - 推奨 大規模なワークロード 10〜50 ワーカーノードの場合は 64 GB RAM 以上
        - 50 ワーカーノードの場合 64+ GB RAM
      - Disk
        - 最小 100 GB SSD
        - 推奨 大規模なワークロードの場合は 200 GB SSDs
        - 50 ワーカーの場合ノード >200 GB SSD
- **ワーカーノード** HM 制御コンポーネントと Postgres クラスターを処理するために十分なリソースをワーカーノードに割り当てます。小規模なワークロードには、HM 制御コンポーネント用に 3 つのワーカーノードと Postgres ワークロード用の 3 つのワーカーノードをお勧めします。5〜20 の Postgres クラスター)。ここから Postgres ワークロードをスケールアップすると、Postgres ワークロード用により多くのワーカーノードが必要になり、最終的には Postgres ワークロードワーカーノードの数の増加をサポートするためにさらに多くの制御ワーカーノードが必要になる場合があります。
  - 一般的な推奨事項
  - 6 つのワーカーノードそれぞれ:
    - CPU :
      - 最小 4 vCPU これは、ワークロードの種類によっては高くなる場合があります
    - メモリ
      - 最小 ノードごとに 32 GB の RAM
    - ディスク
      - 最小 ノードごとに 100 GB の永続ストレージデータベースとロギングの要件に基づいて調整します)。最適なパフォーマンスを得るには、高速ディスク SSD を使用します。

HM 制御ノードと Postgres ワークロードをホストする HM Postgres データノードに異なるタイプのノードをオプションで使用するには、RHOS に追加のマシンセットが必要です。

- 制御ノードマシンセット

```
spec:
    replicas: 3
    template:
    spec:
        metadata:
        labels:
            edbaiplatform.io/control-plane: "true"
    taints:
    - key: edbplatform.io/control-plane
        value: "true
        effect: NoSchedule
```

- Postgres ワークロードマシンセットをホスティングするデータノード

```
spec:
    replicas: 3
    template:
    spec:
        metadata:
        labels:
            edbaiplatform.io/control-plane: “true”
        taints:
        - key: edbplatform.io/control-plane
            value: "true"
            effect: NoSchedule
```

リソース予約

- HM ワークロードとの競合を防ぐには、システムコンポーネント用に CPU とメモリを予約します。
- HM モニタリングおよび可観測性ツールたとえば、Grafana、Loki、および Prometheus に追加リソースを割り当てます。
- ノードプール構成
    - 必要に応じて、テイントと寛容を使用して、特定のノードで HM ワークロードを分離します。
    - 特定のタスクたとえば、「モニタリング」、「データベース」のノードにラベルを付けて、リソース割り当てを最適化します。
- スケーリング
- 自動スケーリングを設定して、クラスターが需要の増加を動的に処理できるようにします。OpenShift のホリゾンタルポッドオートスケーラー HPA を使用して、必要に応じてポッドをスケーリングできます。

ネットワーキング帯域幅

各ノードに、HM 通信と外部データ転送 S3 バックアップなどを処理するための十分なネットワーク容量があることを確認します。

ベースラインの推奨事項

Component	Recommended bandwidth	Justification
K8s control plane nodes	1 Gbps+ per node	Handles internal Kubernetes traffic, API server requests, logs/metrics, and orchestration tasks.
Worker nodes (HM)	1 Gbps+ per node	For HM control components, PostgreSQL replication, internode communication, and metrics/logs.
External bandwidth	1-20 Gbps (aggregated)	S3 backups and inter-cluster replication may require high throughput.

ステップ 3 必要な CSI ドライバーをインストールする

HM は、CSI ドライバーに依存するストレージおよびスナップショット機能を使用します。次のドライバーが OpenShift クラスターにインストールされ、適切に構成されていることを確認します。

- 永続的ストレージドライバー ストレージバックエンドたとえば、OpenShift Container Storage、Ceph、またはその他のオンプレミスソリューションなどと互換性のあるストレージクラスドライバーをインストールします。
- スナップショットコントローラボリュームスナップショット管理用に CSI スナップショットコントローラを構成します。

ステップ 4 ストレージクラスを構成する

デフォルトのストレージクラス HM ワークロードに最適化されたデフォルトのストレージクラスを設定します。

カスタムストレージクラス特定のワークロードでカスタム構成が必要な場合、追加のストレージクラスを作成します。

ローカルストレージオプショナル共有外部ストレージが不足しているオンプレミスクラスターの場合、LVM Storage Operator を使用してローカルストレージを管理します。この方法は、ワーカーノード上のローカルディスクから直接、高性能、低レイテンシーの永続ボリュームをプロビジョニングします。詳細については、LVM storage installation を参照してください。

ステップ 5 名前空間とシークレットを準備する

HM の名前空間 HM コンポーネントの専用の名前空間を作成して、分離と管理性を保証します。

```
oc create ns edbpgai-bootstrap
```

シークレットオブジェクトストレージ資格情報、データベースアクセストークン、またはその他の機密情報など、必要な資格情報の Kubernetes シークレットを作成します。以下は、HM 固有のシークレットと構

成です。最初の ImagePullSecret は、HM が適切に機能するために必要です。カタログのグリップテープと DeltaLake オブジェクトストレージとシークレットは、GenAI Builder またはカタログのどちら用にクラスターを設定するかによって異なるため、オプショナル。

- **ImagePullSecret と名前空間の構成** HM に必要

1. 環境変数を設定します。

OC Registry にもっと簡単にログインできるように、設定するコンテナレジストリ変数がいくつかあります。

```
export CONTAINER_REGISTRY_URI=<local-registry-uri>
export CONTAINER_REGISTRY_USERNAME=<container-registry-username>
export CONTAINER_REGISTRY_PASSWORD=<container-registry-password>
```

2. プルシークレットファイルを準備します。

イメージプルシークレットのディレクトリを準備します。

```
tmpdir=$(mktemp -d /tmp/hm-pull-secret)
chmod 0700 "${tmpdir}"
```

3. レジストリにログインし、シークレットを更新します。

OC レジストリにログインします。

```
oc registry login \
--registry="${CONTAINER_REGISTRY_URI}" \
--auth-basic="${CONTAINER_REGISTRY_USERNAME}:${CONTAINER_REGISTRY_PASSWORD}" \
--to="$tmpdir/pull-secret"
```

次に、シークレットを更新します。

```
oc set data secret/pull-secret -n openshift-config \
    --from-file=.dockerconfigjson="$tmpdir/pull-secret"
```

4. クリーンアップ。

シークレット用に作成した一時ディレクトリをクリーンアップします。

```
rm -rf $tmpdir
```

- **

Create a secret for Griptape and configure DeltaLake object storage GenAI Builder のオプショナル**

- **

Create a secret for Catalog カタログを使用する場合のオプショナル**

- **

Create custom secrets for Migration Portal Migration Portal との内部通信を保護するオプション**

ステップ 6 認証キーとセキュリティキーを構成する

HM とその基になるコンポーネントには、コンポーネント間の適切な通信を保証し、機密データを保護するために、安全な認証メカニズムが必要です。

1. **AES-256 暗号化キーを生成します。**

HM は、AES-256 暗号化を使用して、通信中または保存中の機密データたとえば、データベース資格情報、トークンなどを保護します。ランダムな AES-256 暗号化キーを生成するには

```
export AES_256_KEY=$(openssl rand -base64 32)
```

2. **キーを Kubernetes に保存します。**

HM および関連サービスがキーにアクセスできるようにするには、適切な名前空間に Kubernetes シークレットを作成します。

1. 次のコマンドを実行してシークレットを作成します。

```
kubectl create secret generic hm-auth-key \
    --namespace <hm-namespace> \
    --from-literal=aes-256-key=$AES_256_KEY
```

2. シークレットを確認します。

```
kubectl get secret hm-auth-key --namespace <hm-namespace>
```

ステップ 7 オブジェクトストレージを構成する

オブジェクトストレージバックエンド HM バックアップをサポートするには、クラスターを専用のオブジェクトストレージソリューションたとえば、MinIO、Ceph Object Gateway、または AWS S3 互換ストレージに接続します。

特に、ハイブリッドマネージャーはこのバケットを使用して以下を保存します。

- 管理対象ストレージの場所
- Postgres WAL + バックアップ
- ログ Postgres と内部サービスの両方
- メトリック Postgres と内部サービスの両方

バケットを作成する

MinIO または Ceph bucket を作成します。

バケットユーザーを作成する

プロバイダーでハイブリッドマネージャー用の専用バケットを作成した後、バケットへの完全なアクセスを持つ ID ユーザーと、ID ユーザーの AWS 静的資格情報を作成して、バケットへのハイブリッドマネージャーにアクセスを付与します。

MinIO Client をダウンロードします。

1. MinIO 展開名を取得し、ユーザー名とパスワードを作成します。

```
export MINIO_DEPLOYMENT_NAME=<minio_deployment_name> # deployment name to minio
export BUCKET_NAME=<minio_bucket_name> # bucket name
export AWS_ACCESS_KEY_ID=<minio_user_name> # user name
export AWS_SECRET_ACCESS_KEY=<minio_user_password> # password to the user
export MINIO_POLICY_NAME=<minio_policy_name> # policy to minio
```

2. 新しいユーザーを作成します。

```
mc admin user add ${MINIO_DEPLOYMENT_NAME} ${AWS_ACCESS_KEY_ID} ${AWS_SECRET_ACCESS_
↪KEY}
```

3. MinIO ポリシーを 1 つ作成します。

```
cat << EOF > policy.json
{
    "Version" : "2012-10-17",
    "Statement": [
        {
            "Effect" : "Allow",
            "Action" : [
                "s3:*"
            ],
            "Resource" : [
                "arn:aws:s3:::${BUCKET_NAME}",
                "arn:aws:s3:::${BUCKET_NAME}/*"
            ]
        }
    ]
}
EOF
```

4. ポリシーを適用します。

```
mc admin policy create ${MINIO_DEPLOYMENT_NAME} ${MINIO_POLICY_NAME} ./policy.json
```

5. ポリシーをユーザーにアタッチします。

```
mc admin policy attach ${MINIO_DEPLOYMENT_NAME} ${MINIO_POLICY_NAME} --user ${AWS_
↪ACCESS_KEY_ID}
```

Ceph のコマンドラインインターフェイス Radosgw-admin utility をダウンロードします。

1. バケットといくつかの環境変数を取得します。

```
export BUCKET_NAME=<bucket_name> # bucket name
export USER_NAME=<user_name> # user name
```

2. ユーザーを作成します。

```
radosgw-admin user create --uid ${USER_NAME} --display-name= ${USER_NAME} > user.json

export AWS_ACCESS_KEY_ID=$(cat user.json | jq -r .keys[0].access_key)
export AWS_SECRET_ACCESS_KEY=$(cat user.json | jq -r .keys[0].secret_key)
```

3. ユーザーにバケットへのアクセスを付与します。

```
radosgw-admin bucket link --bucket ${BUCKET_NAME} --uid ${USER_NAME}
```

バケットアクセスのシークレットを作成する

バケットへの HM アクセスのための専用ユーザーを準備した後、作成したユーザーをハイブリッドマネージャーに関連付けるシークレットを作成して、オブジェクトストレージへのアクセスを提供します。

```
apiVersion: v1
kind: Secret
metadata:
  name: edb-object-storage # name cannot be changed
  namespace: default # namespace cannot be changed
stringData:
    auth_type: credentials

    # Optional: Used only when the object storage servers certificate is not issued␣
↪by a well-known CA
    #
    # Base64 string of the CA bundle for the certificate used by the object storage␣
↪server
    aws_ca_bundle_base64: xxx

    # Required: Endpoint URL to the object storage
    aws_endpoint_url_s3: xxx

    # Required: AWS Static Credentials - AWS_ACCESS_KEY_ID
    aws_access_key_id: xxx

    # Required: AWS Static Credentials - AWS_SECRET_ACCESS_KEY
    aws_secret_access_key: xxx

    # Required: Bucket name
    bucket_name: xxx
```

(次のページに続く)

(前のページからの続き)

```
    # Required: Region of the bucket
    aws_region: xxx

    # Optional: true or false
    #
    # When server-side encryption is disabled, set this to true. By default, its␣
↪value is false, indicating that server-side encryption is enabled.
    server_side_encryption_disabled: bool
```

ステップ 8 ネットワークセキュリティを検証する

- ファイアウォールルール OpenShift クラスター、外部オブジェクトストレージ、および HM の外部依存関係の間で必要な通信がファイアウォールルールで許可されていることを確認します。
- 上り/下りルール必要なエンドポイントと IP 範囲のみにアクセスを制限して、露出を最小限に抑えます。

ステップ 9 クラスター構成を調整する

一部のデフォルトの OpenShift 構成は調整が必要になる場合があります。

- ノードプールの調整リソースの割り当て、テイント/トレランスを含む、HM ワークロードのノード構成を最適化します。
- ストレージクラスの調整必要に応じて、デフォルトのストレージクラスを置き換えまたは変更します。
- ネットワークポリシーネットワーク構成を調整して、HM 要件との互換性を保証します。

ステップ 10 必要なアドオンをインストールする

- 必要な OpenShift オペレーターまたはアドオンのインストールと構成
  - サービスメッシュ HM は istio サービスに依存しています Mesh.
  - モニタリングとロギング OpenShift モニタリングおよびロギングソリューションを設定して、HM 可観測性コンポーネントと統合します。

必要な環境変数の設定に進みます。

クラスターが構成されたら、プレインストールの次のフェーズ *setting the necessary environmental variables for installation* に進みます。

Set the necessary environmental variables for install

HM クラスターは、インストール中に特定のユーザードメインへのアクセスが必要です。

ドメイン変数を設定する

1. RHOS クラスターのデフォルトのイングレスドメイン名を定義する変数を設定します。

```
export DEFAULT_INGRESS_DOMAIN=<rhos default ingress domain name> # rhos default␣
↪ingress domain
```

重要

DEFAULT_INGRESS_DOMAIN は、クラスターのデフォルトのイングレスドメイン名です。次のコマンドを使用して照会できます。

```
oc get ingresses.config/cluster -o jsonpath={.spec.domain}
```

ingresses.config/cluster オブジェクトは、インストール中に作成されます。

詳細については、 the OpenShift documentation を参照してください。

1. OpenShift コンソールにアクセスするようにドメイン名を定義する変数を設定します。

```
export OPENSHIFT_CONSOLE_DOMAIN_NAME=<openshift console domain name> # Domain name to␣
↪OpenShift console
```

1. ハイブリッドマネージャーのドメイン名の変数を定義します。このドメインは、HM コンソールにアクセスするためのパブリック URL として機能します。

```
export PORTAL_DOMAIN_NAME=<your-hm-portal-domain-name>
```

たとえば、会社の名前が **Example Company** で、トップレベルドメインが examplecompany.com の場合、EDB はポータルドメイン名がこれらの例と同様のパターンに従うことをお勧めします。

```
portal.examplecompany.com
portal-dev.examplecompany.com
portal.hm001.examplecompany.com
portal-hm-us-east-001.examplecompany.com
```

1. Data Migration Service ドメイン名の変数内部で transporter と呼ばれますを設定します。

```
export TRANSPORTER_RW_SERVICE_DOMAIN_NAME=dms<your-hm-portal-domain-name> # Customer␣
↪domain name for the EDB Postgres AI Platform Transporter RW Service
```

EDB では、この一貫性により、サービスの管理が容易になるため、ポータルドメイン名から DMS サービスドメイン名を取得することをお勧めします、たとえば、portal を dms に置き換えます。前の例に従って

```
dms.examplecompany.com
dms-dev.examplecompany.com
dms.hm001.examplecompany.com
dms-hm-us-east-001.examplecompany.com
```

重要

PORTAL_DOMAIN_NAME および `TRANSPORTER_RW_SERVICE_DOMAIN_NAME` は次のとおりである必要があります。 - 互いに異なる - 両方に同じ名前を使用しないでください - *OPENSHIFT_CONSOLE_DOMAIN_NAME* とは異なる

1. EDB Postgres AI Beacon サービスのドメイン名と cert-manager がインストールされている名前空間を定義する変数を設定します。

```
export BEACON_SERVICE_DOMAIN_NAME=<beacon service domain name> # Customer domain name␣
↪for the EDB Postgres AI Beacon Service
export CERT_MANAGER_NAMESPACE=<cert-manager namespace> # Customer namespace in which␣
↪cert-manager is installed as pre-requirements
```

ストレージクラス変数を設定します

HM クラスターには、事前定義されたストレージクラスが必要です。

```
export STORAGE_CLASS=<storage class> # Customer storage class on OpenShift Container␣
↪Platform
```

認証変数を設定する

HM のインストールにはいくつかの認証値が必要です。

```
export AUTHENTICATION_EMAIL=<email> # The email address of the user.
export AUTHENTICATION_HASH=<password hash>  # The hashed password of the user␣
↪obtained for example as: echo "mypassword" | htpasswd -BinC 10 admin | cut -d: -f
export AUTHENTICATION_USERNAME=<Owner MyCompany> # The username of the user
export AUTHENTICATION_USER_ID="c5998173-a605-449a-a9a5-4a9c33e26df7" # A unique␣
↪identifier for the user must not be changed since the first time installation.
```

オプション FIPs モードを設定します Transporter Data Operator

オプションで、Transporter Data Operator の FIPS モードを設定できます。

```
export TRANSPORTER_FIPS_ENABLED=true # Enable FIPS mode for Transporter Data Operator
```

Install Hybrid Manager on Red Hat OpenShift using EDB Postgres AI Operator

オペレーターの依存関係をインストールします

EDB Postgres AI Operator には、すべての名前空間を管理する Red Hat OpenShiftRHOS の明示的な依存関係 cert-manager オペレーターがあります。 EDB Postgres AI Platform Operator の前に自動的にインストールされます。

Red Hat のオペレーターには、次のような暗黙の依存関係もあります。

- すべての名前空間を管理する RHOS Service Mesh 2
- 名前空間 `openshift-nfd` を管理するノード機能検出オペレーター
- 名前空間 `nvidia-gpu-operator` を管理する NVIDIA GPU Operator

これらのオペレーターを既にインストールしている場合、再インストールする必要はありません。

RHOS のオペレーターをインストールする

1. RHOS に EDB Postgres AI Operator をインストールするには、最初に EDB Cloudsmith リポジトリにアクセスするためのイメージプルシークレットを準備していることを確認します。また、リポジトリ `docker.enterprisedb.com/pgai-platform/edb-hcp-operator/manager` からイメージをプルするために必要な権限があることを確認します。

```
export DOCKER_SERVER="docker.enterprisedb.com"
export DOCKER_USERNAME="pgai-platform"
export DOCKER_PASSWORD="<your-docker-password>"
```

1. EDB EDB Postgres AI Operator の新しい OC プロジェクトを作成します。

```
oc new-project edb-hcp-operator-system
```

1. EDB Cloudsmith リポジトリにアクセスするようにイメージプルシークレットを設定します。

```
oc create secret docker-registry edb-hcp-operator-pull-secret \
  --docker-server="$DOCKER_SERVER" \
  --docker-username="$DOCKER_USERNAME" \
  --docker-password="$DOCKER_PASSWORD" \
  --namespace=edb-hcp-operator-system
```

1. OpenShift Container PlatformOCP を使用して、Red Hat OpenShift の EDB Postgres AI Operator をインストールします。
2. OCP Web コンソールにログインします。
3. **Operators > OperatorHub** を選択します。
4. フィルタボックスに `EDB Postgres AI` と入力します。
5. **EDB Postgres AI Operator** を選択し、 **Install** を選択します。
6. [Install Operator] ページで、次のことを行います。オペレーターには、前に作成した `edb-hcp-operator-system` 名前を使用して **インストールされた名前空間** を選択します。 b. **Install** を選択します。

> **Tip**
>
> RHOS では、cert-manager はハイブリッドマネージャーの必須依存関係です。 cert-manager をインストールするときは、正常に動作するように、**インストールモード**が**クラスター上のすべての名前空間**として構成されていることを確認します。

RHOS にハイブリッドマネージャーをインストールする

EDB Postgres AI Operator を RHOS クラスターにインストールした後、Operator を使用して Hybrid Manager をインストールする必要があります。次の YAML ファイルは、ハイブリッドマネージャーのインストールを開始します。

```
apiVersion: edbpgai.edb.com/v1alpha1
kind: HybridControlPlane
metadata:
  name: edbpgai
spec:
  source:
    useLocalKustomizations: true
  flavour: rhos
  version: v1.2.0+appl
  imageRegistry: ${CONTAINER_REGISTRY_URI}
  imagePullSecrets:
    - name: edb-cred
      namespace: edbpgai-bootstrap
  globalParameters:
    internal_backup_folder: 4befab698c8d
    portal_domain_name: ${PORTAL_DOMAIN_NAME}
    default_ingress_domain: ${DEFAULT_INGRESS_DOMAIN}
    storage_class: ${STORAGE_CLASS}
  componentsParameters:
    edb-migration-portal:
      enable_pdb: "false"
    transporter-data-operator:
      fips_enabled: "${TRANSPORTER_FIPS_ENABLED}"
    transporter-dp-agent:
      rw_service_url: https://${TRANSPORTER_RW_SERVICE_DOMAIN_NAME}/transporter
    transporter-rw-service:
      domain_name: ${TRANSPORTER_RW_SERVICE_DOMAIN_NAME}
    upm-api-pnxt-metric:
      log_level: info
      tsdb_host: thanos-query.monitoring.svc.cluster.local
      tsdb_port: "9090"
    upm-api-pnxt-pd:
      log_level: info
    upm-beaco-ff-base:
      usage_generator_interval: 24h
    upm-beaco-usages:
      usage_generator_interval: 24h
    upm-beacon:
      server_host: ${BEACON_SERVICE_DOMAIN_NAME}
      beacon_server_replicas: "1"
```

(次のページに続く)

(前のページからの続き)

```
      external_metrics_replicas: "1"
      beacon_db_hibernation: "\"off\""
    upm-istio-gateway:
      openshift_console_domain_name: ${OPENSHIFT_CONSOLE_DOMAIN_NAME}
      cookie_aeskey: ${AES_256_KEY}
      ingress_http_node_port: "32542"
      ingress_https_node_port: "30288"
      ingress_grpc_tls_node_port: "30290"
    upm-prometheus:
      memory: 4Gi
    upm-trust-manager:
      cert_manager_namespace: ${CERT_MANAGER_NAMESPACE}
  pgai:
    portal:
      authentication:
        idpConnectors: []
        staticPasswords:
          - email: ${AUTHENTICATION_EMAIL}
            hash: ${AUTHENTICATION_HASH}
            username: ${AUTHENTICATION_USERNAME}
            userID: ${AUTHENTICATION_USER_ID}
  beaconAgent:
    applianceProviderEnabled: true
    plaintext: true
    beaconServerAddress: "upm-beacon-server-ingestion.upm-beacon.svc.cluster.
↪local:8080"
    tlsInsecure: false
    location: "devspatcher"
    logLevel: "debug"
    beaconDBConnected: true
    inCluster: true
    provisioning:
      loadBalancersEnabled: true
      nodePortDomain: "nodeportdomain.com"
      imageDiscovery: true
      imagesetDiscoveryAuthenticationType: "basic"
      imagesetDiscoveryContainerRegistryURL: "us-east1-docker.pkg.dev"
    providers:
      - appliance
```

YAML が適用された後、次のコマンドを使用してインストールステータスを監視できます。

```
oc get hybridcontrolplane edbpgai -o yaml
```

インストールが現在いるフェーズに応じて、3 つの異なるステータスを返すことができます。

```
- deployed # the installation was successful

- deploying # the installation is still in progress

- failed: # the install was unsuccessful
```

インストールが成功すると、 *updating DNS zone records and verify the install* .

Operator を介してインストールされた Hybrid Manager のアンインストール

> **警告**
>
> 次の手順は、Hybrid Manager をアンインストールします。 Hybrid Manager をアンインストールする場合にのみ、この手順を使用します。

ハイブリッドマネージャーを削除する必要がある場合があります。ハイブリッドマネージャーはデフォルトで削除から保護されているため、最初にその保護を無効にする必要があります。

```
kubectl annotate hybridcontrolplane edbpgai biganimal.enterprisedb.io/deletion-
↪protect=disabled --overwrite
```

次に、アンインストールできます。

```
kubectl delete hybridcontrolplane edbpgai --wait=false
```

プロセスをモニタするには、次を実行します。

```
kubectl get hybridcontrolplane edbpgai -o yaml
```

Connecting to Hybrid Manager

EDB Postgres AI Hybrid Manager プラットフォームのインストールが成功したことを確認するには、ブラウザーを使用して Hybrid Manager コンソールに接続します。

```
https://portal.examplecompany.com
```

この構成では、ホスト名が広域インターネットに存在しないため、https は検証されません。証明書の警告を受け入れて、ポータルに進みます。証明書の警告を受け入れて、ポータルに進みます。

ログイン画面で、ブートストラップインストールで生成されたユーザー名とパスワードを入力できます。詳細については、Managing users を参照してください。

ログインしたら、 *Create a database cluster template* ができます。

Updating DNS zone records

Helm または Operator のインストールが完了したら、DNS ゾーンレコードを更新し、ハイブリッドマネージャーが予想通りに動作していることを確認します。

DNS ゾーンレコードの更新

1. ポータルサービスのホスト名を取得します。このホスト名は、ハイブリッドマネージャーの入力エンドポイントを表します。

```
kubectl get svc -n istio-system istio-ingressgateway -o yaml | yq .status.
↪loadBalancer.ingress[0].hostname
```

1. 次のドメイン名の DNS ゾーンに CNAME レコードを作成し、取得したホスト名をポイントします。
    - PORTAL_DOMAIN_NAME
    - TRANSPORTER_RW_SERVICE_DOMAIN_NAME
    - BEACON_SERVICE_DOMAIN_NAME

各 CNAME レコードは、ユーザー向けドメイン名を入力ホスト名にマッピングします。

これで、 *connect to the portal* を実行して、インストールを確認できます。

5.3.2 Install Hybrid Manager on Rancher RKE2

Rancher に Hybrid ManagerHM をインストールするには

- *Prerequisites for AI Factory on Hybrid Manager* を実行します。
- *installation using the Helm chart* を実行します
- *Update your DNS zone records*
- *Connecting to Hybrid Manager* によるインストールの検証

The installation prerequisites for installing Hybrid Manager on Rancher RKE2

Rancher に Hybrid ManagerHM をインストールする前に、満たす必要がある前提条件があります。

インストールプロセスを開始する前に、前提条件を満たす必要があることを確認します。

前提条件

- 次のユーティリティを使用してインストールプロセスを実行できるシステム。
    - kubectl Kubernetes を管理する cluster
    - Helm Kubernetes マニフェストをレンダリングおよび適用します。
    - edbctl
- *A RKE2 cluster preconfigured for use with HM*
- *Sync container images to your registry*

次のステップ

これらの要件を満たしたら、*Installation phase of Hybrid Manager on Rancher RKE2* に進むことができます。

Preconfigure a Rancher RKE2 cluster for use with Hybrid Manager

Prerequisites for AI Factory on Hybrid Manager が完了したら、Hybrid ManagerHM で使用する Rancher RKE2 クラスターをセットアップする準備がほぼ整います。

ただし、最初に、事前構成とインストールの両方が依存する Helm チャートが必要です。

Helm チャートを構成する

Helm チャート values.yaml は、HM プラットフォームのコア構成であり、インストールの中心です。

事前構成プロセスとインストールフェーズを通じて、Helm チャート `values.yaml` の更新を参照する手順があります。これらは、事前構成後のインストールフェーズを成功させるために重要です。

EDB Helm リポジトリの追加とチャートの取得

EDB Repos 2.0 トークンを使用して、リポジトリを追加し、構成するチャートを取得するだけです。

1. EDB Repos 2.0 から Helm チャートリポジトリを追加します。

```
helm repo add enterprisedb-edbpgai "https://downloads.enterprisedb.com/<your-edb-
↪repos-token>/pgai-platform/helm/charts/"
```

<your-edb-repos-token> は、EDB Repos 2.0 アクセストークンです。

2. リポジトリを更新します。

```
helm repo update
```

3. デフォルトの Helm チャート `values.yaml` ファイルを取得します。

```
helm show values enterprisedb-edbpgai/edbpgai-bootstrap > values.yaml
```

RKE2 クラスターの作成

HM を事前構成およびインストールするには、RKE2 クラスターが必要です。HM インストールを成功させるには、RKE2 クラスターを正確に事前構成する必要があります。

このガイドの残りの部分では、完全な事前構成プロセスを説明します。

サポートされている RKE2:Kubernetes バージョンのペア

- HM は現在以下をサポートしています
  - RKE2 1.31.X with Kubernetes 1.31.X
  - RKE2 1.32.X with Kubernetes 1.32.X

これらの構成のいずれを実装するかがわかったら、クラウドベースの展開用の CSP または標準のオンプレミスワークフローを使用して、クラスター作成プロセスを開始します。

ノードサイズとリソースの構成

HM とその関連コンポーネントをサポートするには、Rancher RKE2 クラスター内のノードがリソース要件を満たしている必要があります。

HM を実行するワーカーノードに進む前に、これらのマスターノードを設定する方法については、Rancher RKE2 を参照してください。

ノードのサイジング

RKE2 マスターノードと Kubernetes クラスターが作成された後、HM に必要な 2 セットのワーカーノード、HM コントロールプレーンノードと HM Postgres データノードに移動できます。

HM コントロールプレーンノード

HM コントロールプレーン CP ワーカーノードは、以下に概要を説明するように、リソース割り当ての最小要件ワーカーノードの数と各 CPU、メモリ、およびディスクサイズを満たしている必要があります。

クラスターに必要なテレメトリスタック特に Prometheus と Thanos のサイズは、HM CP ノードのスケールアップ/アウトを促進する主な要因です。テレメトリスタックは、CP が監視している Postgres データベースの数に応じてスケーリングするため、CP ノードは管理対象データベースの数に応じてスケーリングします。

- 一般的な推奨事項
  - 3 つの CP ノードそれぞれが次のとおりです。
    - CPU:
      - 最小 8 vCPU 最大 10 の Postgres データベースの場合
      - 推奨 16 vCPU 中規模のクラスター 10〜50 の Postgres データベース用
      - 50 Postgres データベースの場合 16+ vCP Us
    - メモリ
      - 最小 32 GB RAM 最大 10 の Postgres データベースの場合
      - 推奨 64 GB RAM 以上大規模なワークロード 10-50 の Postgres データベース用
      - 50 Postgres データベースの場合 64+ GB RAM
    - Disk
      - 最小 100 GB SSD 最大 10 の Postgres データベースの場合
      - 推奨 200 GB SSD10-50 の Postgres データベースの場合
      - 50 Postgres データベースの場合 >200 GB SSD

Postgres データノード

小規模なワークロード 5〜20 の Postgres データベースの場合、Postgres データノードの少なくとも 3 つのワーカーノードをお勧めします。Postgres データベースを約 20 を超えてスケールアップすると、Postgres ワークロード用により多くのワーカーノードが必要になり、最終的には増加する Postgres データベースをサポートするためにさらに多くの CP ノードが必要になる場合があります。

- 一般的な推奨事項
  - 6 つの Postgres データノード、それぞれに
    - CPU:
      - 推奨 16 vCPU
    - メモリ:
      - 推奨: ノードごとに 32 GB の RAM
    - ディスク:
      - 最小: ノードごとに 100 GB の永続ストレージデータベースとロギングに基づいて調整します要件)。最適なパフォーマンスを得るには、高速ディスク SSD を使用します。

GPU ノードのプロビジョニングとオペレーターのセットアップ AI / ML のオプション

モデルの展開に GPU 機能を活用する予定がある場合は、次の手順に従って、GPU ノードをプロビジョニングおよび有効にします。

Rancher マシンプールの回避策

Rancher UI は、マシンプールを作成するときに GPU を備えた特定のインスタンスタイプを見逃す場合があります。この回避策を使用して、必要なノードをプロビジョニングします。

任意の標準のインスタンスタイプでマシンプールを作成し、初期レプリカ数を 0 に設定します。

マシンプール構成を手動で更新して、必要な GPU インスタンスタイプ g6e.12xlarge などを指定します。Rancher は、この更新フェーズ中に不足しているインスタンスタイプを受け入れる必要があります。

マシンプールを目的の数の GPU 対応ノードにスケールアップします。

NVIDIA GPU Operator のインストール

GPU ノードがプロビジョニングされた後、NVIDIA GPU Operator をインストールして、Kubernetes ポッドで GPU リソースを利用できるようにする必要があります。

公式の Rancher RKE2 documentation に従って、NVIDIA GPU Operator をインストールします。

インストール後、これらの新しいノードに適切なラベルを付け、AI / ML モデルの展開に活用できます。

CP vs Postgres データノードマシンセットのテイントとラベル

HM CP ノードと、Postgres ワークロードをホストする HM Postgres データノードのさまざまなタイプのノードを使用するには、RKE2 がそれぞれ独自のテイントとラベルを持つ 2 つの異なるマシンセットが必要です。

- CP ノードのテイントとラベル

```
spec:
    replicas: 3
    template:
    spec:
        metadata:
        labels:
            edbaiplatform.io/control-plane: "true"
    taints:
    - key: edbplatform.io/control-plane
        value: "true"
        effect: NoSchedule
```

- Postgres データノードのテイントとラベル

```
spec:
    replicas: 3
    template:
    spec:
        metadata:
        labels:
            edbaiplatform.io/postgres: “true”
        taints:
        - key: edbplatform.io/postgres
            value: "true"
            effect: NoSchedule
```

ネットワーキング帯域幅

各ノードに、HM 通信と外部データ転送 S3 バックアップなどを処理するための十分なネットワーク容量があることを確認します。

ベースラインの推奨事項

Component	Recommended bandwidth	Justification
K8s control plane nodes	1 Gbps+ per node	Handles internal Kubernetes traffic, API server requests, logs/metrics, and orchestration tasks.

次のページに続く

表 2 – 前のページからの続き

Component	Recommended bandwidth	Justification
Worker nodes (HM)	1 Gbps+ per node	For HM control components, PostgreSQL replication, internode communication, and metrics/logs.
External bandwidth	1-20 Gbps (aggregated)	S3 backups and inter-cluster replication may require high throughput.

ネットワーク構成と要件

基本計画とネットワーキングアーキテクチャ

Kubernetes クラスターの作成を終了する前に、一連の高レベルのネットワークとファイアウォールの決定を行う必要があります。これらの選択は、展開プロセスの残りの部分の基礎を形成します。クラウドで展開するかオンプレミスで展開するかにかかわらず、全体的なシーケンスは同じですが、メカニズムが異なります。

高レベルのネットワークの決定

ネットワークの基本的な特性を決定します。

- **ネットワークスタック** ネットワークは IPv4 である必要があります。
- **CNI/ネットワークタイプ** Calico を使用した標準の SDN オーバーレイをお勧めします。
- **DHCP vs static** DHCP がノードアドレスを割り当てるかどうか、または静的 IP を割り当てる必要があるかどうかを確認します。

クラウドベースの展開の場合、これらの決定は、CSP のクラスター作成ウィザードの管理オプションにマッピングされることがよくあります。

オンプレミス展開の場合、ネットワークチームは VLAN、サブネット、ルーティング、および DHCP サーバーを構成する必要があります。

IP アドレスと CIDR の計画

アーキテクチャに基づいて、オーバーラップを回避するようにアドレス空間を注意深く割り当てる必要があります。これはネットワークのブループリントです。

静的 IP

静的 IP

- API 仮想 IP HA コントロールプレーン/API エンドポイント用
- イングレス仮想 IP MetalLB のようなオンプレミス LB、またはハードウェア LB によって使用されます
- デフォルトゲートウェイ IP
- DNS サーバー IP

- NTP サーバー IP

ネットワーク範囲

- マシンネットワーク CIDR アンダーレイ/pnetwork の場合。
- Management Network CIDR 管理アクセス用。
- クラスターネットワーク CIDR Kubernetes 内のポッド IP アドレスの場合。
- Service Network CIDR 内部 Kubernetes サービスの場合。

ダウンストリームクラススペース

プラットフォームで必要な場合、各ダウンストリームクラスターの/16 IPv4 ブロックを予約します。

コアクラスターネットワーキング

ネットワーク計画が整ったら、次のステップは、クラスターの内部通信ファブリックを有効にし、ポッドレベルで保護することです。これには、コンテナネットワークインターフェイス CNI のインストールまたは確認、およびベースラインネットワークポリシーの適用が含まれます。

コンテナネットワークインターフェイス CNI の構成

動作する CNI は、ポッドが相互に通信するために、およびネットワークポリシーのために重要です。

すべての Kubernetes クラスターには、ポッド間のネットワーキングを提供する CNI が必要です。

クラウド展開

- EKSAWS デフォルトで AWS VPC CNI を使用します。これは、VPC からポッドにセカンダリ IP を割り当てます。これは、AWS ネットワーキングおよびセキュリティグループとネイティブに統合されます。高度なネットワークポリシーまたは eBPF 機能が必要な場合は、Calico ポリシーのみを追加するか、Cilium をインストールできます。
- GKE Google Cloud デフォルトで Google の管理データプレーンを使用します。これは、ネットワークポリシーをサポートし、Google のファイアウォールと統合します。サードパーティ CNI が必要になることはほとんどありません。

オンプレミス展開

選択した CNI Calico が推奨されますが、ドキュメントに従って正しくインストールおよび構成されていることを確認します。

RKE2 は、デフォルトで有効になった Canal Calico + Flannel を同梱して出荷されています。別の CNI を使用するには、インストール中に Canal を無効にし、優先する CNI を展開します。

この手順により、クラスターの内部ネットワークが機能します。

!!!警告

すべてのノードが `Ready` を報告し、ポッドがポッド CIDR から IP を受信し、クロスノードポッドネットワーキングが機能するまで、続行しないでください。

CNI がアクティブになったら、サービスを外部に公開する前に、ベースライン分離ポリシーをすぐに確立します。

- 各名前空間のデフォルトの deny ingress ポリシーで開始します。
-
  - DNS egress to CoreDNS.
  - コントロールプレーン →Postgres データノード通信の許可ルールを追加します。

メトリックとロギングパイプライン。

これにより、クラスターは安全な最小権限のベースラインから起動します。

クラウドベースの展開の場合、CNI でポリシーの適用を有効にする必要がある場合があります AWS VPC CNI の Calico ポリシーモードなど。

オンプレミスシナリオの場合、Calico、Cilium、および OVN-K はネイティブにネットワークポリシーを適用します。

ロードバランサーを作成する

HM をクラウドに展開する場合は、適切なロードバランサーコントローラをインストールまたは有効にしてから、Service または Ingress リソースを適用してロードバランサーの作成をトリガーします。

外部アクセスとファイアウォールルール

クラスターの内部ネットワークが実行されているので、外部トラフィックが内部で実行されているサービスに到達する方法を構成します。

オンプレミス展開 NodePort またはカスタム LB

オンプレミス展開の場合、外部アクセスは通常 **NodePort** サービスを介して提供されます。

- デフォルトでは、HM コンポーネントは特定の NodePort 値でサービスを公開します。
- よりわかりやすい DNS 名とより良いフェイルオーバーのために、NodePort を直接使用するか、**MetalLB** ソフトウェアロードバランサーまたは**ハードウェアロードバランサー**でこれらのポートのフロントを使用できます。

必要なポートデフォルトの NodePort 値

- **32542** – HM ポータル HTTP
- **30288** – HM ポータル HTTPS
- **30290** – Beacon gRPC
- **30292** – Spire TLS

これらのデフォルトを変更する必要がある場合は、Helm チャートの `parameters.upm-istio-gateway` の下の `values.yaml` を更新します。

```
ingress_http_node_port: <port>
ingress_https_node_port: <port>
```

(次のページに続く)

(前のページからの続き)

```
ingress_grpc_tls_node_port: <port>
ingress_spire_tls_node_port: <port>
```

次に、再定義したポートのファイアウォールを開きます。

クラウドベースの展開ロードバランサー

クラウド EKS、GKE に HM を展開する場合は、Kubernetes Service `LoadBalancer` または `Ingress` リソースを使用します。

EKS で、AWS ロードバランサーコントローラをインストールします。GKE では、コントローラはビルトインされています。

必要なポートクラウドロードバランサー

- **443** – HM ポータル HTTPS イングレス
- **8444** – HM 内部 API
- **9443** – Beacon gRPC API
- **9445** – Spire TLS

DNS、TLS、およびアプリケーション構成

外部アクセスパスが定義されると、サービスに使いやすい名前を割り当て、保護できるようになりました。

DNS 構成

ロードバランサーまたはノードポート構成

オンプレミスで展開する場合、`loadBalancerEnabled` を `true` に設定しません。代わりに、Helm チャートの `values.yaml` をそれに応じて構成します。

```
beaconAgent:
  provisioning:
    loadBalancersEnabled: false
    nodePortDomain: "<your-node-port-domain>"
```

!!!注

`nodePortDomain` 値は、すべての Postgres インスタンスの URL として使用されます。これは、Postgres クラスターが実行されているノードの IP アドレスを指す DNS 名である必要があります。

外部オンプレミスロードバランサー戦略をノードポート戦略と組み合わせて使用する場合、`nodePortDomain` は、Postgres ノードプールを指すロードバランサーの FQDN を指す必要があります。このようにして、各 Postgres インスタンスに対して生成された URL は機能します。`metallb` (https://metallb.io/) などのロードバランサーコントローラが使用されている場合、トラフィックを適切なノードプールサービスに適切にルーティングするには、別の方法で構成する必要があります。

パブリックドメイン名と内部ドメイン名が、前の手順で構成したエントリーポイントをポイントします。

- HM のルートドメイン名を決定した後、Helm チャートの `parameters` の下の `values.yaml` フィールドが構成されていることを確認します。
- `global.portal_domain_name: "portal.<root_domain>` HM ポータルのホスト名。
- `upm-beacon.server_host: "beacon.<root_domain>"` ビーコンサーバー API が到達可能なホスト名。
- `transporter-rw-service:domain_name: "transporter.<root_domain>"` 内部トランスポーター移行読み取り/書き込みサービスのドメイン名。
- `transporter-dp-agent:rw_service_url: "transporter.<root_domain>/transporter""` 内部 Transporter 移行読み取り/書き込みサービスの URL。
- クラウドベースのロードバランサー IP または `nodePortDomain` によってカバーされるノード IP を指すポータルドメイン portal_domain_name の DNS A レコードを作成します。

TLS 証明書管理

TLS certificates を使用してエンドポイント HM ポータルなどを保護します。

これには、DNS ホスト名をファイナライズする必要がありますので、前の手順を参照して、正しい名前に証明書を発行できるようにします。

デフォルトのフォールバックオプションとして自動生成の自己署名証明書を使用する *Custom certificates* を強くお勧めします。

HM ポータルの *HM* **ポータルのカスタム** *cert-manager* **発行者を設定する** をセットアップし、独自の **独自のプライベート認証局を使用する** をセットアップすることもできます。

ユーザー ID 構成

ユーザー ID を構成するには、独自の IdP を実装するか、HM のネイティブユーザー機能を使用するかを選択する必要があります。

実稼働環境でのユーザーアクセスを管理するには、HM で *set up your identity provider (IdP)* を行うことを強くお勧めします。

Changing password for native users はデフォルトでサポートされていますが、実稼働環境ではこの方法でユーザーを管理することはお勧めできません。

最初のネイティブユーザー userA を作成するには、ユーザーのメールアドレス、ハッシュ、ユーザー ID、およびユーザー名が必要です。

Parameter	Description
email	ユーザーのメールアドレス。コンソールにアクセスするためのユーザーのログイン識別子としても機能します。

次のページに続く

表 3 – 前のページからの続き

Parameter	Description
hash	パスワードストアの Bcrypt ハッシュ化されたユーザーパスワード。この値を生成するには、 `echo ${password} I htpasswd -BinC 10 userA I cut -d: -f2`を使用します。ここで、実際のパスワードは`${password}`変数の後ろに保存されます。 `userA`は、パスワードのハッシュプロセス中に使用されるユーザー名を表します。構成内の他の場所で使用されていないため、任意のテキストにすることができます。結果のハッシュのみが利用されます。
userID	静的パスワードで構成された各新しいユーザーには、ユーザーの一意の識別子 uuid が必要です。 UUID ジェネレータツールでこの値を生成するか、文字のランダムなシーケンスを手動で割り当てることを選択できます。
username	ユーザーの一意のユーザー名。これは、コンソールにログインするためのプライマリ識別子です。 `email`と同じでもかまいません。

次に、Helm チャートの `values.yaml` の `pgai:portal:authentication:staticPasswords` の下にこれらの値を設定します。

ネットワークセキュリティポリシーの強化

サービスが公開され、TLS / DNS が構成されたら、ネットワークポリシーとファイアウォールルールを調整して、より厳密な境界を適用します。

ネットワークポリシー

Kubernetes `NetworkPolicy` リソースを作成して、相互に通信できるポッドを制御します。これは、コントロールプレーン、データノード、およびその他のシステムコンポーネント間のトラフィックを保護するために重要です。

上り/下りルール

ファイアウォールの上り/下りルールを検証および適用して、必要なトラフィックのみがクラスターへの出入りを許可され、他のすべてのアクセスを制限します。

<<<<<<< 私たちのもの

EDB Postgres AI Platform コンテナイメージを顧客所有のレジストリに同期する

HM のソフトウェアスタックは EDB Repos 2.0 レジストリにプッシュされ、ローカルレジストリで使用するアーティファクトを提供します。Rancher RKE2 オンプレミスシナリオの顧客管理の内部レジストリまたはクラウドサービスプロバイダー CSP の顧客管理レジストリ CSP シナリオで Rancher の HM のレジストリ AWS Elastic Container Registry ECR、 Google Cloud Artifact Registry AR。

独自の安全で承認されたローカルレジストリを持っており、以下の同期プロセスでローカルコンテナレジストリの URI、ユーザー、およびパスワードを使用します。インストールする EDB PGAI バージョンを知っている必要があります。この情報により、EDB Repos 2.0 のすべてのアーティファクトは、Helm チャートでソフトウェアスタックをインストールまたはアップグレードする前に、ローカルレジストリと内部同期します。

同期プロセスは、コンテナイメージの SHA256 を保持して、さまざまな環境全体でイメージのセキュリティと不変性を保証する必要があります。

edbctl を使用して同期する

1. edbctl is installed and configured がまだない場合は確認します。

2. 必要な環境変数を構成します。

- 取得する EDB PGAI リリースを定義します。

```
export EDBPGAI_RELEASE=<EDB-pgai-release-version>
```

- EDB Repos 2.0 アクセストークンを定義します。

```
export CS_EDB_TOKEN=<your-edb-repos-token>
```

- EDB Repos 2.0 レジストリソースを定義します。

```
export EDB_SOURCE_REGISTRY=pgai-platform
```

- ローカルコンテナレジストリ、ユーザー、およびパスワードを定義します。

```
export LOCAL_REGISTRY_URI=<your_local_container_registry_uri
export LOCAL_REGISTRY_USER=><your_local_registry_user>
export LOCAL_REGISTRY_PWD=<your_local_registry_password_for_your_user>
```

1. sync-to-local-registry コマンドを実行します。

```
edbctl image sync-to-local-registry \
    --destination-registry-uri "${LOCAL_REGISTRY_URI}" \
    --version "${EDBPGAI_RELEASE}" \
    --source-registry-username "${EDB_SOURCE_REGISTRY}" \
    --source-registry-password "${CS_EDB_TOKEN}" \
    --destination-registry-username "${LOCAL_REGISTRY_USER}" \
  --destination-registry-password "${LOCAL_REGISTRY_PWD}"
```

2. EDB PGAI Operator イメージを宛先レジストリに同期します。

```
edbctl operator sync-to-local-registry \
    --destination-registry-uri "${LOCAL_REGISTRY_URI}" \
    --version "${EDBPGAI_RELEASE}" \
    --source-registry-username "${EDB_SOURCE_REGISTRY}" \
    --source-registry-password "${CS_EDB_TOKEN}" \
    --destination-registry-username "${LOCAL_REGISTRY_USER}" \
    --destination-registry-password "${LOCAL_REGISTRY_PWD}"
```

ローカルレジストリが EDB Repos 2.0 レジストリと同期されました。 =======

同期コンテナイメージ

彼らのもの

続行する前に、*Sync container images to your registry* の説明に従って、ハイブリッドマネージャーコンテナイメージをプライベートレジストリに同期していることを確認してください。これはインストールの必須前提条件です。

同期後、Helm チャート `values.yaml` を更新してレジストリを参照します。

```
containerRegistryURL: "<your-local-container-registry-url>"
```

イメージ検出構成

イメージ検出は、エージェントビーコンで実行され、Postgres イメージを検出するプロセスです。このプロセスは、顧客管理のローカルコンテナレジストリ前の手順で同期されたに接続します。これは OCI compliant である必要があります。OCI 準拠のレジストリは HM でサポートされています。

イメージ検出のための Helm チャートの構成

1. HM のイメージ検出を有効にするには、最初に `values.yaml` の `beaconAgent.provisioning:imageDiscovery` の値を `true` に変更します。
2. 次に、`beaconAgent.provisioning:imagesetDiscoveryContainerRegistryURL` を、HM が Postgres コンテナイメージを検出するコンテナレジストリであるため、前の手順で EDB イメージを同期したローカルコンテナレジストリに設定します。
3. 証明書の検証なしで TLS 接続を確立する予定がある場合は、`imagesetDiscoveryAllowInsecureRegistry` オプションを `true` に設定します。

レジストリの資格情報

イメージ検出プロセスは、Kubernetes イメージプルシークレットを使用してローカルコンテナイメージレジストリで認証します次のステップを参照してください。したがって、使用されるサービスアカウントまたはプリンシパルには、次の権限が必要です。

- リポジトリのリスト
- リストタグ
- タグマニフェストの読み取り

オンプレミスシナリオ前の手順で設定の構成指示については、ローカルの内部コンテナレジストリのドキュメント quay などを参照して、HM がアクセスできるようにコンテナレジストリに必要なレジストリ権限を構成します。

イメージに CSP ベースのローカルレジストリ AWS Elastic Container Registry、Google Cloud の Artifact RegistryAR を使用する場合、次の例をガイドとして使用します。

AWS ECR

EKS で ECR を使用する場合、サポートされている認証タイプは eks_managed_identity のみです。eks_managed_identity を使用する前に、権限 AmazonEC2ContainerRegistryReadOnly を持つロールを作成し、そのロールを EKS クラスターポッド ID に関連付ける必要があります。

```
EKS_CLUSTER_NAME="<eks_cluster_name>"
EKS_CLUSTER_REGION="<eks_cluster_region>"
IMAGE_DISCOVERY_IAM_ROLE_NAME="<iam_role_name>"

cat <<EOF > ./image-discovery-trust.json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
            "Effect": "Allow",
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ]
        }
    ]
}
EOF

aws iam create-role --role-name "${IMAGE_DISCOVERY_IAM_ROLE_NAME}" \
    --assume-role-policy-document file://image-discovery-trust.json

aws iam attach-role-policy --role-name "${IMAGE_DISCOVERY_IAM_ROLE_NAME}" \
    --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly

IMAGE_DISCOVERY_IAM_ROLE_ARN=$(aws iam get-role --role-name ${IMAGE_DISCOVERY_IAM_
↪ROLE_NAME} | jq -r .Role.Arn)
aws eks create-pod-identity-association --cluster-name "${EKS_CLUSTER_NAME}" \
    --namespace upm-beacon \
    --service-account upm-beacon-agent-k8s \
    --role-arn "${IMAGE_DISCOVERY_IAM_ROLE_ARN}" \
    --region "${EKS_CLUSTER_REGION}"
```

Google Cloud AR

Google Kubernetes EngineGKE で Google Cloud AR を使用する場合、イメージプルシークレットの生成に使用されるサービスアカウントには、次のロールが必要です。

- プロジェクトレベルの roles/artifactregistry.reader
- プロジェクトレベルの roles/browser 特に、AR 内のリポジトリの取得を許可するには、権限 resourcemanager.projects.list が必要です。

```
gcloud projects add-iam-policy-binding <PROJECT-ID> \
    --member="serviceAccount:<SERVICE-ACCOUNT-NAME>@<PROJECT-ID>.iam.gserviceaccount.
↪com" \
    --role="roles/artifactregistry.reader"

gcloud projects add-iam-policy-binding <PROJECT-ID> \
    --member="serviceAccount:<SERVICE-ACCOUNT-NAME>@<PROJECT-ID>.iam.gserviceaccount.
↪com" \
    --role="roles/browser"
```

詳細については、Google Cloud AR roles documentation および Resource Manager roles documentation を参照してください。

シークレット値の設定

イメージ検出の構成の最後のステップは、parameters.upm-beacon:image_discovery_secret_name を確認することです。これは、イメージ検出用のコンテナレジストリの資格情報を含む Kubernetes シークレットの名前です。デフォルトでは、edb-cred です。

名前空間と予備的なシークレット

オブジェクトストレージと一般的な Kubernetes シークレットの準備

オブジェクトストレージ資格情報例 aws_secret_access_key 、データベースアクセストークン、またはその他の機密情報などの必要な資格情報の Kubernetes シークレットを作成します。

ImagePullSecret 名前空間と必要なシークレット

edbctl を使用して、ImagePullSecret 名前空間と必要なシークレットを作成します。

1. 必要な名前空間を作成し、シークレットをプルします。

```
edbctl image-pull-secret create \
  --username <container registry username> \
  --password <container registry password> \
  --registry <local registry URI>
```

2. 現在の Kubernetes コンテキストで Proceed? [y/N] とプロンプトが表示されたら、y を選択します。

次の例のようなものが表示されるはずです。

```
2025/02/10 10:10:10 Creating Kubernetes Namespaces and ImagePullSecrets with the␣
↪provided credentials...
2025/02/07 15:29:08 Namespaces and ImagePullSecrets creation completed
```

3. ImagePullSecret リストを作成します。

```
edbctl image-pull-secret list
```

次の出力例のようなものが表示されます。

```
Current Kubernetes context is: <your-KubeContext>
Namespace edbpgai-bootstrap: exists, all set!
  Secret edb-cred: exists, all set!
Namespace upm-replicator: exists, all set!
  Secret edb-cred: exists, all set!
```

認証キーとセキュリティキー

HM とその基になるコンポーネントには、コンポーネント間の適切な通信を保証し、機密データを保護するために、安全な認証メカニズムが必要です。

キーを生成し、シークレットに保存します

1. AES-256 暗号化キーを生成します。

HM は、AES-256 暗号化を使用して、通信中または保存中の機密データたとえば、データベース資格情報、トークンなどを保護します。ランダムな AES-256 暗号化キーを生成するには

```
export AES_256_KEY=$(openssl rand -base64 32)
```

2. キーを Kubernetes に保存します。

HM および関連サービスがキーにアクセスできるようにするには、適切な名前空間に Kubernetes シークレットを作成します。

1. 次のコマンドを実行してシークレットを作成します。

```
kubectl create secret generic hm-auth-key \
    --namespace <hm-namespace-created-above> \
    --from-literal=aes-256-key=$AES_256_KEY
```

2. シークレットを確認します。

```
kubectl get secret hm-auth-key --namespace <hm-namespace>
```

Helm チャートの更新

Helm チャート `values.yaml` で、フィールド `parameters.upm-istio-gateway.cooke_aeskey` を前の手順で生成した AES-256 暗号化キーに設定します。

その他に必要なシークレット

1. Create a secret for GenAI Builder and configure Delta Lake object storage 。

2. Create a secret for Catalog 。

Migration Portal の保護

Create custom secrets for Migration Portal 移行ポータルの内部通信を保護する場合。

ストレージとクラスターの準備

ブロックストレージ構成

HM は、すべてのプライマリおよびステートフルワークロードにブロックストレージを使用します。Kubernetes ストレージクラスを使用してパフォーマンスとコストを管理する戦略的アプローチをお勧めします。

ワークロードの分離

カスタムストレージクラスを定義して使用して、ストレージ I/O プロファイルを特定の種類のワークロードに一致させます。

HM component	Storage type	I/O requirement	Example optimization
HM Control Plane (CP)	Internal DBs and microservices (such as Thanos)	Moderate IOPS, High throughput	Standard SSD tier for internal state.
HM Postgres data nodes	Primary database I/O	High IOPS, low latency	Premium/high-performance SSD tier (crucial for production)

スナップショットクラスオプショナルの CSI 対応バックエンドのみ

ボリュームをプロビジョニングするためのストレージクラスに加えて、CSI ドライバーがスナップショットをサポートしている場合、Kubernetes `VolumeSnapshotClass` を構成することもできます。

<<<<<<< 私たちのもの

- これは、 `PersistentVolume` スナップショットを作成および管理する方法を定義します。
- 通常、1 つまたは 2 つのスナップショットクラスのみが必要ですたとえば、ストレージバックエンドごとに 1 つなど。
- TopoLVM を使用したオンプレミス スナップショットがサポートされています。

- ローカル CSI スナップショットはサポートされていません。この場合、HM のオブジェクトストアバックアップメカニズムを使用します。
-
- これは、 `PersistentVolume` スナップショットを作成および管理する方法を定義します。
- 通常、1 つまたは 2 つのスナップショットクラスのみが必要ですたとえば、ストレージバックエンドごとに 1 つなど。
- TopoLVM を使用したオンプレミス スナップショットがサポートされています。
- ローカル CSI スナップショットはサポートされていません。この場合、HM のオブジェクトストアバックアップメカニズムを使用します。
- クラウド展開またはエンタープライズオンプレミス CSI は、通常、スナップショットをサポートしています。>>>>>>> 彼らのもの

!!!注

スナップショットはオブジェクトストアのバックアップを補完しますが、それに置き換えられるものではありません。スナップショットを使用して、同じクラスター内の高速なロールバックとリカバリーを行います。長期保持またはディザスターリカバリーにはオブジェクトストアのバックアップを使用します。

ブロックストレージ戦略とオプションのスナップショット戦略の概要を示します。HM CP および Postgres データノードのワークロード用に特定されたストレージクラス前述の手順を参照してください。およびオプションの VolumeSnapshotClasses を確認すると、適切な CSI ドライバーをインストールすることにより、ブロックストレージとオプションのスナップショットを実装する準備が整います。。

クラスター前提条件

クラスターが **動的ボリュームプロビジョニング**をサポートしていることを確認します。

- クラウド展開では、これは通常デフォルトで有効になっています。
- オンプレミス展開では、選択した CSI ドライバー TopoLVM、Portworx、Ceph RBD などが適切にインストールおよび構成されていることを確認します。

CSI ドライバーのインストール

選択された StorageClass は、必要なコンテナストレージインターフェイス CSI ドライバーを指示します。これにより、クラスターは永続ボリュームを動的にプロビジョニングできます。

- **永続ストレージドライバー必須**
  - **オンプレミスプライマリユースケース**ノード接続ストレージの場合は TopoLVM などのローカル CSI ドライバー、または Portworx または Ceph RBD などのエンタープライズドライバーを使用します可能な場合.
  - **クラウド展開代替** CSPAWS EBS、GCE PD などから提供される CSI ドライバーを使用します。
- **スナップショットコントローラオプショナル、CSI 対応バックエンドのみ**

- CSI ドライバーがスナップショットをサポートしている場合、Kubernetes CSI スナップショットコントローラを有効にし、`VolumeSnapshotClass` を構成できます。これにより、オペレーションリカバリーのための高速なボリュームレベルのスナップショットが可能になります。
- **オンプレミス TopoLVM/ローカル CSI)** スナップショットはサポートされていません。データ保護のために HM のビルトインオブジェクトストアのバックアップとリストアを使用します。
- **オンプレミススナップショットサポートを備えたエンタープライズ CSI** ドライバーがサポートしている場合、スナップショットコントローラを有効にできます。
- **クラウド:** ほとんどのクラウド CSI ドライバーは、スナップショットをサポートしています。短期のロールバックにスナップショットを使用できますが、クロスクラスターリカバリーと長期保持にはオブジェクトストアバックアップを使用します。

Helm チャートの更新

選択したストレージクラスを作成したら、Helm チャート `values.yaml` を更新します。

```
global:
  storage_class: <hm-cp-storage-class>
```

KMS キーのセットアップ

透過的データ暗号化 TDE を使用する予定がある場合、 *configuring a Key Management Store* KMS が必要です。

オブジェクトストレージ構成 MinIO セットアップ

HM では、データ保護とアーカイブのために S3 互換のオブジェクトストレージ MinIO のようなが必要です。このバケットには、すべての非ブロックストレージデータが保存されます。

- Postgres WAL とバックアップポイントインタイムリカバリーの有効化。
- 管理対象ストレージの場所。
- アーカイブされたログとメトリック。

MinIO ユーザーとポリシーの作成

HM プラットフォームの専用のバケット、ユーザー、およびポリシーを設定します。

1. *MinIO Client `mc* <https://min.io/docs/minio/linux/reference/minio-mc.html>`_ をダウンロードし、MinIO インスタンスを管理するための `mc` クライアントを構成します。
2. 環境変数を定義します。括弧で囲まれた値を、目的の名前と資格情報に置き換えます。

```
export MINIO_DEPLOYMENT_NAME=<minio_deployment_name> # deployment name to MinIO
export BUCKET_NAME=<minio_bucket_name> # bucket name
export AWS_ACCESS_KEY_ID=<minio_user_name> # user name
export AWS_SECRET_ACCESS_KEY=<minio_user_password> # users password
export MINIO_POLICY_NAME=<minio_policy_name> # policy to MinIO
```

3. mc mb コマンドを使用して MinIO bucket を作成します。

4. 新しいユーザーを作成します。

```
mc admin user add ${MINIO_DEPLOYMENT_NAME} ${AWS_ACCESS_KEY_ID} ${AWS_SECRET_ACCESS_
↪KEY}
```

1. MinIO ポリシーを 1 つ作成します。

```
cat << EOF > policy.json
{
    "Version" : "2012-10-17",
    "Statement": [
        {
            "Effect" : "Allow",
            "Action" : [
                "s3:*"
            ],
            "Resource" : [
                "arn:aws:s3:::${BUCKET_NAME}",
                "arn:aws:s3:::${BUCKET_NAME}/*"
            ]
        }
    ]
}
EOF
```

1. ポリシーを適用します。

```
mc admin policy create ${MINIO_DEPLOYMENT_NAME} ${MINIO_POLICY_NAME} ./policy.json
```

1. ユーザーにポリシーをアタッチします。

```
mc admin policy attach ${MINIO_DEPLOYMENT_NAME} ${MINIO_POLICY_NAME} --user ${AWS_
↪ACCESS_KEY_ID}
```

バケットアクセスにシークレットを適用する

バケットへの HM アクセスのための専用ユーザーを準備した後、作成したユーザーを HM に関連付けてオブジェクトストレージへのアクセスを提供する次のシークレットを作成および適用します。

```
apiVersion: v1
kind: Secret
metadata:
  name: edb-object-storage # name cannot be changed
  namespace: default # namespace cannot be changed
stringData:
    auth_type: credentials
```

(次のページに続く)

(前のページからの続き)

```
    # Optional: Used only when the object storage servers certificate is not issued␣
↪by a well-known CA
    #
    # Base64 string of the CA bundle for the certificate used by the object storage␣
↪server
    aws_ca_bundle_base64: <aws_ca_bundle_base64>

    # Required: Endpoint URL to the object storage
    aws_endpoint_url_s3: <endpoint-url-to-object-storage>

    # Required: AWS Static Credentials - AWS_ACCESS_KEY_ID
    aws_access_key_id: <AWS_ACCESS_KEY_ID>

    # Required: AWS Static Credentials - AWS_SECRET_ACCESS_KEY
    aws_secret_access_key: <AWS_SECRET_ACCESS_KEY>

    # Required: Bucket name
    bucket_name: <bucket_name>

    # Required: Region of the bucket
    aws_region: <aws_region>

    # Optional: true or false
    # When server-side encryption is disabled, set this to true. By default, its␣
↪value is false, indicating that server-side encryption is enabled.
    server_side_encryption_disabled: <boolean>
```

インストールに進む

クラスターが構成されたら、 *Installation phase of Hybrid Manager on Rancher RKE2* に進みます。

Installation phase of Hybrid Manager on Rancher RKE2

Rancher RKE2 クラスターは、ハイブリッドマネージャー HM インストールの準備が整うようにほぼ事前構成されたため、Helm チャート `values.yaml` でいくつかの重要な値を確認し、インストールプロセスを実行および監視する必要があります。

最終的な Helm チャートのプリフライトチェック/最終構成

HM のインストールをトリガーする前に、Helm チャート `values.yaml` にいくつかのフィールドがあり、インストールプロセスが成功するには確認が必要です。

コンテナレジストリ URL

image sync and preconfiguration 中に、 `values.yaml` で `containerRegistryURL` を既に設定している必要があります。まだそうしていない場合は、今すぐそうしてください。

bootstrapImageName と bootstrapImageTag

事前構成のクラスター要件 中に `bootstrapImageName` および `bootstrapImageTag` はすでに設定している必要があります。まだそうしていない場合は、今すぐそうしてください。

portal_domain_name

preconifiguration 中に `parameters:global:portal_domain_name` は既に設定しているはずです。まだそうしていない場合は、今すぐそうしてください。

Transporter-rw-service.domain_name

事前構成のクラスター要件 中に `parameters.transporter-rw-service.domain_name` は既に設定しているはずです。まだそうしていない場合は、今すぐそうしてください。

Transporter-dp-agent:rw_service_url

事前構成のクラスター要件 中に `parameters.transporter-dp-agent:rw_service_url` は既に設定しているはずです。まだそうしていない場合は、今すぐそうしてください。

Transporter-data-operator:fips_enabled

すべての HM コンポーネントで FIPS をまだサポートしていないため、Transporter Data Operator のこのフィールドは `false` に設定したままにします。

upm-beacon:server_host

事前構成のクラスター要件 中に `parameters:upm-beacon:server_host` は既に設定しているはずです。まだそうしていない場合は、今すぐそうしてください。

upm-istio-gateway:cooke_aeskey

事前構成のクラスター要件 中に `parameters:upm-beacon:server_host` は既に設定しているはずです。まだそうしていない場合は、今すぐそうしてください。

upm-istio-gateway:cookie_aeskey

preconfiguration 中に `parameters:upm-istio-gateway:cookie_aeskey` は既に設定しているはずです。まだそうしていない場合は、今すぐそうしてください。

pgai:portal:authentication:staticPasswords

インストールプロセスに進む前に、`pgai:portal:authentication:staticPasswords: email` 、`hash` 、`username` 、および `userID` を設定する必要があります。

独自の IDP を使用する場合、事前構成のクラスター要件 中に pgai:portal:authentication:staticPasswords 値を既に設定している必要があります。まだそうしていない場合は、今すぐそうしてください。

ネイティブ HM ユーザーアプローチ user0 を使用する場合、 事前構成のクラスター要件 中にこれらの値を既に設定している必要があります

まだそうしていない場合は、今すぐそうしてください。

beaconAgent: プロビジョニング: ノードポートドメイン

事前構成のクラスター要件 中に beaconAgent:provisioning:nodePortDomain はすでに設定しているはずです。まだそうしていない場合は、今すぐそうしてください。

Helm チャートのインストール

すべてのプリフライトをチェックしたら、次のコマンドを使用してインストールプロセスを開始します

```
helm upgrade \
    -n edbpgai-bootstrap \
    --install \
    -f ./values.yaml \
    --version "${EDBPGAI_BOOTSTRAP_HELM_CHART_VERSION}" \
    edbpgai-bootstrap enterprisedb-edbpgai/edbpgai-bootstrap
```

インストールプロセスの監視

次のコマンドを使用して、インストールプロセスを監視します。

```
kubectl get hybridcontrolplane edbpgai -o yaml
```

インストールが完了したら、 *Updating DNS zone records* の更新に進みます。

Updating DNS zone records

Helm または Operator のインストールが完了したら、DNS ゾーンレコードを更新し、ハイブリッドマネージャーが予想通りに動作していることを確認します。

DNS ゾーンレコードの更新

1. ポータルサービスのホスト名を取得します。このホスト名は、ハイブリッドマネージャーの入力エンドポイントを表します。

```
kubectl get svc -n istio-system istio-ingressgateway -o yaml | yq .status.
↪loadBalancer.ingress[0].hostname
```

1. 次のドメイン名の DNS ゾーンに CNAME レコードを作成し、取得したホスト名をポイントします。

   - PORTAL_DOMAIN_NAME
   - TRANSPORTER_RW_SERVICE_DOMAIN_NAME

- BEACON_SERVICE_DOMAIN_NAME

各 CNAME レコードは、ユーザー向けドメイン名を入力ホスト名にマッピングします。

これで、 *connect to the portal* を実行して、インストールを確認できます。

Connecting to Hybrid Manager

EDB Postgres AI Hybrid Manager プラットフォームのインストールが成功したことを確認するには、ブラウザーを使用して Hybrid Manager コンソールに接続します。

```
https://portal.examplecompany.com
```

この構成では、ホスト名が広域インターネットに存在しないため、https は検証されません。証明書の警告を受け入れて、ポータルに進みます。

ログイン画面で、ブートストラップインストールで生成されたユーザー名とパスワードを入力できます。詳細については、Managing users を参照してください。

ログインしたら、 create a database cluster を開始できます。

5.3.3 Install Hybrid Manager on AWS EKS

Helm チャートまたは Hybrid Manager オペレーターを使用して、 Amazon EKS に Hybrid ManagerHM をインストールできます。

- *Prerequisites for AI Factory on Hybrid Manager*
- *Installation phase of Hybrid Manager on Rancher RKE2*
- *Installing using the Hybrid Manager Operator*
- *Installing using a Helm chart*
- *Connecting to the portal*
- **ハイブリッドマネージャーのアンインストール**

EKS installation prerequisites

AWS EKS には、EDB Postgres AI ソフトウェア展開をインストールする前に満たす必要がある前提条件があります。インストールプロセスを開始する前に、前提条件を満たす必要があることを確認します。

前提条件

- *A system from which you can run the installation process*
- *An AWS account with the necessary permissions*
- *An EKS cluster configured to host HM*
- *Sync container images to your registry*

次のステップ

前提条件を完了したら、 *installing the EDB Postgres AI Software Deployment on AWS EKS* に進むことができます。

Install system

インストールプロセスを実行できるシステムが必要です。このシステムには、次のソフトウェアをインストールする必要があります。

- AWS CLI —AWS のサービスを管理するための AWS コマンドラインインターフェイス CLI。
- kubectl Kubernetes クラスターを管理します。
- yq HM インストールファイルを準備します。
- Helm Kubernetes マニフェストをレンダリングおよび適用します。

リンクされた手順、または以下にある特定のオペレーティングシステムの手順を使用して、これらのツールをインストールできます。

macOS

AWS CLI、kubectl、yq、および Helm をインストールします。

```
brew install awscli kubectl yq helm
```

Linux

> **注釈**
>
> これらの手順は、デフォルトでスナップサポートが利用できることを前提としているため、Ubuntu 16.04 以降を対象としています。

```
sudo snap install aws-cli --classic
sudo snap install helm --classic
sudo snap install kubectl --classic
sudo snap install yq
```

AWS CLI をインストールします。

```
sudo yum install awscli
```

kubectl をインストールします。

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.
↪txt)/bin/linux/amd64/kubectl"
chmod +x kubectl
```

(次のページに続く)

(前のページからの続き)

```
sudo mv kubectl /usr/local/bin/
kubectl version --client
```

Helm をインストールします

```
curl -LO https://get.helm.sh/helm-v3.12.3-linux-amd64.tar.gz
tar -xvf helm-v3.12.3-linux-amd64.tar.gz
sudo mv linux-amd64/helm /usr/local/bin/helm
helm version
```

yq をインストールします。

```
sudo curl -L "https://github.com/mikefarah/yq/releases/latest/download/yq_linux_amd64
↪" -o /usr/bin/yq
sudo chmod +x /usr/bin/yq
yq --version
```

```
sudo apt update
sudo apt install snapd
sudo snap install snapd
sudo snap install aws-cli --classic
sudo snap install kubectl --classic
sudo snap install yq
```

Windows

```
winget install Amazon.awscli --source winget
winget install kubectl --source winget
winget search helm
winget install helm
winget install yq --source winget
```

1. AWS CLI をインストールします。
2. kubectl をインストールします。
3. yq をインストールします。
4. *Helm* **チャートを編集する** をインストールします。

Helm のインストール

1. Windows の Helm バイナリをダウンロードします。 Helm GitHub Releases page で、最新リリースを見つけて、help-v3.xx-windows-amd64.zip ファイルをダウンロードします。
2. Zip ファイルを解凍します。7-Zip、WinRAR、またはビルトインの Windows 抽出ユーティリティなどのツールを使用して、ダウンロードしたファイルを解凍します。

3. helm.exe を、抽出したフォルダーからパス内のディレクトリにコピーします。一般的なディレクトリには、C:\Program Files\ 、C:\Program Files (x86)\ が含まれます。または、 C:\Helm\ などのディレクトリを作成し、パスに追加できます。

4. インストールを確認します。

    1. 新しいコマンドプロンプトまたは PowerShell ウィンドウを開きます。

    2. 次のコマンドを実行して、Helm が正しくインストールされていることを確認します。

```
helm version
```

3. 出力には、Helm バージョン情報が表示されます。

次のステップ

必要なソフトウェアをインストールしたら、*setting up an AWS account with the necessary permissions* に進むことができます。

Setting up an AWS account

Hybrid Manager クラスターをインストールするには、必要なリソースを作成する権限を持つ AWS アカウントが必要です。

必要な権限

EKS クラスターを作成するには、次の権限を持つ AWS アカウントが必要です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:*",
                "sts:AssumeRole",
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:GetRole",
                "iam:GetRolePolicy",
                "iam:PutRolePolicy",
                "iam:PassRole",
                "iam:ListPolicies",
                "iam:ListRolePolicies",
                "iam:ListAttachedRolePolicies",
                "iam:ListRoleTags",
                "iam:TagRole",
                "iam:GenerateServiceLastAccessedDetails",
                "iam:ListInstanceProfilesForRole",
```

(次のページに続く)

(前のページからの続き)

```
                "iam:ListOpenIDConnectProviders",
                "iam:GetOpenIDConnectProvider",
                "iam:CreateOpenIDConnectProvider",
                "iam:TagOpenIDConnectProvider",
                "iam:GetServiceLastAccessedDetails",
                "iam:AttachRolePolicy",
                "ec2:TerminateInstances",
                "iam:deleteOpenIDConnectProvider",
                "iam:detachRolePolicy",
                "iam:DeleteRolePolicy"
            ],
            "Resource": "*"
        }
    ]
}
```

上記の権限でロールを作成し、ユーザーにアタッチして既存の権限に追加することをお勧めします。また、セキュリティチームとこれらの権限を確認して、ユースケースに適切であることを確認することをお勧めします。

AWS アカウントを作成する

AWS アカウントをまだお持ちでない場合は、 AWS website で作成できます。**Create an AWS Account** を選択し、指示に従います。

IAM ユーザーを作成する

1. AWS Management Console にログインします。
2. 検索バーで **IAM** を検索して、IAM サービスに移動します。
3. IAM ダッシュボードの左側のメニューで、 **Users** を選択し、次に **Add user** を選択します。
4. 新しいユーザーのユーザー名を入力し、 **Programmatic access** チェックボックスを選択します。
5. **Next: Permissions** を選択します。
6. ポリシーを添付します。 [**アクセス許可の設定**] ステップで、[**既存のポリシーを直接アタッチします**] を選択し、AdministratorAccess ポリシーを検索します。ポリシーを選択し、**Next: Tags** を選択します。
7. 必要に応じて、ユーザーにタグを追加し、 **Next: Review** を選択します。
8. ユーザーの詳細を確認します。 IAM ユーザーを作成するには、 **Create user** を選択します。

> **注釈**
>
> アクセスキー ID とシークレットアクセスキーを記録します。 AWS CLI を構成するために必要です。

AWS CLI の設定

AWS CLI を構成するには、最初にローカルマシンに AWS CLI をインストールする必要があります。*Install system* 前提条件の一部としてこの手順を実行します。

CLI を構成するには、次のコマンドを実行します。

```
aws configure
```

次のステップ

AWS CLI を構成すると、 *create an EKS cluster* ができます。

Creating an EKS cluster

installed your system および *configured your AWS account* を取得したら、AWS マネジメントコンソールを使用して EKS クラスターを作成できます。

ステップ 1AWS マネジメントコンソールにログインする

https://aws.amazon.com/console/ にアカウント資格情報を使用してログインして、管理コンソールにアクセスします。

検索バーに EKS と入力し、検索結果から **Elastic Kubernetes Service** を選択します。

ステップ 2 ブートストラップキットを入手する

ブートストラップキットを使用して Hybrid ManagerHM をインストールします。このキットは、EDB Software Deployment プラットフォームのインストールに使用されるスクリプトと Helm チャートのコレクションです。

次のファイルが必要です。

- *eks-fix-auto.sh*

ファイルをローカルマシン上のディレクトリにコピーし、cd をそのディレクトリにコピーします。

次に、AWS マネジメントコンソールを使用して次のことを行います。

1. クラスターを作成する AWS リージョンを選択します。
2. 選択したリージョンをメモします。これは、後のプロセスで必要になります。
3. EKS ページに移動します。検索バーに EKS と入力し、検索結果から **Elastic Kubernetes Service** を選択します。
4. EKS ページで、 **Create Cluster** を選択します。

ステップ 3 クラスターを構成する

構成オプションで

- **Quick configuration (with EKS Auto Mode) -*new** *を選択します。
- **Cluster name** フィールドに、クラスターの名前を入力し、メモします。

- **Kubernetes version** フィールドで、使用するバージョンを選択します。(EKS 自動モードのデフォルトは 1.31 です。)
- **Cluster IAM Role** フィールドでは、値 `AmazonEKSAutoClusterRole` のままにします。
- **Node IAM Role** フィールドでは、値 `AmazonEKSAutoNodeRole` のままにします。

クラスターの VPC を選択または作成します

クラスターに VPC を提供する必要があります。VPC を作成するか、既存の VPC を使用できます。

1. **Create VPC** を選択します。
2. [**Create VPC**] ページの **Resources to create** の下で、 **VPC and more** を選択します。
3. 次の情報を入力します。
    - **Name tag auto-generate** の下に、VPC の名前を入力します。
    - **Number of Availability Zones** の下で、3.
    - **パブリックサブネットの数** の下で 3.
    - 、3 が選択されていることを確認します。
    - **プライベートサブネットの数**の下で、3 が選択されていることを確認します。
    - NAT ゲートウェイで、 **In 1 AZ** を選択します
4. **Create VPC** を選択します。 [**Create VPC workflow**] ページが表示され、VPC の作成手順が実行されます。完了したら、[**View VPC**] を選択します。

VPC のビューで、リソースマップのパブリックサブネットごとに

1. サブネットにカーソルを合わせ、表示される矢印アイコンのある正方形のボックスを選択します。サブネットを編集できるページが開きます。
2. **Actions > Edit subnet settings** を選択します。
3. **Auto-assign public IPv4 address > Enable auto-assign public IPv4 address** を選択します。
4. **Save** を選択します。

各パブリックサブネットのタグを追加します。

1. **Tags** タブを選択します。
2. **Manage Tags** を選択します
3. **Add Tag** を選択します
4. **Key** フィールドに `kubernetes.io/role/elb` を入力し、 **Value** フィールドに 1 を入力します。
5. **Save** を選択します
6. タブ/ウィンドウを閉じ、VPC ビューに戻ります。

リソースマップ内の各プライベートサブネットのタグを追加します。

1. サブネットにカーソルを合わせ、表示される矢印アイコンのある正方形のボックスを選択します。

2. **Tags** タブを選択します。
3. **Manage Tags** を選択します
4. **Add Tag** を選択します
5. **Key** フィールドに `kubernetes.io/role/internal-elb` を入力し、**Value** フィールドに 1 を入力します。
6. **Save** を選択します
7. タブ/ウィンドウを閉じ、VPC ビューに戻ります。

使用する既存の VPC がある場合は、ドロップダウンから選択します。

VPC には、異なるアベイラビリティーゾーンに少なくとも 2 つのパブリックサブネットと 2 つのプライベートサブネットが必要で、次のタグが必要です。

パブリックサブネット

- `kubernetes.io/role/elb` 1

プライベートサブネット

- `kubernetes.io/role/internal-elb` 1

EKS クラスター作成ページに戻る

新しい VPC を作成した場合、EKS クラスター作成ページを更新する必要があります。ドロップダウンを更新して、作成した VPC を選択できるようにするには、**Create VPC** の横にある円形の矢印アイコンを選択します。

1. クラスターのサブネットを選択します。異なるアベイラビリティーゾーンで少なくとも 2 つのプライベートサブネットを選択する必要があります。

> **注釈**
>
> これらのサブネットは、Kubernetes コントロールプレーンと Kubernetes ワーカーノード間の通信を促進することであるため、パブリックではなくプライベートである必要があります。パブリックサブネットは、クラスターの内部を不必要なセキュリティの脅威にさらしたままにする可能性があります。サブネットのタグ付けは、正しいサブネットにイングレスロードバランサーを作成するように EKS に通知します。

2. **Create** を選択します。

クラスターを作成しています。これには数分かかる場合があります。ページの上部にあるパンくずリストで**クラスター**を選択することにより、EKS コンソールで作成をモニタできます。

ステップ 4kubectl を構成する

クラスターを作成した後、クラスターと対話するように kubectl を構成する必要があります。

ターミナルを開き、実行します。

```
aws eks --region <REGION> update-kubeconfig --name <CLUSTER-NAME>
```

そこで

- <REGION> は、クラスターを作成したリージョンです。
- <CLUSTER-NAME> は、クラスターに付けた名前です。

> **注釈**
>
> *AWS_PROFILE* 環境変数を設定していない場合は、コマンドに `--profile <profile-name>` オプションを含めます。

EKS クラスターと対話できるようになりました。テストするには、次を実行します。

```
kubectl get pods -A
```

ステップ 5S3 バケットを作成し、クラスターに接続する

HM には、バックアップやその他のデータを保存するために S3 バケットが必要です。S3 バケットを作成し、クラスターに接続します。

S3 バケットを作成する

1. AWS マネジメントコンソールで S3 サービスを開きます。検索バーに S3 と入力し、検索結果から **S3** を選択します。
2. **バケットの作成** を選択します。
3. バケット名フィールドの名前を入力します。これはグローバルに一意である必要があります。
4. **Create Bucket** を選択します。

アクセス用のロールを作成する

1. EKS ダッシュボードに戻ります。検索バーに EKS と入力し、検索結果から **Elastic Kubernetes Service** を選択します。
2. 作成したクラスターを選択します。
3. 概要の **Details** セクションで、OpenID Connect プロバイダーの URL をコピーします。
4. AWS マネジメントコンソールで IAM サービスを開きます。検索バーに IAM と入力し、検索結果から **IAM** を選択します。
5. 左側のメニューで、 **Identity Providers** を選択します。
6. **Add Provider** を選択します。
7. **Configure Provider** ページで、**OpenID Connect** を選択し、コピーした URL を **Provider URL** フィールドに貼り付けます。

8. オーディエンスフィールドに sts.amazonaws.com と入力します。

9. **Add Provider** を選択します。

緑色のバナーにメッセージ Provider created successfully が表示され、[**役割の割り当て**] ボタンが表示されます。

10. **役割の割り当て** を選択します。

11. **ロールオプション** の下で、 **Create a New Role** を選択します。**Next** を選択します。

信頼できるエンティティを選択します

1. [**信頼できるエンティティの選択**] ページで、 **Web ID** を選択します。

Web attempts セクションの **Identity Provider** フィールドには、作成したばかりのプロバイダーが表示されます。

2. **Audience** フィールドで sts.amazonaws.com を選択し、 **Next** を選択します。

新しいロールに権限を付与します。

1. **Add Permissions** ページで、検索フィールドに s3full と入力します。 AmazonS3FullAccess ポリシーを選択し、 **Next** を選択します。

2. <clustername>-s3access などのロール名を入力し、 **Create Role** を選択します。

緑色のバナーがメッセージ Role created successfully とともに表示され、[**View Role**] ボタンが表示されます。

3. **役割の表示** を選択します。

4. ロールのビューで、ロールの **ARN** 値をコピーします。

シークレットを作成する

正常に機能するには、デフォルトの名前空間-n default にシークレットを作成する必要があります。

ターミナルで、次を実行します。

```
kubectl create secret generic -n default edb-object-storage \
    --from-literal auth_type=workloadIdentity \
    --from-literal aws_region=<REGION> \
    --from-literal aws_role_arn=<ROLE-ARN> \
    --from-literal bucket_name=<BUCKET-NAME>
```

そこで

- <REGION> は、クラスターを作成するリージョンです。
- <ROLE-ARN> は、作成したロールの ARN です。
- <BUCKET-NAME> は、作成したバケットの名前です。

ステップ 6EBS CSI ドライバーと CSI スナップショットコントローラを追加する

PGAIHM でスナップショットを正しく管理するには、EBS CSI ドライバーと CSI スナップショットコントローラの 2 つの AWS CSI 関連ドライバーが必要です。

AWS EBS CSI ドライバー

EBS CSI ダイバーを使用すると、Kubernetes クラスターは、Elastic Block StoreEBS や Elastic File SystemEFS などの AWS サービスを使用して、ストレージリソースを動的にプロビジョニングおよび管理できます。この統合により、Kubernetes ワークロードは AWS ストレージサービスを永続ストレージとしてシームレスに使用できます。

EBS CSI ドライバーをインストールするには

1. PGAIHM をインストールする EKS クラスターの EKS cluster ページで、**Add-ons** タブを選択します。
2. **Get More Addons** を選択します。
3. [**アドオン**] ページの EBS CSI Driver のタイルで、チェックボックスを選択し、[**Next**] を選択します。
4. [**Configure**] ページで、[**Next**] を選択します。
5. **Review** ページで、 **Create** を選択します。

EBS CSI アドオンドライバーがクラスターに追加されます。

AWS CSI スナップショットコントローラ

CSI スナップショットコントローラは、ボリュームスナップショットを管理する Kubernetes コントローラです。

CSI スナップショットコントローラをインストールするには

1. PGAIHM をインストールする EKS クラスターの EKS cluster ページで、**Add-ons** タブを選択します。
2. **Get More Addons** を選択します。
3. [**アドオン**] ページの CSI スナップショットコントローラのタイルで、チェックボックスを選択し、[**Next**] を選択します。
4. [**Configure**] ページで、[**Next**] を選択します
5. **Review** ページで、 **Create** を選択します。

CSI スナップショットコントローラアドオンがクラスターに追加されます。

ステップ 7 互換性のためにクラスターを更新する

AWS マネジメントコンソールによって作成された EKS クラスターは、HM と完全な互換性がないため、クラスターにいくつかの変更を加える必要があります。これらの変更を行うためのスクリプトを利用できます。

Helm eks-fix-auto.sh スクリプトまたは *Operator eks-fix-auto.sh* スクリプトをダウンロードし、ターミナルで実行します。

```
$SHELL eks-fix-auto.sh
```

スクリプトは、デフォルトのノードプールとストレージクラスをワークロードにより適切なものに置き換え、その他のいくつかの設定を構成します。

ステップ 8 ハイブリッドマネージャーをインストールする

次に、 *Install the Hybrid Manager on EKS* 。

Install Hybrid Manager on EKS

ブートストラップする Helm チャートまたは Hybrid Manager オペレーターを使用して、 Amazon EKS に Hybrid ManagerHM をインストールできます。

EKS にハイブリッドマネージャーをインストールする前に、 *Prerequisites for AI Factory on Hybrid Manager* を満たす必要があります。

次に、次のいずれかの方法を使用してハイブリッドマネージャーをインストールできます。

- *Installing Hybrid Manager on AWS EKS using the Hybrid Manager Operator*
- *Installing Hybrid Manager on AWS EKS*

Hybrid Manager をインストールした後、 *Connecting to the Hybrid Manager Console* ガイドを使用して EDB Postgres AI ハイブリッドマネージャーポータルに接続できます。

Assets for Helm

default-env.sh

```
#  exports for EKS

export EDB_PLATFORM_VERSION="v1.2.0"
export EDB_TARGET_PLATFORM="eks"
export CONTAINER_REGISTRY_URI="docker.enterprisedb.com/pgai-platform"
export IMAGESET_REGISTRY_URI=$CONTAINER_REGISTRY_URI
export IMAGESET_AUTHTYPE="token"
export PORTAL_DOMAIN_NAME="portal.foo.network"
export TRANSPORTER_RW_SERVICE_DOMAIN_NAME="transporter.foo.network"
export BEACON_SERVICE_DOMAIN_NAME="beacon.foo.network"
export AUTHENTICATION_EMAIL="owner@mycompany.com"
export AUTHENTICATION_USER="owner@mycompany.com"
export LOCATION_NAME="default-location"

#  Set a password hash for the user or pass a password to have it hashed for you.

#  If you pass a password, you will need to remove it from your history.

#  You can hash a password using the following command:
```

(次のページに続く)

(前のページからの続き)

```
#  echo -n "password" | htpasswd -BinC 10 admin | cut -d: -f2

#

#  export AUTHENTICATION_PASSWORD="password"

#  If setting AUTHENTICATION_PASSWORD_HASH, ensure it is single quoted to prevent␣
↪variable expansion (e.g. $2y$10$...).

export AUTHENTICATION_PASSWORD_HASH=$2y$10$vKOAXfLHbeV1OQxMpxlLdOIwnX.JAN.ZrD9ZU//
↪ocrNQwhQIMtXhy
```

eks-fix-auto.sh

```
# !/bin/bash

#  Reprovision nodepools for HM workloads

cat <<EOF | kubectl apply -f -
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: edbpgai
spec:
  disruption:
    budgets:
      - nodes: 10%
    consolidateAfter: 30s
    consolidationPolicy: WhenEmptyOrUnderutilized
  template:
    metadata:
      labels:
        edbaiplatform.io/control-plane: "true"
    spec:
      expireAfter: 336h
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      taints:
        - key: edbaiplatform.io/control-plane
          value: "true"
          effect: NoSchedule
```

(次のページに続く)

(前のページからの続き)

```
      requirements:
        - key: karpenter.sh/capacity-type
          operator: In
          values:
            - on-demand
        - key: eks.amazonaws.com/instance-category
          operator: In
          values:
            - c
            - m
            - r
        - key: eks.amazonaws.com/instance-generation
          operator: Gt
          values:
            - "4"
        - key: eks.amazonaws.com/instance-cpu
          operator: Gt
          values:
            - "4"
        - key: kubernetes.io/arch
          operator: In
          values:
            - amd64
        - key: kubernetes.io/os
          operator: In
          values:
            - linux
      terminationGracePeriod: 24h0m0s
EOF

#  Reprovision gp2 storage class

kubectl delete storageclass gp2
cat <<EOF | kubectl apply -f -
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
  name: gp2
parameters:
  encrypted: "true"
  type: gp2
provisioner: ebs.csi.eks.amazonaws.com
reclaimPolicy: Delete
```

(次のページに続く)

(前のページからの続き)

```
volumeBindingMode: WaitForFirstConsumer
allowVolumeExpansion: true
EOF

#  Create a gp3 storage class for EKS

cat <<EOF | kubectl apply -f -
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gp3
  annotations:
    storageclass.kubernetes.io/is-default-class: "false"
provisioner: ebs.csi.eks.amazonaws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
  type: gp3
  encrypted: "true"
EOF

#  Create a VolumeSnapshotClass for EKS

cat <<EOF | kubectl apply -f -
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: ebs-snapshotclass
  annotations:
    snapshot.storage.kubernetes.io/is-default-class: "true"
driver:  ebs.csi.eks.amazonaws.com
deletionPolicy: Delete
EOF
```

eks-install-secrets.sh

```
registry="docker.enterprisedb.com"
username="pgai-platform"
password=""

echo "Enter the password for ${username}@${registry}"
read -s password

if [ -z "$password" ]; then
    echo 1>&2 "Error: password is required" >&2
```

(次のページに続く)

(前のページからの続き)

```
    echo 1>&2
    usage
    exit 1
fi

kubectl create ns edbpgai-bootstrap
kubectl create ns upm-replicator

kubectl create secret docker-registry edb-cred \
    -n edbpgai-bootstrap \
    --docker-server="${registry}" \
    --docker-username="${username}" \
    --docker-password="${password}"

kubectl create secret docker-registry edb-cred \
    -n upm-replicator \
    --docker-server="${registry}" \
    --docker-username="${username}" \
    --docker-password="${password}"

kubectl annotate secret -n upm-replicator edb-cred replicator.v1.mittwald.de/
↪replicate-to="*" --overwrite

kubectl create namespace upm-griptape

FERNET_KEY=$(dd if=/dev/urandom bs=32 count=1 2>/dev/null | base64)

kubectl apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
    name: fernet-secret
    namespace: upm-griptape
stringData:
    FERNET_KEY: ${FERNET_KEY}
EOF

kubectl create namespace upm-lakekeeper

PG_CONFOUNDING_KEY=$(dd if=/dev/urandom bs=32 count=1 2>/dev/null | base64)

kubectl apply -f - <<EOF
  apiVersion: v1
  kind: Secret
  metadata:
```

(次のページに続く)

(前のページからの続き)

```
    name: pg-confounding-key
    namespace: upm-lakekeeper
  stringData:
    PG_CONFOUNDING_KEY: ${PG_CONFOUNDING_KEY}
EOF

echo "PG_CONFOUNDING_KEY is ${PG_CONFOUNDING_KEY} - store safely"
```

prepare.sh

```
# !/bin/bash
echo "HM Helm values.yaml preparation\n"

#  If EDB_PLATFORM_VERSION is not set, error and exit

if [ -z "$EDB_PLATFORM_VERSION" ]; then
  echo "EDB_PLATFORM_VERSION is not set, please set it to the version of EDB Platform␣
↪you are deploying"
  exit
fi

#  Test for the presence of yq

if ! command -v yq &> /dev/null
then
    echo "yq could not be found, please install it"
    exit
fi

#  Import the eks-default-env file

source ./default-env.sh

#  Test for presence of helm repo

value=`helm repo list -o yaml | yq eval contains([{ "name":"edbpgai"}]) -`
if [[ "$value" == "false" ]]; then
  echo "EDB Platform Helm repository not found, adding in now"
  helm repo add edbpgai "https://downloads.enterprisedb.com/${EDB_SUBSCRIPTION_TOKEN}/
↪pgai-platform/helm/charts/"
fi

#  Always update the helm repo
```

(次のページに続く)

(前のページからの続き)

```
helm repo update
helm show values edbpgai/edbpgai-bootstrap --version $EDB_PLATFORM_VERSION > values.
↪yaml
echo "\n\n"

#  Prepare version dependent values

export AES_256_KEY=$(openssl rand -base64 32)

if [[ "$EDB_PLATFORM_VERSION" == v1.2.* ]]; then
    echo "EDB Platform version is v1.2.*"
    yq -i .bootstrapImageName = env(CONTAINER_REGISTRY_URI)+"/edbpgai-bootstrap/
↪bootstrap-"+env(EDB_TARGET_PLATFORM) |
    .bootstrapImageTag=env(EDB_PLATFORM_VERSION) |
    del(.bootstrapImage) |
    .parameters.upm-istio-gateway.cookie_aeskey = env(AES_256_KEY) |
    del(.parameters.upm-beacon-ff-base) |
    .beaconAgent.provisioning.imagesetDiscoveryContainerRegistryURL= env(IMAGESET_
↪REGISTRY_URI) |
    .containerRegistryURL = env(CONTAINER_REGISTRY_URI) values.yaml
fi

#  Prepare values.yaml for EKS

#  if AUTHENTICATION_PASSWORD_HASH is not set, generate it

if [ -z "$AUTHENTICATION_PASSWORD_HASH" ]; then
  echo "Generating password hash - please ensure you remove the password from your␣
↪history"
  export AUTHENTICATION_PASSWORD_HASH=$(echo -n $AUTHENTICATION_PASSWORD | htpasswd -
↪BinC 10 admin | cut -d: -f2)
else
  echo "Using prehashed password"
fi

yq -i .system = env(EDB_TARGET_PLATFORM) |
.parameters.global.portal_domain_name = env(PORTAL_DOMAIN_NAME) |
.parameters.transporter-rw-service.domain_name = env(TRANSPORTER_RW_SERVICE_DOMAIN_
↪NAME) |
.parameters.transporter-dp-agent.rw_service_url= "https://"+env(TRANSPORTER_RW_
↪SERVICE_DOMAIN_NAME)+"/transporter" |
.parameters.upm-beacon.server_host= env(BEACON_SERVICE_DOMAIN_NAME) |
.pgai.portal.authentication.staticPasswords[0].email=env(AUTHENTICATION_EMAIL) |
.pgai.portal.authentication.staticPasswords[0].hash=env(AUTHENTICATION_PASSWORD_HASH)␣
↪|
```

(次のページに続く)

(前のページからの続き)

```
.pgai.portal.authentication.staticPasswords[0].username=env(AUTHENTICATION_USER) |
.pgai.portal.authentication.staticPasswords[0].userID="c5998173-a605-449a-a9a5-
↪4a9c33e26df" |
.beaconAgent.location=env(LOCATION_NAME) values.yaml

echo "Your values.yaml file is ready"
```

Assets for Operator

default-env.sh

```
#  exports for EKS

export EDB_PLATFORM_VERSION="v1.2.0"
export CONTAINER_REGISTRY_URI="docker.enterprisedb.com/pgai-platform"
export IMAGESET_REGISTRY_URI=$CONTAINER_REGISTRY_URI
export IMAGESET_AUTHTYPE="token"
export PORTAL_DOMAIN_NAME="portal.foo.network"
export TRANSPORTER_RW_SERVICE_DOMAIN_NAME="transporter.foo.network"
export BEACON_SERVICE_DOMAIN_NAME="beacon.foo.network"
export AUTHENTICATION_EMAIL="owner@mycompany.com"
export AUTHENTICATION_USER="owner@mycompany.com"
export LOCATION_NAME="default-location"
export STORAGE_CLASS="gp2"
export TRANSPORTER_FIPS_ENABLED=false

#  Set a password hash for the user or pass a password to have it hashed for you.

#  If you pass a password, you will need to remove it from your history.

#  You can hash a password using the following command:

#  echo -n "password" | htpasswd -BinC 10 admin | cut -d: -f2

#

#  export AUTHENTICATION_PASSWORD="password"

#  If setting AUTHENTICATION_PASSWORD_HASH, ensure it is single quoted to prevent␣
↪variable expansion (e.g. $2y$10$...).

export AUTHENTICATION_PASSWORD_HASH=$2y$10$vKOAXfLHbeV1OQxMpxlLdOIwnX.JAN.ZrD9ZU//
↪ocrNQwhQIMtXhy
```

eks-fix-auto.sh

```
# !/bin/bash

#  Reprovision nodepools for HM workloads

cat <<EOF | kubectl apply -f -
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: edbpgai
spec:
  disruption:
    budgets:
      - nodes: 10%
    consolidateAfter: 30s
    consolidationPolicy: WhenEmptyOrUnderutilized
  template:
    metadata:
      labels:
        edbaiplatform.io/control-plane: "true"
    spec:
      expireAfter: 336h
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      taints:
        - key: edbaiplatform.io/control-plane
          value: "true"
          effect: NoSchedule
      requirements:
        - key: karpenter.sh/capacity-type
          operator: In
          values:
            - on-demand
        - key: eks.amazonaws.com/instance-category
          operator: In
          values:
            - c
            - m
            - r
        - key: eks.amazonaws.com/instance-generation
          operator: Gt
          values:
            - "4"
```

(次のページに続く)

(前のページからの続き)

```
        - key: eks.amazonaws.com/instance-cpu
          operator: Gt
          values:
            - "4"
        - key: kubernetes.io/arch
          operator: In
          values:
            - amd64
        - key: kubernetes.io/os
          operator: In
          values:
            - linux
      terminationGracePeriod: 24h0m0s
EOF

#  Reprovision gp2 storage class

kubectl delete storageclass gp2
cat <<EOF | kubectl apply -f -
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
  name: gp2
parameters:
  encrypted: "true"
  type: gp2
provisioner: ebs.csi.eks.amazonaws.com
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
allowVolumeExpansion: true
EOF

#  Create a gp3 storage class for EKS

cat <<EOF | kubectl apply -f -
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gp3
  annotations:
    storageclass.kubernetes.io/is-default-class: "false"
provisioner: ebs.csi.eks.amazonaws.com
volumeBindingMode: WaitForFirstConsumer
```

(次のページに続く)

(前のページからの続き)

```
parameters:
  type: gp3
  encrypted: "true"
EOF

#  Create a VolumeSnapshotClass for EKS

cat <<EOF | kubectl apply -f -
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: ebs-snapshotclass
  annotations:
    snapshot.storage.kubernetes.io/is-default-class: "true"
driver:  ebs.csi.eks.amazonaws.com
deletionPolicy: Delete
EOF
```

eks-install-operator.sh

```
registry="docker.enterprisedb.com"
username="pgai-platform"
password=""

echo "Enter the password for ${username}@${registry}"
read -s password

if [ -z "$password" ]; then
    echo 1>&2 "Error: password is required" >&2
    echo 1>&2
    usage
    exit 1
fi

kubectl create namespace edb-hcp-operator-system

helm repo add enterprisedb-edbpgai "https://downloads.enterprisedb.com/${password}/$
↪{RELEASE}pgai-platform/helm/charts/"
helm repo update

kubectl create secret docker-registry edb-hcp-operator-pull-secret \
    --docker-server="${registry}" \
    --docker-username="${RELEASE}${username}" \
    --docker-password="${password}" \
```

(次のページに続く)

(前のページからの続き)

```
    --namespace edb-hcp-operator-system

helm upgrade \
    -n edb-hcp-operator-system \
    --install \
    --set "imagePullSecrets[0].name=edb-hcp-operator-pull-secret" \
    --set controllerManager.manager.image.tag=$OPERATOR_VERSION \
    --set controllerManager.manager.image.repository=docker.enterprisedb.com/$
↪{RELEASE}pgai-platform/edb-hcp-operator/manager \
    --version $OPERATOR_VERSION \
    edb-hcp-operator enterprisedb-edbpgai/edb-hcp-operator
```

eks-install-secrets.sh

```
registry="docker.enterprisedb.com"
username="pgai-platform"
password=""

echo "Enter the password for ${username}@${registry}"
read -s password

if [ -z "$password" ]; then
    echo 1>&2 "Error: password is required" >&2
    echo 1>&2
    usage
    exit 1
fi

kubectl create ns edbpgai-bootstrap
kubectl create ns upm-replicator

kubectl create secret docker-registry edb-cred \
    -n edbpgai-bootstrap \
    --docker-server="${registry}" \
    --docker-username="${username}" \
    --docker-password="${password}"

kubectl create secret docker-registry edb-cred \
    -n upm-replicator \
    --docker-server="${registry}" \
    --docker-username="${username}" \
    --docker-password="${password}"

kubectl annotate secret -n upm-replicator edb-cred replicator.v1.mittwald.de/
```

(次のページに続く)

(前のページからの続き)

```
↪replicate-to="*" --overwrite

kubectl create namespace upm-griptape

FERNET_KEY=$(dd if=/dev/urandom bs=32 count=1 2>/dev/null | base64)

kubectl apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
    name: fernet-secret
    namespace: upm-griptape
stringData:
    FERNET_KEY: ${FERNET_KEY}
EOF

kubectl create namespace upm-lakekeeper

PG_CONFOUNDING_KEY=$(dd if=/dev/urandom bs=32 count=1 2>/dev/null | base64)

kubectl apply -f - <<EOF
  apiVersion: v1
  kind: Secret
  metadata:
    name: pg-confounding-key
    namespace: upm-lakekeeper
  stringData:
    PG_CONFOUNDING_KEY: ${PG_CONFOUNDING_KEY}
EOF

echo "PG_CONFOUNDING_KEY is ${PG_CONFOUNDING_KEY} - store safely"
```

hm-default.yaml

```
apiVersion: edbpgai.edb.com/v1alpha1
kind: HybridControlPlane
metadata:
  name: edbpgai
spec:
  source:
    useLocalKustomizations: true
  flavour: eks
  version: v1.2.0+appl
  imageRegistry: ${CONTAINER_REGISTRY_URI}
```

(次のページに続く)

(前のページからの続き)

```
imagePullSecrets:
  - name: edb-cred
    namespace: edbpgai-bootstrap
globalParameters:
  internal_backup_folder: 4befab698c8d
  portal_domain_name: ${PORTAL_DOMAIN_NAME}
  storage_class: ${STORAGE_CLASS}
componentsParameters:
  edb-migration-portal:
    enable_pdb: "false"
  transporter-data-operator:
    fips_enabled: "${TRANSPORTER_FIPS_ENABLED}"
  transporter-dp-agent:
    rw_service_url: https://${TRANSPORTER_RW_SERVICE_DOMAIN_NAME}/transporter
  transporter-rw-service:
    domain_name: ${TRANSPORTER_RW_SERVICE_DOMAIN_NAME}
  upm-api-pnxt-metric:
    log_level: info
    tsdb_host: thanos-query.monitoring.svc.cluster.local
    tsdb_port: "9090"
  upm-api-pnxt-pd:
    log_level: info
  upm-beaco-ff-base:
    usage_generator_interval: 24h
  upm-beaco-usages:
    usage_generator_interval: 24h
  upm-beacon:
    server_host: ${BEACON_SERVICE_DOMAIN_NAME}
    beacon_server_replicas: "1"
    external_metrics_replicas: "1"
    beacon_db_hibernation: "\"off\""
  upm-istio-gateway:
    cookie_aeskey: ${AES_256_KEY}
  upm-prometheus:
    memory: 4Gi
pgai:
  portal:
    authentication:
      idpConnectors: []
      staticPasswords:
        - email: ${AUTHENTICATION_EMAIL}
          hash: ${AUTHENTICATION_HASH}
          username: ${AUTHENTICATION_USERNAME}
          userID: ${AUTHENTICATION_USER_ID}
beaconAgent:
```

(次のページに続く)

(前のページからの続き)

```
    applianceProviderEnabled: true
    plaintext: true
    beaconServerAddress: "upm-beacon-server-ingestion.upm-beacon.svc.cluster.
↪local:8080"
    tlsInsecure: false
    location: "devspatcher"
    logLevel: "debug"
    beaconDBConnected: true
    inCluster: true
    provisioning:
      loadBalancersEnabled: true
      nodePortDomain: "nodeportdomain.com"
      imageDiscovery: true
      imagesetDiscoveryAuthenticationType: "eks_managed_identity"
      imagesetDiscoveryContainerRegistryURL: 123456079902.dkr.ecr.us-east-1.amazonaws.
↪com
    providers:
      - appliance
```

prepare-op.sh

```
# !/bin/bash
echo "HM Operator values.yaml preparation\n"

#  Import the eks-default-env file

source ./default-env.sh

#  If EDB_PLATFORM_VERSION is not set, error and exit

if [ -z "$EDB_PLATFORM_VERSION" ]; then
  echo "EDB_PLATFORM_VERSION is not set, please set it to the version of EDB Platform␣
↪you are deploying"
  exit
fi

#  Test for the presence of yq

if ! command -v yq &> /dev/null
then
    echo "yq could not be found, please install it"
    exit
fi
```

(次のページに続く)

(前のページからの続き)

```
cp hm-default.yaml hm-values.yaml

#  Prepare values.yaml for EKS

export AES_256_KEY=$(openssl rand -base64 32)

#  if AUTHENTICATION_PASSWORD_HASH is not set, generate it

if [ -z "$AUTHENTICATION_PASSWORD_HASH" ]; then
  echo "Generating password hash - please ensure you remove the password from your␣
↪history"
  export AUTHENTICATION_PASSWORD_HASH=$(echo -n $AUTHENTICATION_PASSWORD | htpasswd -
↪BinC 10 admin | cut -d: -f2)
else
  echo "Using prehashed password"
fi

if [[ "$EDB_PLATFORM_VERSION" == v1.2.* ]]; then
    echo "EDB Platform version is v1.2.*"
    yq -i .spec.imageRegistry = env(CONTAINER_REGISTRY_URI) |
    .spec.componentsParameters.upm-istio-gateway.cookie_aeskey = env(AES_256_KEY) |
    .spec.globalParameters.portal_domain_name = env(PORTAL_DOMAIN_NAME) |
    .spec.globalParameters.storage_class = env(STORAGE_CLASS) |
    .spec.componentsParameters.transporter-data-operator.fips_enabled =␣
↪(env(TRANSPORTER_FIPS_ENABLED)|tostring) |
    .spec.componentsParameters.transporter-rw-service.domain_name = env(TRANSPORTER_
↪RW_SERVICE_DOMAIN_NAME) |
    .spec.componentsParameters.transporter-dp-agent.rw_service_url= "https://
↪"+env(TRANSPORTER_RW_SERVICE_DOMAIN_NAME)+"/transporter" |
    .spec.componentsParameters.upm-beacon.server_host= env(BEACON_SERVICE_DOMAIN_
↪NAME) |
    .spec.pgai.portal.authentication.staticPasswords[0].email=env(AUTHENTICATION_
↪EMAIL) |
    .spec.pgai.portal.authentication.staticPasswords[0].hash=env(AUTHENTICATION_
↪PASSWORD_HASH) |
    .spec.pgai.portal.authentication.staticPasswords[0].username=env(AUTHENTICATION_
↪USER) |
    .spec.pgai.portal.authentication.staticPasswords[0].userID="c5998173-a605-449a-
↪a9a5-4a9c33e26df" |
    .spec.beaconAgent.provisioning.imagesetDiscoveryContainerRegistryURL=␣
↪env(IMAGESET_REGISTRY_URI) |
    .spec.beaconAgent.provisioning.imagesetDiscoveryAuthenticationType= env(IMAGESET_
↪AUTHTYPE) |
    .spec.beaconAgent.location=env(LOCATION_NAME) hm-values.yaml
fi
```

(次のページに続く)

(前のページからの続き)

```
echo "Your hm-values.yaml file is ready"
```

Installing Hybrid Manager on AWS EKS

Helm とブートストラップキットを使用して、Amazon EKS に Hybrid ManagerHM をインストールできます。

重要

インストールプロセスのこの部分を開始する前に、 *Prerequisites for AI Factory on Hybrid Manager* を完了します。

インストールするバージョンを設定します

EDB Software Deployment プラットフォームのバージョンは、環境変数 EDB_PLATFORM_VERSION によって設定されます。これは、インストールするプラットフォームのバージョンに設定する必要があります。たとえば、バージョン v1.2.0 をインストールするには、次のように環境変数を設定します。

```
export EDB_PLATFORM_VERSION="v1.2.0"
```

トークンを設定します

このインストールプロセスでは、EDB サブスクリプショントークンを環境変数として保存する必要があります。EDB Account Profile に移動して、取得できます。プロンプトが表示されたら、ログインします。そこに、Repos 2.0 トークンのエントリがあります。

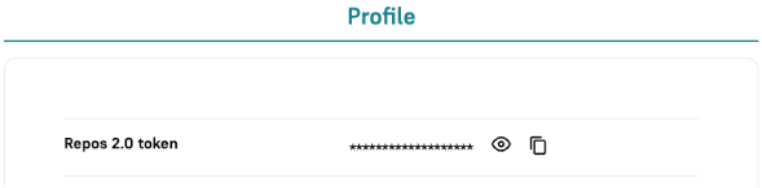


図 1: Repo Token

この値を取得して、環境変数として設定します。

```
export EDB_SUBSCRIPTION_TOKEN="your-token"
```

ブートストラップキットを入手する

Hybrid Manager は、ブートストラップキットを使用してインストールされます。このキットは、EDB Software Deployment プラットフォームのインストールに使用されるスクリプトと Helm チャートのコレクションです。

- *eks-install-secrets.sh*
- *prepare.sh*

- *default-env.sh*

すべてのファイルをローカルマシンのディレクトリにコピーし、cd をそのディレクトリにコピーします。

シークレットのインストール

ブートストラッププロセスでは、Kubernetes クラスターに多数のシークレットをインストールする必要があります。これらのシークレットは、EDB ダウンロードリポジトリで認証し、EDB ソフトウェア展開プラットフォームの資格情報を保存するために使用されます。

ブートストラッププロセスに必要なシークレットをインストールするには、install-secrets.sh スクリプトを実行します。トークンは環境変数として既に設定しているため、スクリプトにパイプできます。

```
echo $EDB_SUBSCRIPTION_TOKEN | $SHELL eks-install-secrets.sh
```

```
Enter the password for pgai-platform@docker.enterprisedb.com
Creating secret edb-cred
namespace/upm-replicator configured
secret/edb-cred configured
namespace/edbpgai-bootstrap configured
secret/edb-cred configured
secret/edb-cred annotated
PG_CONFOUNDING_KEY is 012345678901234567890ABCDEFG - store safely
Installation completed
```

このプロセスでは、ハイブリッドマネージャーを実行するために必要なシークレットを作成します。これらのシークレットの詳細については、次の場所で確認できます。

- **

A secret for Griptape and DeltaLake object storage configuration GenAI Builder のオプショナル**

- **

A secret for Catalog カタログを使用する場合のオプショナル**

- **

custom secrets for Migration Portal 移行ポータルとの内部通信を保護するオプション
も参照してください。

値ファイルの準備

ブートストラッププロセスでは、Helm チャートで多くの値を設定する必要があります。これらの値は、values.yaml というファイルに保存されます。このファイルは、 default-env.sh ファイルの変数を使用する prepare.sh スクリプトを実行して作成できます。 default-env.sh ファイルを編集して、必要な値を設定します。

これは、デフォルトの default-env.sh ファイルです。

```
#  exports for EKS

export EDB_PLATFORM_VERSION="v1.2.0"
export EDB_TARGET_PLATFORM="eks"
export CONTAINER_REGISTRY_URI="docker.enterprisedb.com/pgai-platform"
export IMAGESET_REGISTRY_URI=$CONTAINER_REGISTRY_URI
export IMAGESET_AUTHTYPE="token"
export PORTAL_DOMAIN_NAME="portal.foo.network"
export TRANSPORTER_RW_SERVICE_DOMAIN_NAME="transporter.foo.network"
export BEACON_SERVICE_DOMAIN_NAME="beacon.foo.network"
export AUTHENTICATION_EMAIL="owner@mycompany.com"
export AUTHENTICATION_USER="owner@mycompany.com"
export LOCATION_NAME="default-location"

#  Set a password hash for the user or pass a password to have it hashed for you.

#  If you pass a password, you will need to remove it from your history.

#  You can hash a password using the following command:

#  echo -n "password" | htpasswd -BinC 10 admin | cut -d: -f2

#

# export AUTHENTICATION_PASSWORD="password"

export AUTHENTICATION_PASSWORD_HASH="$2y$10$vKOAXfLHbeV1OQxMpxlLdOIwnX.JAN.ZrD9ZU//
↪ocrNQwhQIMtXhy"
```

次に、 `default-env.sh` ファイルの変数の簡単なガイドを示します。

Variable	Description
EDB_TARGET_PLATFO	インストールのターゲットプラットフォーム。 Amazon EKS にインストールするため、eks に設定します。
CONTAINER_REGISTRY_UR	使用するコンテナレジストリの URI。これは EDB ダウンロードリポジトリに設定されます。
IMAGESET_REGISTRY_URI	イメージセットレジストリの URI。これは EDB ダウンロードリポジトリに設定されます。
IMAGESET_AUTHTYPE	イメージセットレジストリの認証タイプ。 EDB ダウンロードリポジトリでの認証にトークンを使用するため、トークンに設定します。
PORTAL_DOMAIN_NAME	ドメインは、HM コンソールにアクセスするためのパブリック URL として機能します。以下の推奨事項を参照してください。
TRANSPORTER_RW_SERVICE_	Data Migration Service 内部的に`transporter`のドメイン名。以下の推奨事項を参照してください。

次のページに続く

表 5 – 前のページからの続き

Variable	Description
BEACON_SERVICE_DOMAI	ビーコンサービスのドメイン名。
AUTHENTICATION_EMAIL	作成するユーザーのメールアドレス。
AUTHENTICATION_USER	作成するユーザーのユーザー名。
AUTHENTICATION_PASSWORD	作成するユーザーのパスワード。これは、プレーンテキストとして、またはAUTHENTICATION_PASSWORD_HASH を使用したハッシュとして設定できます。
AUTHENTICATION_PASSWORD_HAS	作成するパスワードのハッシュ。
LOCATION_NAME	エージェントのロケーション名。以前はビーコンエージェントとして知られていました。

!!!note PORTAL_DOMAIN_NAME および TRANSPORTER_RW_SERVICE_DOMAIN_NAME の推奨事項

一貫性と管理を容易にするために、PORTAL_DOMAIN_NAME から TRANSPORTER_RW_SERVICE_DOMAIN_NAME を取得します。最も一般的なアプローチは、`portal` サブドメインを `dms` に置き換えることです。

次の表は、トップレベルドメイン `examplecompany.com` に基づいたさまざまな例を使用してこの関係を示しています。

PORTAL_DOMAIN_NAME	TRANSPORTER_RW_SERVICE_DOMAIN_NAME
portal.examplecompany.com	dms.examplecompany.com
portal-dev.examplecompany.com	dms-dev.examplecompany.com
portal.hm001.examplecompany.com	dms.hm001.examplecompany.com
portal-hm-us-east-001.examplecompany.com	dms-hm-us-east-001.examplecompany.com

`prepare.sh` スクリプトを実行して、 `values.yaml` ファイルを作成します。

```
$SHELL prepare.sh
```

これにより、`default-env.sh` ファイルで設定された値を使用して `values.yaml` ファイルが作成されます。

次のコマンドを使用して Helm チャートリポジトリをインストールします。

```
helm repo add edbpgai "https://downloads.enterprisedb.com/${EDB_SUBSCRIPTION_TOKEN}/
↪pgai-platform/helm/charts"
```

チャートリポジトリが追加されたら、最新の状態であることを確認します。

```
helm repo update
```

これで、次のコマンドを使用して helm リポジトリから `values.yaml` ファイルを作成できます。

```
helm show values edbpgai/edbpgai-bootstrap > values.yaml
```

これで、ブートストラッププロセスのチャートのデフォルト値を含む values.yaml ファイルが完成しました。

ブートストラッププロセス用に 2 つの値を作成する必要があります。これらは AES_256_KEY と AUTHENTICATION_PASSWORD_HASH です。次のコマンドを使用してこれらの値を作成できます。

```
export AES_256_KEY=$(openssl rand -base64 32)
```

ハッシュするパスワードがある場合は、次のコマンドで行うことができます。

```
export AUTHENTICATION_PASSWORD_HASH=$(echo -n $AUTHENTICATION_PASSWORD | htpasswd -
↪BinC 10 admin | cut -d: -f2)
```

次に、values.yaml ファイルを編集し、インストールに必要な値を設定する必要があります。values.yaml ファイルで適切なキーを見つけ、次のように値を設定します。$<name> エントリを、同じ名前の環境変数を設定した値に置き換えます。

1. bootstrapImageName を $CONTAINER_REGISTRY_URI$、続いて「$/edbpgai-bootstrap/bootstrap-$」、次にEDB_TARGET_PLATFORM を設定します。たとえば、上記の値を指定すると、これは docker.enterprisedb.com/pgai-platform/edbpgai-bootstrap/bootstrap-eks になります。
2. bootstrapImageTag を$EDB_PLATFORM_VERSION に設定します。
3. parameters.upm-istio.cookie_aeskey を$AES_256_KEY に設定します。
4. containerRegistryURL を$CONTAINER_REGISTRY_URI に設定します。
5. system を$EDB_TARGET_PLATFORM に設定します。
6. parameters.global.portal_domain_name を$PORTAL_DOMAIN_NAME に設定します。
7. parameters.transporter-rw-service.domain_name を$TRANSPORTER_RW_SERVICE_DOMAIN_NAME に設定します。
8. parameters.transporter-dp-agent.rw_service_url を https:// 、続いて$TRANSPORTER_RW_SERVICE_DOMAIN_NAME、/transporter に設定します。前に提供した例の最初の例に従って、これは https://dms.examplecompany.com/transporter になります。
9. parameters.upm-beacon.server_host を $BEACON_SERVICE_DOMAIN_NAME$ に設定します。 10.‘$pgai.portal.authentication.staticPasswords[0].email$ を `AUTHENTICATION_EMAIL に設定します。11. pgai.portal.authentication.staticPasswords[0].hash を $AUTHENTICATION_PASSWORD_HASH$ に設定します。12.‘$pgai.portal.authentication.staticPasswords[0].username$ を `AUTHENTICATION_USER に設定します。13. pgai.portal.authentication.staticPasswords[0].userID “c5998173-a605-449a-a9a5-4a9c33e26df”を設定します。 14. beaconAgent.location を$LOCATION_NAME に設定します

Helm チャートを適用する

values.yaml ファイルを準備したら、Helm チャートを適用してブートストラッププロセスを開始できます。これにより、Kubernetes クラスターに必要なリソースが作成され、ブートストラッププロセスを開始します。

```
helm upgrade -n edbpgai-bootstrap \
   --install -f ./values.yaml \
   --version "${EDB_PLATFORM_VERSION/-appl/+appl}" \
   edbpgai-bootstrap edbpgai/edbpgai-bootstrap
```

このコマンドは、Kubernetes クラスターにブートストラップコンテナをインストールします。ブートストラップコンテナは、EDB Software Deployment プラットフォームの残りの部分をインストールします。

ブートストラッププロセスに従います

ブートストラッププロセスが完了するまでに時間がかかります。ブートストラップコンテナのログを確認することにより、進行状況を追跡できます。

```
kubectl logs -f -l app=edbpgai-bootstrap -n edbpgai-bootstrap
```

ホスト名解決の構成

イングレスゲートウェイの IP アドレスを取得します。実行します

```
kubectl -n istio-system get service istio-ingressgateway -o jsonpath={.status.
↪loadBalancer.ingress[0].hostname} | strings | nslookup
```

このコマンドは、次のような出力を結果ます。

```
Server:         192.168.111.1
Address:        192.168.111.1#53

Non-authoritative answer:
Name:   k8s-istiosys-istioing-b3dc9b4f3c-81dc5c45b1d5f6d2.elb.us-east-1.amazonaws.com
Address: 52.70.180.102
```

次に、 default-env.sh で指定されたドメインが上記のコマンドから取得した IP アドレスを指すように DNS サービスまたは/etc/hosts ファイルを構成する必要があります。これらのドメインは PORTAL_DOMAIN_NAME、TRANSPORTER_RW_SERVICE_DOMAIN_NAME、および BEACON_SERVICE_DOMAIN_NAME にあります。上記の例では、それぞれ portal.examplecompany.com 、dms.examplecompany.com 、および beacon.examplecompany.com に設定されます。

DNS の構成

DNS 構成で、これらのドメインごとに、取得した IP アドレスをポイントする CNAME レコードを作成します。したがって、この例では、examplecompany.com の DNS ゾーンファイルに次のレコードを作成します。

```
cluster IN A 52.70.180.102
portal      IN CNAME cluster.examplecompany.com
transporter IN CNAME cluster.examplecompany.com
beacon      IN CNAME cluster.examplecompany.com
```

これは単純化された例です。実際の構成は、DNS プロバイダーと DNS レコードの管理方法によって異なります。

/etc/hosts の構成 DNS サービスがない場合

DNS サービスを構成していない場合は、IP アドレスをローカルの/etc/hosts ファイルに追加することを検討します。この追加により、default-env.sh ファイルで設定したドメイン名を使用してハイブリッドマネージャーコンソールにアクセスできます。

次に、次のように IP アドレスを/etc/hosts ファイルに追加できます。

```
echo "52.70.180.102 portal.examplecompany.com dms.examplecompany.com beacon.
↪examplecompany.com" | sudo tee -a /etc/hosts
```

IP アドレスを、前に取得した値に置き換えます。これは一時的な解決策であり、運用環境では使用しないでください。運用展開には、適切な DNS サービスを使用することをお勧めします。/etc/hosts ファイルの設定は、設定したマシンでのみ機能します。

次のステップ

connecting to the HM Console によるインストールを検証できるようになりました。

Installing Hybrid Manager on AWS EKS using the Hybrid Manager Operator

Hybrid Manager Operator を使用して、 Amazon EKS に Hybrid Manager をインストールできます。これは、EKS クラスターに Hybrid Manager プラットフォームを展開する Helm チャートです。オペレーターは、プラットフォームのインストール、アップグレード、アンインストールを含む、Hybrid Manager プラットフォームのライフサイクルを管理する責任を負います。

> **重要**
>
> インストールプロセスのこの部分を開始する前に、 *Prerequisites for AI Factory on Hybrid Manager* を完了します。 EKS クラスターの作成、必要なコンポーネントのインストール、および Hybrid Manager Operator の準備ができた環境の構成をステップ実行します。

インストールするバージョンを設定します

EDB Hybrid Manager プラットフォームのバージョンは、環境変数 EDB_PLATFORM_VERSION によって設定されます。これは、インストールするプラットフォームのバージョンに設定する必要があります。たとえば、バージョン v1.2.0 をインストールするには、次のように環境変数を設定します。

```
export EDB_PLATFORM_VERSION="v1.2.0"
```

トークンを設定します

このインストールプロセスでは、EDB サブスクリプショントークンを環境変数として保存する必要があります。EDB Account Profile に移動して、取得できます。プロンプトが表示されたら、ログインします。そこに、Repos 2.0 トークンのエントリがあります。

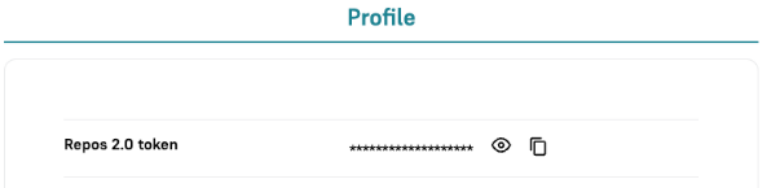


図 2: Repo Token

この値を取得して、環境変数として設定します。

```
export EDB_SUBSCRIPTION_TOKEN="your-token"
```

ブートストラップキットを入手する

Hybrid Manager は、ブートストラップキットを使用してインストールされます。このキットは、EDB Hybrid Manager プラットフォームのインストールに使用されるスクリプトとチャートのコレクションです。

- *eks-install-operator.sh* - このスクリプトは、EKS クラスターに Hybrid Manager Operator をインストールします。
- *eks-install-secrets.sh* - このスクリプトは、インストールプロセスに必要なシークレットをインストールします。
- *prepare-op.sh* - このスクリプトは、インストールプロセスの値ファイルを準備します。
- *default-env.sh* - このファイルには、インストールプロセスのデフォルトの環境変数が含まれています。
- *hm-default.yaml* - このファイルには、構成のデフォルト値が含まれています。

すべてのファイルをローカルマシンのディレクトリにコピーし、cd をそのディレクトリにコピーします。

Hybrid Manager Operator のインストール

Hybrid Manager Operator をインストールするには、`eks-install-operator.sh` スクリプトを実行します。すでにトークンを環境変数として設定しているため、スクリプトにパイプできます。また、環境変数 `OPERATOR_VERSION` をインストールするオペレーターのバージョン、たとえば `v1.2.0` に設定する必要があります。これは通常、インストールする EDB Hybrid Manager プラットフォームと同じバージョンであるため、次のように設定できます。

```
export OPERATOR_VERSION=$EDB_PLATFORM_VERSION
```

次に、スクリプトを実行してオペレーターをインストールできます。

```
echo $EDB_SUBSCRIPTION_TOKEN | $SHELL eks-install-operator.sh
```

シークレットのインストール

インストールプロセスでは、多数のシークレットを Kubernetes クラスターにインストールする必要があります。これらのシークレットは、EDB ダウンロードリポジトリで認証し、EDB ソフトウェア展開プラットフォームの資格情報を保存するために使用されます。

ブートストラッププロセスに必要なシークレットをインストールするには、`install-secrets.sh` スクリプトを実行します。トークンは環境変数として既に設定しているため、スクリプトにパイプできます。

```
echo $EDB_SUBSCRIPTION_TOKEN | $SHELL eks-install-secrets.sh
```

```
Enter the password for pgai-platform@docker.enterprisedb.com
namespace/edbpgai-bootstrap created
namespace/upm-replicator created
secret/edb-cred created
secret/edb-cred created
secret/edb-cred annotated
namespace/upm-griptape created
secret/fernet-secret created
namespace/upm-lakekeeper created
secret/pg-confounding-key created
PG_CONFOUNDING_KEY is 012345678901234567890ABCDEFG - store safely
```

このプロセスでは、ハイブリッドマネージャーを実行するために必要なシークレットを作成します。これらのシークレットの詳細については、次の場所で確認できます。

- **

A secret for Griptape and configures DeltaLake object storage GenAI Builder のオプショナル**

- **

A secret for Catalog カタログを使用する場合のオプショナル**

- **

custom secrets for Migration Portal 移行ポータルとの内部通信を保護するオプション
も参照してください。

値ファイルの準備

ブートストラッププロセスでは、Helm チャートで多くの値を設定する必要があります。これらの値は、`values.yaml` というファイルに保存されます。このファイルは、`default-env.sh` ファイルの変数を使用する `prepare.sh` スクリプトを実行して作成できます。`default-env.sh` ファイルを編集して、必要な値を設定します。

これは、デフォルトの `default-env.sh` ファイルです。

```
#  exports for EKS

export EDB_PLATFORM_VERSION="v1.2.0"
```

(次のページに続く)

(前のページからの続き)

```
export CONTAINER_REGISTRY_URI="docker.enterprisedb.com/pgai-platform"
export IMAGESET_REGISTRY_URI=$CONTAINER_REGISTRY_URI
export IMAGESET_AUTHTYPE="token"
export PORTAL_DOMAIN_NAME="portal.foo.network"
export TRANSPORTER_RW_SERVICE_DOMAIN_NAME="transporter.foo.network"
export BEACON_SERVICE_DOMAIN_NAME="beacon.foo.network"
export AUTHENTICATION_EMAIL="owner@mycompany.com"
export AUTHENTICATION_USER="owner@mycompany.com"
export LOCATION_NAME="default-location"
export STORAGE_CLASS="gp2"
export TRANSPORTER_FIPS_ENABLED=false

#  Set a password hash for the user or pass a password to have it hashed for you.

#  If you pass a password, you will need to remove it from your history.

#  You can hash a password using the following command:

#  echo -n "password" | htpasswd -BinC 10 admin | cut -d: -f2

#

#  export AUTHENTICATION_PASSWORD="password"

#  If setting AUTHENTICATION_PASSWORD_HASH, ensure it is single quoted to prevent␣
↪variable expansion (e.g. $2y$10$...).

export AUTHENTICATION_PASSWORD_HASH=$2y$10$vKOAXfLHbeV1OQxMpxlLdOIwnX.JAN.ZrD9ZU//
↪ocrNQwhQIMtXhy
```

次に、 `default-env.sh` ファイルの変数の簡単なガイドを示します。

Variable	Description
CONTAINER_REGISTRY_UR	使用するコンテナレジストリの URI。これは EDB ダウンロードリポジトリに設定されます。
IMAGESET_REGISTRY_URI	イメージセットレジストリの URI。これは EDB ダウンロードリポジトリに設定されます。
IMAGESET_AUTHTYPE	イメージセットレジストリの認証タイプ。 EDB ダウンロードリポジトリでの認証にトークンを使用するため、トークンに設定します。
PORTAL_DOMAIN_NAME	ドメインは、HM コンソールにアクセスするためのパブリック URL として機能します。以下の推奨事項を参照してください。
TRANSPORTER_RW_SERVICE_	Data Migration Service 内部的に`transporter`のドメイン名。以下の推奨事項を参照してください。

次のページに続く

表 7 – 前のページからの続き

Variable	Description
BEACON_SERVICE_DOMAI	ビーコンサービスのドメイン名。
AUTHENTICATION_EMAIL	作成するユーザーのメールアドレス。
AUTHENTICATION_USER	作成するユーザーのユーザー名。
AUTHENTICATION_PASSWORD	作成するユーザーのパスワード。これは、プレーンテキストとして、またはAUTHENTICATION_PASSWORD_HASH を使用したハッシュとして設定できます。
AUTHENTICATION_PASSWORD_HAS	作成するパスワードのハッシュ。
LOCATION_NAME	エージェントのロケーション名。以前はビーコンエージェントとして知られていました。
STORAGE_CLASS	インストールに使用するストレージクラス。これは、AWS の EBS ボリュームの gp2 に設定されます。
TRANSPORTER_FIPS_ENABLE	Transporter サービスの FIPS モードを有効にするかどうか。

!!!note PORTAL_DOMAIN_NAME および TRANSPORTER_RW_SERVICE_DOMAIN_NAME の推奨事項

一貫性と管理を容易にするために、PORTAL_DOMAIN_NAME から TRANSPORTER_RW_SERVICE_DOMAIN_NAME を取得します。最も一般的なアプローチは、`portal` サブドメインを `dms` に置き換えることです。

次の表は、トップレベルドメイン `examplecompany.com` に基づいたさまざまな例を使用してこの関係を示しています。

PORTAL_DOMAIN_NAME	TRANSPORTER_RW_SERVICE_DOMAIN_NAME
portal.examplecompany.com	dms.examplecompany.com
portal-dev.examplecompany.com	dms-dev.examplecompany.com
portal.hm001.examplecompan	dms.hm001.examplecompany.com
portal-hm-us-east-001.examplecompany.com	dms-hm-us-east-001.examplecompany.com

`prepare-op.sh` スクリプトを実行して、 `values.yaml` ファイルを作成します。

```
$SHELL prepare-op.sh
```

これにより、`default-env.sh` ファイルで設定された値を使用して `values.yaml` ファイルが作成されます。

ブートストラッププロセス用に 2 つの値を作成する必要があります。これらは AES_256_KEY と AUTHENTICATION_PASSWORD_HASH です。次のコマンドを使用してこれらの値を作成できます。

```
export AES_256_KEY=$(openssl rand -base64 32)
```

ハッシュするパスワードがある場合は、次のコマンドで行うことができます。

```
export AUTHENTICATION_PASSWORD_HASH=$(echo -n $AUTHENTICATION_PASSWORD | htpasswd -
↪BinC 10 admin | cut -d: -f2)
```

次に、values.yaml ファイルを編集し、インストールに必要な値を設定する必要があります。values.yaml ファイルで適切なキーを見つけ、次のように値を設定します。$<name> エントリを、同じ名前の環境変数を設定した値に置き換えます。

1. spec.componentsParameters.upm-istio-gateway.cookie_aeskey を$AES_256_KEY に設定します。
2. spec.imageRegistry を$CONTAINER_REGISTRY_URI に設定します。
3. spec.globalParameters.portal_domain_name を$PORTAL_DOMAIN_NAME に設定します。
4. spec.componentsParameters.transporter-rw-service.domain_name を$TRANSPORTER_RW_SERVICE_DOMAIN_NAME に設定します。
5. parameters.transporter-dp-agent.rw_service_url を https:// 、続いて$TRANSPORTER_RW_SERVICE_DOMAIN_NAME、/transporter に設定します。前に提供した例の最初の例に従って、これは https://dms.examplecompany.com/transporter になります。
6. spec.componentsParameters.upm-beacon.server_host を$BEACON_SERVICE_DOMAIN_NAME に設定します。
7. spec.pgai.portal.authentication.staticPasswords[0].email を$AUTHENTICATION_EMAIL に設定します。
8. spec.pgai.portal.authentication.staticPasswords[0].hash を$AUTHENTICATION_PASSWORD_HASH に設定します。
9. spec.pgai.portal.authentication.staticPasswords[0].username を$AUTHENTICATION_USER に設定します。
10. spec.pgai.portal.authentication.staticPasswords[0].userID “c5998173-a605-449a-a9a5-4a9c33e26df”を設定します。
11. spec.beaconAgent.location を$LOCATION_NAME に設定します
12. spec.globalParameters.storage_class を$STORAGE_CLASS に設定します。
13. spec.componentsParameters.transporter-data-operator.fips_enabled を$TRANSPORTER_FIPS_ENABLED に設定します。ブール値は文字列として処理されるため、引用符で囲まれていることを確認します。

hm-values.yaml ファイルには、インストールプロセスに必要な値が含まれています。

Hybrid Manager プラットフォームのインストール

次の手順は、前の手順で作成した値ファイルをハイブリッドマネージャーオペレーターに適用することです。これにより、EKS クラスターへの Hybrid Manager プラットフォームのインストールがトリガーされます。

```
kubectl apply -f ./hm-values.yaml
```

次を実行して、インストールのステータスを確認できます。

```
kubectl get hybridcontrolplanes.edbpgai.edb.com
```

完了すると、出力はこれに似ているはずです。

```
NAME      PHASE
edbpgai   deployed
```

デプロイしない場合は、次を実行します。

```
kubectl get hybridcontrolplanes.edbpgai.edb.com -o yaml
```

インストールのステータスの詳細については、Hybrid Manager Operator のログを確認できます。

ホスト名解決の構成

イングレスゲートウェイの IP アドレスを取得します。実行します

```
kubectl -n istio-system get service istio-ingressgateway -o jsonpath={.status.
↪loadBalancer.ingress[0].hostname} | strings | nslookup
```

このコマンドは、次のような出力を結果ます。

```
Server:         192.168.111.1
Address:        192.168.111.1#53

Non-authoritative answer:
Name:   k8s-istiosys-istioing-b3dc9b4f3c-81dc5c45b1d5f6d2.elb.us-east-1.amazonaws.com
Address: 52.70.180.102
```

次に、 `default-env.sh` で指定されたドメインが上記のコマンドから取得した IP アドレスを指すように DNS サービスまたは`/etc/hosts` ファイルを構成する必要があります。これらのドメインは `PORTAL_DOMAIN_NAME`、`TRANSPORTER_RW_SERVICE_DOMAIN_NAME`、および `BEACON_SERVICE_DOMAIN_NAME` にあります。上記の例では、それぞれ `portal.examplecompany.com`、`dms.examplecompany.com`、および `beacon.examplecompany.com` に設定されます。

DNS の構成

DNS 構成で、これらのドメインごとに、取得した IP アドレスをポイントする CNAME レコードを作成します。したがって、この例では、examplecompany.com の DNS ゾーンファイルに次のレコードを作成します。

```
cluster IN A 52.70.180.102
portal      IN CNAME cluster.examplecompany.com
transporter IN CNAME cluster.examplecompany.com
beacon      IN CNAME cluster.examplecompany.com
```

これは単純化された例です。実際の構成は、DNS プロバイダーと DNS レコードの管理方法によって異なります。

/etc/hosts の構成 DNS サービスがない場合

DNS サービスを構成していない場合は、IP アドレスをローカルの/etc/hosts ファイルに追加することを検討します。この追加により、default-env.sh ファイルで設定したドメイン名を使用してハイブリッドマネージャーコンソールにアクセスできます。

次に、次のように IP アドレスを/etc/hosts ファイルに追加できます。

```
echo "52.70.180.102 portal.examplecompany.com dms.examplecompany.com beacon.
↪examplecompany.com" | sudo tee -a /etc/hosts
```

IP アドレスを、前に取得した値に置き換えます。これは一時的な解決策であり、運用環境では使用しないでください。運用展開には、適切な DNS サービスを使用することをお勧めします。/etc/hosts ファイルの設定は、設定したマシンでのみ機能します。

次のステップ

connecting to the HM Console によるインストールを検証できるようになりました。

Connecting to the Hybrid Manager Console

EDB ソフトウェア展開プラットフォームの **インストールとセットアップ** が成功したことを確認するには、ブラウザーを使用して HM コンソールに接続します。

ブラウザーに URL を入力して、ポータルに接続します。

```
https://portal.examplecompany.com
```

この構成では、ホスト名がより広範なインターネットに存在しないため、https は検証されません。証明書の警告を受け入れて、ポータルに進みます。

ログイン画面では、

Helm values file または *Operator values file* を介してブートストラップインストールで生成されたユーザー名とパスワードを入力できます。詳細については、*Managing users in Hybrid Manager* を参照してください。

ログインしたら、 *Create a database cluster template* ができます。

Uninstall Hybrid Manager from EKS

EKS クラスターからのハイブリッドマネージャーのアンインストールは、インストール方法によって異なります。

Bootstrap Helm チャートを使用してインストールされたハイブリッドマネージャーのアンインストール

Bootstrap Helm チャートを使用してインストールされたハイブリッドマネージャーをアンインストールすることはできません。ハイブリッドマネージャーとすべての関連リソースを削除するには、EKS クラスター自分自身を削除する必要があります。

Hybrid Manager Operator を使用してインストールされた Hybrid Manager のアンインストール

Hybrid Manager Operator インストールをアンインストールするには、次のコマンドを実行できます。

```
#  by default, EDB Postgres AI is protected from the deletion, disable the protection␣
↪first

kubectl annotate hybridcontrolplane edbpgai biganimal.enterprisedb.io/deletion-
↪protect=disabled --overwrite

#  uninstall the EDB Postgres AI

kubectl delete hybridcontrolplane edbpgai --wait=false

#  monitor the uninstallation process

kubectl get hybridcontrolplane edbpgai -o yaml
```

これにより、Hybrid Manager Operator とすべての関連リソースが EKS クラスターから削除されます。

アンインストール後

Hybrid Manager をアンインストールした後、EKS クラスターには、インストールの副作用として作成されたリソースがまだ保持されていることに注意してください。これらのリソースは、必要がなくなった場合は手動で削除したり、将来の使用のために保存したり、EKS クラスター自分自身を削除したりできます。これにより、それに関連付けられているすべてのリソースが削除されます。

5.3.4 Install Hybrid Manager on GCP GKE

Google Cloud GKE に Hybrid ManagerHM をインストールできます。

- *Prerequisites for AI Factory on Hybrid Manager*
- *Installation phase of Hybrid Manager on Rancher RKE2*
- *Connecting to the portal*

Prerequisites

GCP GKE には、 EDB Postgres AI ソフトウェア展開をインストールする前に満たす前提条件があります。インストールプロセスを開始する前に、前提条件を満たす必要があることを確認します。

前提条件

- *A system from which you can run the installation process*
- 必要な権限を持つ GCP アカウント
- *A GCP cluster configured to host HM*
- *Sync container images to your registry*

次のステップ

前提条件を完了したら、 *Installation phase of Hybrid Manager on Rancher RKE2* に進むことができます。

Install system

インストールプロセスを実行できるシステムが必要です。このシステムには、次のソフトウェアをインストールする必要があります。

- kubectl Kubernetes クラスターを管理します。
- Helm Kubernetes マニフェストをレンダリングおよび適用します。

リンクされた手順、または以下にある特定のオペレーティングシステムの手順を使用して、これらのツールをインストールできます。

macOS

kubectl と Helm をインストールします。

```
brew install kubectl helm
```

Linux

=== 「Ubuntu」

注釈

これらの手順は、デフォルトでスナップサポートが利用できることを前提としているため、Ubuntu 16.04 以降を対象としています。

```
sudo snap install helm --classic
sudo snap install kubectl --classic
```

=== 「Red Hat」

kubectl をインストールします。

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.
↪txt)/bin/linux/amd64/kubectl"
chmod +x kubectl
sudo mv kubectl /usr/local/bin/
kubectl version --client
```

Helm をインストールします

```
curl -LO https://get.helm.sh/helm-v3.12.3-linux-amd64.tar.gz
tar -xvf helm-v3.12.3-linux-amd64.tar.gz
sudo mv linux-amd64/helm /usr/local/bin/helm
helm version
```

=== “Debian”

```
sudo apt update
sudo apt install snapd
sudo snap install snapd
sudo snap install kubectl --classic
```

Windows

=== “Winget”

```
winget install kubectl --source winget
winget search helm
winget install helm
```

=== “手動”

1. kubectl をインストールします。
2. *Helm* **のインストール** をインストールします。

Helm のインストール

1. Windows の Helm バイナリをダウンロードします。 Helm GitHub Releases page で、最新リリースを見つけて、help-v3.xx-windows-amd64.zip ファイルをダウンロードします。
2. Zip ファイルを解凍します。7-Zip、WinRAR、またはビルトインの Windows 抽出ユーティリティなどのツールを使用して、ダウンロードしたファイルを解凍します。
3. `helm.exe` を、抽出したフォルダーからパス内のディレクトリにコピーします。一般的なディレクトリには、`C:\Program Files\` 、`C:\Program Files (x86)\` が含まれます。または、 `C:\Helm\` などのディレクトリを作成し、パスに追加できます。
4. インストールを確認します。
    1. 新しいコマンドプロンプトまたは PowerShell ウィンドウを開きます。
    2. 次のコマンドを実行して、Helm が正しくインストールされていることを確認します。

```
helm version
```

3. 出力には、Helm バージョン情報が表示されます。

次のステップ

必要なソフトウェアをインストールしたら、必要な権限を持つ GCP アカウントの設定に進むことができます。

Creating or configuring a GKE cluster for Hybrid Manager

GKE クラスター

Hybrid ManagerHM をインストールするには GKE クラスターが必要です。必要に応じて、GKE ドキュメントの Creating a regional cluster に関するドキュメントに従って、新しいものをセットアップします。

アクセス許可

`roles/container.admin` に `Kubernetes Engine Admin` 権限があることを確認します。

さまざまなノードタイプ

HM は、GKE 上のさまざまなノードのセットでの管理サービスハイブリッドマネージャーと PostgreSQL ワークロードの実行をサポートしていますが、追加のノードプールが必要です。

さまざまなノードセットを実装するには、次の手順を実行します。

ハイブリッドマネージャーノード

ハイブリッドマネージャーノードを、独自のノードタイプで実装できる論理グループを作成するには

1. ラベルで、次を設定します。

```
edbaiplatform.io/control-plane: "true"
```

1. テイントで、次を設定します。

```
key: edbaiplatform.io/control-plane
value: "true"
effect: NoSchedule
```

PostgreSQL ワークロードノード

Postgres ワークロードノードを、独自のノードタイプで実装できる論理グループにするには

1. ラベルで、次を設定します。

```
edbaiplatform.io/postgres: "true"
```

1. テイントで、次を設定します。

```
key: edbaiplatform.io/postgres
value: "true"
effect: NoSchedule
```

GKE 管理対象ノードグループのラベルとテイントを追加する方法の詳細については、ドキュメント here を参照してください。

オブジェクトストレージの構成

オブジェクトストレージバックエンド HM バックアップをサポートするには、クラスターを専用のオブジェクトストレージソリューション STORAGE_CLASS 、例、GCP バケットに接続します。

特に、ハイブリッドマネージャーはこのバケットを使用して以下を保存します。

- 管理対象ストレージの場所
- Postgres WAL + バックアップ
- ログ Postgres と内部サービスの両方
- メトリック Postgres と内部サービスの両方

GCP バケットを作成する

gcloud CLI をダウンロードしてインストールします。

1. GCP バケットを作成するための環境変数を設定します。

```
export GCP_PROJECT_ID="appliance-dev-444915" # change me to your own gcp project id
export GCP_BUCKET_NAME="tq-upm45906" # change me to your preferred bucket name
export GCP_BUCKET_REGION="us" # change me to your preferred location id
```

1. HM に使用する GCP バケットをホストするプロジェクトを設定します。

```
gcloud config set project $GCP_PROJECT_ID
```

1. バケットを作成します。

```
gcloud storage buckets create gs://$GCP_BUCKET_NAME --location=$GCP_BUCKET_REGION
```

バケットの階層名前空間が無効になっていることを確認します。

バケットユーザーを作成する

専用バケットを作成した後、バケットにアクセスするために必要な権限 roles/storage.admin を持つハイブリッドマネージャーのサービスアカウントを作成します。

1. 必要な環境変数を設定します。

```
export SERVICE_ACCOUNT_NAME="tq-upm45906" # change this to your preferred service␣
↪account name
```

(次のページに続く)

(前のページからの続き)

```
export SERVICE_ACCOUNT_EMAIL=$SERVICE_ACCOUNT_NAME@$GCP_PROJECT_ID.iam.
↪gserviceaccount.com
```

1. サービスアカウントを作成します。

```
gcloud iam service-accounts create $SERVICE_ACCOUNT_NAME
```

1. バケットのサービスアカウントにストレージ管理ロールを割り当てます。

```
gcloud storage buckets add-iam-policy-binding gs://$GCP_BUCKET_NAME \
  --member="serviceAccount:$SERVICE_ACCOUNT_EMAIL" \
  --role="roles/storage.admin"
```

1. 次のユーザー権限で json キーを作成します。

```
gcloud iam service-accounts keys create key-file.json \
  --key-file-type=json \
  --iam-account=$SERVICE_ACCOUNT_EMAIL
```

1. base64 エンコードされた資格情報を取得します。

```
export GCP_CREDENTIAL_BASE64=$(base64 -w0 < key-file.json)
```

```
export GCP_CREDENTIAL_BASE64=$(base64 < key-file.json)
```

バケットアクセスのシークレットを作成する

バケットへの HM アクセスのための専用ユーザーを準備した後、作成したユーザーをハイブリッドマネージャーに関連付けるシークレットを作成して、オブジェクトストレージへのアクセスを提供します。

```
apiVersion: v1
kind: Secret
metadata:
  # The name must be edb-object-storage
  name: edb-object-storage
  # The namespace must be default
  namespace: default
stringData:
  # Specifies the cloud provider. In this case, it is set to gcp.
  provider: gcp
  # The region where the GCP bucket is located.
  location_id: ${GCP_BUCKET_REGION}
  # The ID of the GCP project.
  project_id: ${GCP_PROJECT_ID}
  # The name of the GCP bucket.
  bucket_name: ${GCP_BUCKET_NAME}
```

(次のページに続く)

(前のページからの続き)

```
  # The base64 encoded JSON credentials for accessing the GCP bucket.
  credentials_json_base64: ${GCP_CREDENTIAL_BASE64}
```

AES-256 キーを作成する

認証用の AES-256 キーを生成します。

```
export AES_256_KEY=$(openssl rand -base64 32)
```

コンテナイメージの同期

続行する前に、*Sync container images to your registry* の説明に従って、ハイブリッドマネージャーコンテナイメージをプライベートレジストリに同期していることを確認してください。これはインストールの必須前提条件です。

HM に必要な名前空間と必要なシークレットを作成する

プリインストールのこのフェーズでは、HM に必要な名前空間と、それに必要なシークレットを作成します。名前空間とシークレットの両方がインストールプロセスで使用されます。

1. 必要なコンテナレジストリ環境変数を設定します。

```
export CONTAINER_REGISTRY_URI=<local-registry-uri>
export CONTAINER_REGISTRY_USERNAME=<container-registry-username>
export CONTAINER_REGISTRY_PASSWORD=<container-registry-password>
```

1. LakeKeeper の Postgres 交絡キーと Griptape の fernet キーを生成および設定します。

```
PG_CONFOUNDING_KEY=$(dd if=/dev/urandom bs=32 count=1 2>/dev/null | base64)
FERNET_KEY=$(dd if=/dev/urandom bs=32 count=1 2>/dev/null | base64)
```

1. 必要な名前空間を作成します。

```
kubectl create ns edbpgai-bootstrap
kubectl create ns upm-replicator
kubectl create ns upm-lakekeeper
kubectl create namespace upm-griptape
```

1. HM、UPM-レプリケーター、LakeKeepr、および Griptape のシークレットを作成します。

```
kubectl create secret docker-registry edb-cred \
    -n edbpgai-bootstrap \
    --docker-server="${CONTAINER_REGISTRY_URI}" \
    --docker-username="${CONTAINER_REGISTRY_USERNAME}" \
    --docker-password="${CONTAINER_REGISTRY_PASSWORD}"
```

```
kubectl create secret docker-registry edb-cred \
    -n upm-replicator \
    --docker-server="${CONTAINER_REGISTRY_URI}" \
    --docker-username="${CONTAINER_REGISTRY_USERNAME}" \
    --docker-password="${CONTAINER_REGISTRY_PASSWORD}"
```

```
kubectl create secret generic pg-confounding-key \
--from-literal=PG_CONFOUNDING_KEY=${PG_CONFOUNDING_KEY} \
--namespace=upm-lakekeeper
```

```
kubectl create secret generic fernet-secret \
--from-literal=FERNET_KEY=${FERNET_KEY} \
--namespace=upm-griptape
```

1. シークレットに注釈を付けます。

```
kubectl annotate secret -n upm-replicator edb-cred replicator.v1.mittwald.de/
↪replicate-to="*" --overwrite
```

グリップテープのシークレット

グリップテープシークレットの設定については、*following documentation* を参照してください。

カタログシークレット

カタログシークレットの設定については、 following documentation を参照してください。

シークレットがすべて設定されたので、インストールに必要な *setting up the environmental variables* に進むことができます。

Set the necessary environmental variables for install

ドメイン変数を設定する

Hybrid ManagerHM クラスターは、インストール中に特定のユーザードメインへのアクセスが必要です。

1. HM のドメイン名の変数を定義します。このドメインは、HM コンソールにアクセスするためのパブリック URL として機能します。

```
export PORTAL_DOMAIN_NAME=<your-hm-portal-domain-name>
```

たとえば、会社の名前が **Example Company** で、トップレベルドメインが `examplecompany.com` の場合、EDB はポータルドメイン名がこれらの例と同様のパターンに従うことをお勧めします。

```
portal.examplecompany.com
portal-dev.examplecompany.com
portal.hm001.examplecompany.com
portal-hm-us-east-001.examplecompany.com
```

1. Data Migration Service ドメイン名の変数内部で transporter と呼ばれますを設定します。

```
export TRANSPORTER_RW_SERVICE_DOMAIN_NAME=dms<your-hm-portal-domain-name> # Customer␣
↪domain name for the EDB Postgres AI Platform Transporter RW Service
```

EDB では、この一貫性により、サービスの管理が容易になるため、ポータルドメイン名から DMS サービスドメイン名を取得することをお勧めします、たとえば、portal を dms に置き換えます。前の例に従って

```
dms.examplecompany.com
dms-dev.examplecompany.com
dms.hm001.examplecompany.com
dms-hm-us-east-001.examplecompany.com
```

1. EDB Postgres AI エージェントドメイン名の変数を設定します。

```
export BEACON_SERVICE_DOMAIN_NAME=<beacon service domain name> # Customer domain name␣
↪for the EDB Postgres AI Beacon Service
```

ストレージクラス変数を設定します

HM クラスターには、事前定義されたストレージクラスが必要です。

```
export STORAGE_CLASS=<storage class>
```

1 つのオプションは、デフォルトの GKE ストレージクラス standard-rwo を使用することです。

認証変数を設定する

HM のインストールにはいくつかの認証値が必要です。

1. メールアドレスの変数を設定します。

```
export AUTHENTICATION_EMAIL=<your-email>
```

1. パスワードハッシュの変数を設定します。取得したハッシュ化されたパスワード。例 echo "mypassword" | htpasswd -BinC 10 admin | cut -d: -f

```
export AUTHENTICATION_HASH=<your-password-hash>
```

1. ユーザー名の変数を設定します。 Owner MyCompany はデフォルトのユーザー名です。

```
export AUTHENTICATION_USERNAME=<Owner MyCompany>
```

1. ユーザー ID を設定します。

```
export AUTHENTICATION_USER_ID="your-unique-user-identifier"
```

オプション FIPs モードを設定します Transporter Data Operator

オプションで、Transporter Data Operator の FIPS モードを設定できます。

```
export TRANSPORTER_FIPS_ENABLED=true # Enable FIPS mode for Transporter Data Operator
```

Hybrid Manager installation methods for Google Cloud GKE

GCP GKE にハイブリッドマネージャーをインストールするために使用できる方法は 2 つあります。ユースケースに合った方法を選択してください

- *Helm チャートを編集する* を使用する 。 Hybrid Manager でのテストと実験には Helm インストールを使用します。Helm インストールは、非実稼働のユースケースに適しています。
- *Install Hybrid Manager on Google Cloud GKE using EDB Postgres AI Operator* を使用する。運用でハイブリッドマネージャーを使用する準備ができている場合は、 EDB Postgres AI Operator インストールを使用します。

Install Hybrid Manager on GCP GKE using Helm

Helm インストールの環境変数を設定する

1. インストールの準備をするには、コンテナレジストリの URI を指定します。

```
export CONTAINER_REGISTRY_URI=<local-registry-URI>
```

2. edb-bootstrap イメージバージョンを設定します。

```
export EDBPGAI_BOOTSTRAP_IMAGE_VERSION="<edb-bootstrap-image-version"
```

<edb-bootstrap-image-version> の値を、このインストールのターゲットとしているイメージバージョンに変更してください。

3. edb-bootstrap Helm チャートバージョンを設定します。

```
export EDBPGAI_BOOTSTRAP_HELM_CHART_VERSION="<edbpgai-bootstrap-helm-version>"
```

<edbpgai-boostrap-helm-version> の値を、このインストールのターゲットとしている edb-pgai Helm チャートバージョンに変更してください。

4. EDB Cloudsmith トークンを設定します。

```
export EDB_CLOUDSMITH_TOKEN=<edb-cloudsmith-token>
```

5. edb-cloudsmith-token を EDB Cloudsmith トークンに変更します。

> **注釈**
>
> Hybrid Manager バージョン v1.1.0 以降の場合、変数 `upm-istio.cooke_aeskey` および `upm-istio.openshift_console_domain_name` を使用します。 HM バージョンの正しい変数を使用していること

を確認し、[Helm upgrade command](Helm チャートを使用してハイブリッドマネージャーをインストールします) の他のパラメーターを削除します。

環境変数が設定されたら、インストールの準備をできます。

インストールの準備

1. EDB Cloudsmith から Helm チャートリポジトリを追加します。

```
helm repo add enterprisedb-edbpgai "https://downloads.enterprisedb.com/${EDB_
↪CLOUDSMITH_TOKEN}/pgai-platform/helm/charts/"
```

2. 追加したリポジトリを更新します。

```
helm repo update
```

3. デフォルトの edbpgai-bootstrap Helm `values.yaml` ファイルを取得します。

```
helm show values enterprisedb-edbpgai/edbpgai-bootstrap > values.yaml
```

4. `values.yaml` ファイルを正しい値に編集し、必要に応じて前に設定した環境変数を置き換えます。

```
system: "gke"
containerRegistryURL: "${CONTAINER_REGISTRY_URI}"
bootstrapImageName: "${CONTAINER_REGISTRY_URI}/edbpgai-bootstrap/bootstrap-gke"
bootstrapImageTag: "${EDBPGAI_BOOTSTRAP_IMAGE_VERSION}"
  global:
    default_ingress_domain: "${DEFAULT_INGRESS_DOMAIN}"
    portal_domain_name: "${PORTAL_DOMAIN_NAME}"
  transporter-rw-service:
    domain_name: "${TRANSPORTER_RW_SERVICE_DOMAIN_NAME}"
  transporter-dp-agent:
    rw_service_url: "https://${TRANSPORTER_RW_SERVICE_DOMAIN_NAME}/transporter"
  transporter-data-operator:
    fips_enabled: ${TRANSPORTER_FIPS_ENABLED}
  upm-beacon:
    server_host: "${BEACON_SERVICE_DOMAIN_NAME}"
  upm-istio-gateway:
    cookie_aeskey: "${AES_256_KEY}"
pgai:
  portal:
    authentication:
      staticPasswords:
        - email: "${AUTHENTICATION_EMAIL}"
          hash: "${AUTHENTICATION_HASH}"
          username: "${AUTHENTICATION_USERNAME}"
          userID: "${AUTHENTICATION_USER_ID}"
```

動的画像検出を有効にする

レジストリからの動的イメージの検出を有効にするには、必要なオプションで構成する必要があります。これを行う方法の詳細な手順については、Image discovery configuration を参照してください。

必要なイメージ検出構成を特定した後、適切な設定で `values.yaml` を更新します。

たとえば、次は、 "basic" 認証を使用して us-east1 リージョンで Google Artifact Registry を使用するサンプル構成です。

```
beaconAgent:
  location: "devspatcher"
  beaconDBConnected: true
  plaintext: true
  logLevel: "debug"
  inCluster: true
  provisioning:
    provider: "gcp"
    imageDiscovery: true
    loadBalancersEnabled: true
    imagesetDiscoveryAuthenticationType: "basic"
    imagesetDiscoveryContainerRegistryURL: "us-east1-docker.pkg.dev"
  providers:
    - appliance
beaconServer:
  logLevel: "debug"
```

Helm チャートを使用してハイブリッドマネージャーをインストールする

すべての準備ができたら、edbpgai-bootstrap Helm チャートを使用してハイブリッドマネージャーをインストールできます。

```
helm upgrade \
    -n edbpgai-bootstrap \
    --install \
    -f ./values.yaml \
    --version "${EDBPGAI_BOOTSTRAP_HELM_CHART_VERSION}" \
    edbpgai-bootstrap enterprisedb-edbpgai/edbpgai-bootstrap
```

次のコマンドを使用して、インストールプロセスを監視できます。20〜30 分ほどかかります。

```
kubectl logs -f -l app=edbpgai-bootstrap -n edbpgai-bootstrap
```

インストールが正常に完了すると、 *update DNS zone records and verify the install* .

Install Hybrid Manager on Google Cloud GKE using EDB Postgres AI Operator

インストールの準備

1. オペレーターを取得するための認証には、次の環境変数が必要です。

```
export EDB_CLOUDSMITH_TOKEN=<edb-cloudsmith-token>
```

1. オペレーターの新しい名前空間を作成します。

```
kubectl create namespace edb-hcp-operator-system
```

1. オペレーターの Helm チャートリポジトリを追加し、更新します。

```
helm repo add enterprisedb-edbpgai "https://downloads.enterprisedb.com/${EDB_
↪CLOUDSMITH_TOKEN}/pgai-platform/helm/charts/"
helm repo update
```

1. 適切なバージョン、この場合 1.2 を使用してインストールを続行します。

```
export OPERATOR_VERSION=1.2.0
```

GKE にオペレーターをインストールする

ネットワークのユースケースの指示を使用して、オペレーターをインストールします。

1. GKE に EDB Postgres AI Operator をインストールするには、最初に EDB Cloudsmith リポジトリにアクセスするためのイメージプルシークレットを準備していることを確認します。また、リポジトリ docker.enterprisedb.com/pgai-platform/edb-hcp-operator/manager からイメージをプルするために必要な権限があることを確認します。

```
export DOCKER_SERVER="docker.enterprisedb.com"
export DOCKER_USERNAME="<your-docker-user-name>"
export DOCKER_PASSWORD="<your-docker-password>"
```

1. オペレーターのイメージプルシークレットを作成します。

```
kubectl create secret docker-registry edb-hcp-operator-pull-secret \
    --docker-server="${DOCKER_SERVER}" \
    --docker-username="${DOCKER_USERNAME}" \
    --docker-password="${DOCKER_PASSWORD}" \
    --namespace edb-hcp-operator-system
```

1. オペレーターをインストールします。

```
helm upgrade \
    -n edb-hcp-operator-system \
    --install \
    --set "imagePullSecrets[0].name=edb-hcp-operator-pull-secret" \
```

(次のページに続く)

(前のページからの続き)

```
    --set controllerManager.manager.image.tag=$OPERATOR_VERSION \
    --set controllerManager.manager.image.repository=docker.enterprisedb.com/pgai-
↪platform/edb-hcp-operator/manager \
    --version $OPERATOR_VERSION \
    edb-hcp-operator enterprisedb-edbpgai/edb-hcp-operator
```

オペレーターを使用してハイブリッドマネージャーをインストールする

YAML を適用してインストールする

EDB Postgres AI Operator を GKE クラスターにインストールした後、Operator を使用して Hybrid Manager をインストールする必要があります。次の YAML ファイルは、ハイブリッドマネージャーのインストールを開始します。

```
apiVersion: edbpgai.edb.com/v1alpha1
kind: HybridControlPlane
metadata:
  name: edbpgai
spec:
  source:
    useLocalKustomizations: true
  flavour: gke
  version: v1.2.0+appl
  imageRegistry: ${CONTAINER_REGISTRY_URI}
  imagePullSecrets:
    - name: edb-cred
      namespace: edbpgai-bootstrap
  globalParameters:
    internal_backup_folder: 4befab698c8d
    portal_domain_name: ${PORTAL_DOMAIN_NAME}
    storage_class: ${STORAGE_CLASS}
  componentsParameters:
    edb-migration-portal:
      enable_pdb: "false"
    transporter-data-operator:
      fips_enabled: "${TRANSPORTER_FIPS_ENABLED}"
    transporter-dp-agent:
      rw_service_url: https://${TRANSPORTER_RW_SERVICE_DOMAIN_NAME}/transporter
    transporter-rw-service:
      domain_name: ${TRANSPORTER_RW_SERVICE_DOMAIN_NAME}
    upm-api-pnxt-metric:
      log_level: info
      tsdb_host: thanos-query.monitoring.svc.cluster.local
      tsdb_port: "9090"
    upm-api-pnxt-pd:
```

(次のページに続く)

(前のページからの続き)

```
    log_level: info
  upm-beaco-ff-base:
    usage_generator_interval: 24h
  upm-beaco-usages:
    usage_generator_interval: 24h
  upm-beacon:
    server_host: ${BEACON_SERVICE_DOMAIN_NAME}
    beacon_server_replicas: "1"
    external_metrics_replicas: "1"
    beacon_db_hibernation: "\"off\""
  upm-istio-gateway:
    cookie_aeskey: ${AES_256_KEY}
  upm-prometheus:
    memory: 4Gi
pgai:
  portal:
    authentication:
      idpConnectors: []
      staticPasswords:
        - email: ${AUTHENTICATION_EMAIL}
          hash: ${AUTHENTICATION_HASH}
          username: ${AUTHENTICATION_USERNAME}
          userID: ${AUTHENTICATION_USER_ID}
beaconAgent:
  applianceProviderEnabled: true
  plaintext: true
  beaconServerAddress: "upm-beacon-server-ingestion.upm-beacon.svc.cluster.
↪local:8080"
  tlsInsecure: false
  location: "devspatcher"
  logLevel: "debug"
  beaconDBConnected: true
  inCluster: true
  provisioning:
    loadBalancersEnabled: true
    nodePortDomain: "nodeportdomain.com"
    imageDiscovery: true
    imagesetDiscoveryAuthenticationType: "basic"
    imagesetDiscoveryContainerRegistryURL: "us-east1-docker.pkg.dev"
  providers:
    - appliance
```

YAML が適用された後、次のコマンドを使用してインストールステータスを監視できます。

```
kubectl get hybridcontrolplane edbpgai -o yaml
```

インストールが現在いるフェーズに応じて、3 つの異なるステータスを返すことができます。

```
- deployed # the installation was successful

- deploying # the installation is still in progress

- failed: # the install was unsuccessful
```

インストールが成功すると、 *updating DNS zone records and verify the install* .

ハイブリッドマネージャーのアンインストール

EDB Postgres AI Operator 対 *Helm* チャートを編集する を使用して Hybrid Manager をインストールした場合、アンインストールできます。

> **警告**
>
> 次の手順は、Hybrid Manager をアンインストールします。 Hybrid Manager をアンインストールする場合にのみ、この手順を使用します。

ハイブリッドマネージャーを削除する必要がある場合があります。ハイブリッドマネージャーはデフォルトで削除から保護されているため、最初にその保護を無効にする必要があります。

```
kubectl annotate hybridcontrolplane edbpgai biganimal.enterprisedb.io/deletion-
↪protect=disabled --overwrite
```

次に、アンインストールできます。

```
kubectl delete hybridcontrolplane edbpgai --wait=false
```

プロセスをモニタするには、次を実行します。

```
kubectl get hybridcontrolplane edbpgai -o yaml
```

Updating DNS zone records

Helm または Operator のインストールが完了したら、DNS ゾーンレコードを更新し、ハイブリッドマネージャーが予想通りに動作していることを確認します。

DNS ゾーンレコードの更新

1. ポータルサービスのホスト名を取得します。このホスト名は、ハイブリッドマネージャーの入力エンドポイントを表します。

```
kubectl get svc -n istio-system istio-ingressgateway -o yaml | yq .status.
↪loadBalancer.ingress[0].hostname
```

1. 次のドメイン名の DNS ゾーンに CNAME レコードを作成し、取得したホスト名をポイントします。

- PORTAL_DOMAIN_NAME
- TRANSPORTER_RW_SERVICE_DOMAIN_NAME
- BEACON_SERVICE_DOMAIN_NAME

各 CNAME レコードは、ユーザー向けドメイン名を入力ホスト名にマッピングします。

これで、 *connect to the portal* を実行して、インストールを確認できます。

Connecting to Hybrid Manager

EDB Postgres AI Hybrid Manager プラットフォームのインストールが成功したことを確認するには、ブラウザーを使用して Hybrid Manager コンソールに接続します。

```
https://portal.examplecompany.com
```

この構成では、ホスト名が広域インターネットに存在しないため、https は検証されません。証明書の警告を受け入れて、ポータルに進みます。証明書の警告を受け入れて、ポータルに進みます。

ログイン画面で、ブートストラップインストールで生成されたデフォルトのユーザー名とパスワード `owner@mycompany.com` および `password` を入力できます。

ログインしたら、 *Create a database cluster template* ができます。

5.3.5 Customizing your installations

インストールをカスタマイズすることにより、ハイブリッドマネージャーの追加機能を有効にできます。

- ユーザー管理のための Configure your own identity provider (IDP) 。
- 透過的データ暗号化 TDE で使用する Set up Key Management Store (KMS)
- *TLS Certificate managment*
- AI エージェントをビルドするための *Enabling GenAI Builder on Hybrid Manager* 。
- *Enabling data catalogs on Hybrid Manager* 分析ワークロードの単一のテーブルでデータを管理します。
- *Migration Portal* **のシークレットを理解する** 内部コンポーネント間の安全な通信のための。

Sync container images to your registry

Hybrid Manager ソフトウェアスタックは、EDB Repos 2.0 レジストリを介して配布されます。 Hybrid Manager をインストールまたはアップグレードする前に、これらのコンテナイメージを独自の安全で承認された内部レジストリに同期する必要があります。

なぜ画像を同期するのですか？

同期プロセスにより、次のことが保証されます。

- **セキュリティ**イメージは組織の承認されたレジストリでホストされています
- **不変性** SHA256 ダイジェストは環境全体で保存されます

- **コンプライアンス** コンテナイメージの保存場所を制御します
- **可用性**外部レジストリが利用できない場合でも、イメージはアクセス可能なままです

前提条件

画像を同期する前に、次のことを確認します。

- インストールする Hybrid Manager のバージョン
- 読み取り権限を持つ EDB Repos 2.0 アクセストークン
- プッシュ権限による宛先コンテナレジストリへのアクセス
- システムにインストールされている edbctl または skopeo

オプション 1edbctl の使用推奨

edbctl は、ハイブリッドマネージャーリソースを管理するための CLI ツールで、イメージを同期する最も簡単な方法を提供します。

edbctl のインストール

edbctl を使用する前に、システムにインストールする必要があります。CLI ツールのインストールと構成の詳細な手順については、edbctl installation guide を参照してください。

プラットフォームイメージの同期

必要な環境変数を構成し、 sync コマンドを実行します。

```
#  Configure the HM release version

export HM_RELEASE=<RELEASE_VERSION>

#  Configure the EDB Repos 2.0 access token

export EDB_TOKEN=<EDB_TOKEN>

#  Configure the EDB Repos 2.0 registry source

export EDB_SOURCE_REGISTRY=pgai-platform

#  Run the sync-to-local-registry command

edbctl image sync-to-local-registry \
    --destination-registry-uri "<LOCAL_REGISTRY_URI>" \
    --version "${HM_RELEASE}" \
    --source-registry-username "${EDB_SOURCE_REGISTRY}" \
    --source-registry-password "${EDB_TOKEN}" \
```

(次のページに続く)

(前のページからの続き)

```
    --destination-registry-username "<LOCAL_REGISTRY_USER>" \
    --destination-registry-password "<LOCAL_REGISTRY_PWD>"
```

プレースホルダーを置き換えます。

- <RELEASE_VERSION> HM バージョン例 v1.3.4
- <EDB_TOKEN> EDB Repos 2.0 アクセストークン
- <LOCAL_REGISTRY_URI> 宛先レジストリ URI
- <LOCAL_REGISTRY_USER> 宛先レジストリのユーザー名
- <LOCAL_REGISTRY_PWD> 宛先レジストリのパスワード

オペレーター画像を同期する

```
#  Sync the HM Operator image

edbctl operator sync-to-local-registry \
    --destination-registry-uri "<LOCAL_REGISTRY_URI>" \
    --version "${HM_RELEASE}" \
    --source-registry-username "${EDB_SOURCE_REGISTRY}" \
    --source-registry-password "${EDB_TOKEN}" \
    --destination-registry-username "<LOCAL_REGISTRY_USER>" \
    --destination-registry-password "<LOCAL_REGISTRY_PWD>"
```

オプション 2skopeo を使用する

Skopeo は、コンテナイメージを操作するためのオープンソースツールです。edbctl を使用したくない場合は、それを使用して画像を同期できます。

skopeo のインストール

ご使用のオペレーティングシステムの official skopeo installation instructions に従ってください。

画像リストをダウンロードする

Hybrid Manager の各リリースは、必要なすべてのコンテナイメージのリストを含むアーティファクトを提供します。

```
#  Configure the HM release version

export HM_RELEASE=<RELEASE_VERSION>

#  Configure the EDB Repos 2.0 access token

export EDB_TOKEN=<EDB_TOKEN>
```

(次のページに続く)

(前のページからの続き)

```
#  Download the image list artifact

curl -sLO "https://downloads.enterprisedb.com/${EDB_TOKEN}/pgai-platform/raw/names/$
↪{HM_RELEASE}-images.txt/versions/${HM_RELEASE}/images.txt"
```

レジストリへの認証

```
#  Configure registries

export EDB_SOURCE_REGISTRY=docker.enterprisedb.com/pgai-platform
export LOCAL_REGISTRY_URI=<LOCAL_REGISTRY_ADDRESS>

#  Login to EDB Repos 2.0 source registry

skopeo login docker.enterprisedb.com

#  Login to destination registry

skopeo login <LOCAL_REGISTRY_ADDRESS>
```

すべての画像を同期する

```
#  Parse the image list and sync each image

while read -r image; do
  skopeo --override-os linux copy \
    --multi-arch all \
    docker://$EDB_SOURCE_REGISTRY/${image/:*@/@} \
    docker://$LOCAL_REGISTRY_URI/${image/:*@/@} \
    --retry-times 3
done < images.txt
```

オペレーター画像を同期する

```
#  Sync the HM Operator image

skopeo --override-os linux copy \
    --multi-arch all \
    docker://${EDB_SOURCE_REGISTRY}/edb-hcp-operator/manager:${HM_RELEASE} \
    docker://${LOCAL_REGISTRY_URI}/edb-hcp-operator/manager:${HM_RELEASE} \
    --retry-times 3
```

!!!note ネットワーク速度とリリース内のイメージの数によっては、同期プロセスが完了するまでに数分かかります。

プラットフォーム固有の注意事項

AWS ECR

ECR は、名前に複数のスラッシュを含むイメージを同じリポジトリに保存することをサポートしていないため、宛先レジストリとして AWS ECR を使用する場合、イメージごとに別個のリポジトリを作成する必要があります。

edbctl で sync コマンドを実行すると、ECR を検出して確認を求め、AWS CLI コマンドのコードスニペットを提供して、必要なすべてのリポジトリを事前に作成します。

認証には、次を使用します。

```
- -destination-registry-username "AWS" \
- -destination-registry-password "$(aws ecr get-login-password --region ${AWS_REGION})
↪"
```

詳細は、 AWS ECR documentation を参照してください。

GCP アーティファクトレジストリ

GCP Artifact Registry 認証の場合、次を使用します。

```
- -destination-registry-username "_json_key" \
- -destination-registry-password "$(cat ~/gcp-service-account-key.json)"
```

Red Hat Quay / OpenShift

標準のユーザー名とパスワード、またはトークン認証がサポートされています。OpenShift 内部レジストリの場合、適切なサービスアカウントトークンが構成されていることを確認します。

次のステップ

コンテナイメージをプライベートレジストリに同期した後

1. レジストリをポイントするように Helm チャート `values.yaml` を更新します。

```
containerRegistryURL: "<your-local-container-registry-url>"
```

2. ご使用のプラットフォームのハイブリッドマネージャーのインストールを続行します。- *Installation phase of Hybrid Manager on Rancher RKE2*

- *Installation phase of Hybrid Manager on Rancher RKE2*
- *Installation phase of Hybrid Manager on Rancher RKE2*
- *Installation phase of Hybrid Manager on Rancher RKE2*

Configuring your own identity provider

本番展開の場合、LDAP、SAML、または別の supported Dex connector のような外部 IdP を構成することを強くお勧めします。これにより、堅牢で安全なユーザー管理が保証されます。次にいくつかの例を示します。

- *Configuring LDAP*
- *Configuring SAML*

テストまたはデモ環境の場合、creating additional native users による操作を簡素化できます。

Configuring LDAP

Dex 認証と連携するように LDAP を構成して、エンドユーザーが LDAP サーバーの資格情報を使用してハイブリッドマネージャー HM コンソールにログインできるようにします。この例では、LDAP 構成に Okta を使用します。

LDAP 統合のための Okta ユーザーの作成

Okta にログインし、読み取り専用の管理者ロールで新しいユーザーを作成します。Dex はこのユーザーを使用して、Okta LDAP インターフェイスと通信します。

Okta は、クライアント統合のための LDAP インターフェイスを提供します。詳細については、

Okta LDAP Interface documentation を参照してください。

LDAP コネクタを介して Okta に接続するように Dex を構成する

Okta LDAP 構成のエントリーを追加し、作成した読み取り専用の管理者ユーザーのユーザー名とパスワード BindDN および BindPW を指定します。使用するインストール方法に応じて、`values.yaml` ファイルまたは `HybridControlPlane` カスタムリソースにエントリーを追加します。

```
portal:
  authentication:
    idpConnectors:
      - type: ldap
        name: okta
        id: oktaLdap
        config:
          host: <ldap_server>:<ldap_server_port>
          insecureNoSSL: false
          insecureSkipVerify: false
          bindDN: uid=<okta_admin@example.com>, dc=<org_subdomain>, dc=<domain>,␣
↪dc=com
          bindPW: <okta_admin_password>
          usernamePrompt: Email Address
          userSearch:
            baseDN: ou=users, dc=<org_subdomain>, dc=<domain>, dc=com
            filter: "(objectclass=inetOrgPerson)"
            username: mail
```

(次のページに続く)

(前のページからの続き)

```
          idAttr: DN
          emailAttr: mail
          nameAttr: cn
        groupSearch:
          baseDN: ou=groups, dc=<org_subdomain>, dc=<domain>, dc=com
          filter: "(objectclass=groupOfUniqueNames)"
          userMatchers:
            - userAttr: memberOf
              groupAttr: uniqueMember
          nameAttr: cn
  staticPasswords:
    - [...]
```

パラメーター

この表は、 `portal.authentication` 構成で必要な ID プロバイダー設定を示しています。環境に基づいてカスタマイズが必要なパラメーターを表示します。ここで記載されていない他のパラメーターは、提供された例に示すように構成できます。

Parameter	Description
idpConnectors	認証に使用する IdP コネクタごとにデータの配列を入力します。詳細なガイダンスについては、公式 *Troubleshooting AI Factory on Hybrid Manager* を参照してください。
idpConnectors.config	LDAP サーバーへの接続の構成。 `<ldap_server>:<ldap_server_port>`形式を使用します。
idpConnectors.config.bindDN	Okta で作成した読み取り専用の管理ユーザーの識別名。Dex が LDAP サーバーとの通信に使用します。また、Dex が LDAP サーバー内のこのユーザーを検索する場所を指定します。
idpConnectors.config.bindPW	Okta で作成した読み取り専用の管理ユーザーのパスワード。Dex が LDAP サーバーとの通信に使用します。
idpConnectors.config.usernameProm	ログイン画面のユーザー名/電子メール入力フィールドの上に表示されるテキスト。
idpConnectors.config.userSearch.bas	Dex が LDAP サーバー内のユーザーを検索する場所を指定します。 Okta LDAP インスタンスの管理者は、ユーザーの baseDN を知っています。
idpConnectors.config.groupSearch.ba	Dex が LDAP サーバー内のグループを検索する場所を指定します。 Okta LDAP インスタンスの管理者は、ユーザーの baseDN を知っています。
idpConnectors.config.groupSearch.ba	ユーザーがグループのメンバーであるかどうかを Dex が判断する方法を定義します。ユーザーエントリとグループエントリの間で属性を一致させます。

Dex 構成の適用

認証設定を変更した後、Dex 構成を調整し、Dex サービスを再起動して変更を有効にする必要があります。手順については、Applying Dex を参照してください。

新しい所有者を割り当て、ネイティブユーザーを無効にします

HM コンソールで新しい組織の所有者を割り当てた後、ネイティブユーザーを無効にします。

Disabling a native user を参照してください。

Configuring SAML

Dex 認証と連携するように SAML を構成して、エンドユーザーが SAML サーバー資格情報を使用してハイブリッドマネージャー HM コンソールにログインできるようにします。この例では、SAML 構成に Okta を使用します。

SAML アプリケーションの作成

1. Okta にログインし、新しいアプリ統合を作成します。
2. **SAML 2.0** を選択します。
3. アプリ名を割り当て、その他の一般設定を指定し、 **Next** を選択します。
4. **シングルサインオン URL** (ACS URL) と **Audience URI** フィールドの両方に Dex コールバック URL を入力します。フォーマットは `https://<hybrid_manager_URL>/auth/callback` です。
5. ユーザー名と電子メールの **属性ステートメント**をマッピングします。例

Name	Name format	Value
name	Unspecified	“user.firstName+” “+user.lastName”
email	Unspecified	user. email

1. アプリケーションが作成されたら、HM にアクセスするユーザーまたはグループを、作成したアプリ統合に割り当てます。これは、[**Assignments**] タブで行うことができます。

SAML コネクタを介して Okta に接続するように Dex を構成する

通信の Okta および Dex エンドポイントを指定する Okta SAML 構成のエントリーを追加します。使用するインストール方法に応じて、`values.yaml` ファイルまたは `HybridControlPlane` カスタムリソースにエントリーを追加します。

```
portal:
  authentication:
    idpConnectors:
      - type: saml
        name: Okta
        id: okta
        config:
          ssoURL: <https://saml.example.com/sso>
          redirectURI: <https://<hybrid_manager_URL>/auth/callback>
          usernameAttr: name
```

(次のページに続く)

(前のページからの続き)

```
        emailAttr: email
        groupsAttr: groups
        caData: <dNekV4TURSYUZ3MHpOREV3T01Rc3dDUVlEVlFRR1CRU=>
  staticPasswords:
    - [...]
```

パラメーター

この表は、portal.authentication 構成における重要な ID プロバイダー設定を示しています。ご使用の環境に合わせてカスタマイズが必要なパラメーターのみをリストしています。ここで詳しく説明されていない他のパラメーターは、提供された例に示すように構成できます。

Parameter	Description
idpConnectors	認証に使用する IdP コネクタごとにデータの配列を入力します。詳細なガイダンスについては、公式 *Troubleshooting AI Factory on Hybrid Manager* を参照してください。
idpConnectors.config.ssoURL	シングルサインオン URL は、Dex が認証要求を送信するエンドポイントです。 Okta サインオン構成から取得できます。
idpConnectors.config.redirectURI	ハイブリッドマネージャーの ACS URL。これは、Okta SAML インスタンスが応答を送信するエンドポイントです。 SAML アプリケーションの作成時に指定した Dex コールバック URL と同じです。
idpConnectors.config.usernameAttr	返信のどの属性にユーザーのユーザー名が含まれるかを Dex に指示します。値は、SAML アプリケーションの作成時に指定した属性名と一致する必要があります。
idpConnectors.config.email	返信のどの属性にユーザーのメールアドレスが含まれるかを Dex に指示します。値は、SAML アプリケーションの作成時に指定した属性名と一致する必要があります。
idpConnectors.config.groupsAttr	応答のどの属性に、ユーザーが属しているグループに関する情報が含まれているかを Dex に示します。
idpConnectors.config.caData	Okta アプリケーションの SAML 構成からの証明書。これにより、HM は SAML サーバーからの応答を認証できます。 Okta サインオン構成、SAML セットアップオプションから取得できます。

Dex 構成の適用

認証設定を変更した後、Dex 構成を調整し、Dex サービスを再起動して変更を有効にする必要があります。Applying Dex を参照してください。

新しい所有者の割り当てとネイティブユーザーの無効化

HM コンソールで新しい組織の所有者を割り当てた後、ネイティブユーザーを無効にします。

Disabling a native user を参照してください。

TLS Certificate managment

ハイブリッドマネージャーポータルのカスタム x.509 証明書を設定する

ハイブリッドマネージャー HM ポータルが使用する自己署名 x.509 証明書を独自に変更できます。

独自の証明書をインポートするには、サーバー証明書とプライベートキーを含む Kubernetes シークレットを作成する必要があります。

証明書を生成する

証明書例 `my-certificate.crt` とプライベートキー例 `my-certificate.key` の base64 エンコード文字列を生成します。

```
cat my-certificate.crt | base64
cat my-certificate.key | base64
```

yaml ファイルを作成する

次の内容を含むファイル例 `my-secret.yaml` を作成し、`<base64 encoded string>` を前の手順で生成された値に置き換えます。

```
apiVersion: v1
data:
  tls.crt: <base64 encoded string>
  tls.key: <base64 encoded string>
kind: Secret
metadata:
  name: my-portal-certificate
  namespace: default
  annotations:
    replicator.v1.mittwald.de/replicate-to: istio-system
type: kubernetes.io/tls
```

Kubernetes クラスターにシークレットを適用する

```
kubectl apply -n default -f ./my-secret.yaml
```

Helm チャートを編集する

`parameters.global` の下に入力変数 `portal_certificate_secret` を追加し、その値を作成したばかりのシークレットの名前に設定することにより、Helm value.yaml ファイルを編集します。

例

```
parameters:
  global:
    portal_certificate_secret: "my-portal-certificate"
```

Helm チャートの更新

Helm upgrade コマンドを実行して、HM に変更を適用します。

```
helm upgrade \
    -n edbpgai-bootstrap \
    --install \
    --f my-values.yaml
```

警告

Helm Chart のインストールに使用した入力変数のセットを常に提供してください。2 回目の Helm upgrade コマンドの実行でカスタム入力のいずれかを指定しない場合、その値はデフォルトに設定されます。

シークレットのフォーマットの詳細については、 Istio documentation を参照してください。

HM ポータルのカスタム cert-manager 発行者を設定する

HM ポータルの証明書は、HM の内部証明書マネージャーでサポートされている x.509 発行者のいずれかを使用して生成および管理することもできます。Let’s Encrypt 証明書の ACME 発行者。

cert-manager Web サイトから直接選択した発行者のドキュメントに従って、設定できます。

注釈

発行者ではなくクラスター発行者を設定することをお勧めします。発行者を設定する場合は、istio-system 名前空間で作成する必要があります。

ClusterIssuer が構成されたら、 values.yaml を更新し、Helm upgrade を再度実行することにより、その名前を HM に渡すことができます。

values.yaml で編集する必要があるセクションは次のとおりです。

```
parameters:
  global:
    portal_certificate_issuer_kind: "ClusterIssuer" # Valid values are Issuer and␣
↪ClusterIssuer
    portal_certificate_issuer_name: "my-issuer" # Your Issuer name
```

最後に、 Helm upgrade コマンドを実行して、HM に変更を適用します。

```
helm upgrade \
    -n edbpgai-bootstrap \
    --install \
    --f my-values.yaml
```

この機能を利用したい場合は、parameters.global.portal_certificate_secret を設定しないようにしてください。

> **警告**
>
> Helm Chart のインストールに使用した入力変数のセットを常に提供してください。 2 回目の Helm upgrade コマンドの実行でカスタム入力のいずれかを指定しない場合、その値はデフォルトに設定されます。

独自のプライベート認証局を使用する

デフォルトでは、HM が使用するすべての証明書は、cert-manager を使用して内部認証局 CA によって署名されます。CA はインストール時に作成されます。

独自のプライベート CA を使用する場合は、このセクションの手順に従ってください。それ以外の場合、このセクションはスキップできます。

> **注釈**
>
> この方法を正常に使用するには、CA プライベートキーにアクセスする必要があります。

CA シークレットが存在しない場合は、作成します。

```
apiVersion: v1
data:
  ca.crt: <base64 encoded string> # real ca crt provided by customers
  tls.crt: <base64 encoded string> # real tls crt provided by customers
  tls.key: <base64 encoded string> # real tls key provided by customers
kind: Secret
metadata:
  name: my-custom-ca
  namespace: default
  annotations:
    replicator.v1.mittwald.de/replicate-to: cert-manager
type: kubernetes.io/tls
```

> **警告**
>
> my-custom-ca は、*global-ca-secret* を除き、選択した名前に変更できます。 次のコマンドを実行するときは、選択した名前と一致するようにしてください。

value.yaml を介して渡すことにより、helm アップグレードを実行して、新しい CA を設定できます。

edb_notranlate_8 bash helm upgrade -n edbpgai-bootstrap –install … –set parameters.global.ca_secret_name=“my-custom-ca” … ```

Enabling GenAI Builder on Hybrid Manager

GenAI Builder は、 AI エージェントをビルドできるツール Griptape の EDB の統合です。

ハイブリッドマネージャーインスタンスで GenAI Builder ランチパッドアプリケーションを有効にするには、アプリケーションのシークレットを作成し、専用のオブジェクトストレージを構成する必要があります。

GenAI Builder のシークレットの作成

`eks-install-secrets.sh` スクリプトを使用した EKS インストールの場合、キーとシークレットの作成をスキップし、*GenAI ビルダーの DataLake オブジェクトストレージの構成* 、 *CORS の設定* 、および *Fernet キーのバックアップ*に進むことができます。他の EKS セットアップとオペレーティングシステムでは、Fernet キーを使用した手動シークレットの作成が必要です。

1. Fernet キーを作成し、変数に保存します。

```
FERNET_KEY=$(dd if=/dev/urandom bs=32 count=1 2>/dev/null | base64)
```

!!!note

Fernet は、Python で使用される暗号化ライブラリです。対称的な暗号化/復号化を提供し、機密データを保存するために必要です。 .. ::

2. 統合の名前空間を作成します。

```
kubectl create namespace upm-griptape
```

3. 新しい Fernet キーを参照し、専用の名前空間に保存されるシークレットを作成します。

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
   name: fernet-secret
   namespace: upm-griptape
stringData:
   FERNET_KEY: ${FERNET_KEY}
EOF
```

GenAI ビルダーの DataLake オブジェクトストレージの構成

1. ハイブリッドマネージャー展開に使用しているオブジェクトストレージに DataLake バケットを作成します。GenAI Builder は、それを使用して、構造、ツール、インデックス付きデータを保存します。

```
aws s3 mb s3://<your-griptape-datalake-bucket-name> -region     <your-region>
```

```
gsutil mb -l <your-region> gs://<your-griptape-datalake-bucket-name>
```

プロバイダーの管理コンソールまたは CLI を使用して、Griptape DataLake バケットの一意の名前を持つバケットを作成します。

2. バケットの次の情報をキャプチャします。後で GenAI ランチパッドアプリケーションを初めて使用するときに必要になります。コンソールには、DataLake バケットの構成の入力が求められます。これには、以下が必要です。

    - `DATA_LAKE_ROOT_BUCKET` DataLake で使用するバケットの名前。
    - `DATA_LAKE_S3_ACCESS_KEY` DataLake バケットへの接続に使用される access_key
    - `DATA_LAKE_S3_SECRET_ACCESS_KEY` DataLake バケットへの接続に使用される secret_access_key
    - `DATA_LAKE_S3_ENDPOINT_UR` DataLake バケットへの接続に使用されるエンドポイント URL

CORS の設定

バケットの設定を更新して、次の CORS configuration ます。

```
[
    {
        "AllowedHeaders": [
            "*"
        ],
        "AllowedMethods": [
            "PUT",
            "POST",
            "DELETE",
            "GET",
            "HEAD"
        ],
        "AllowedOrigins": [
            "https://${PORTAL_DOMAIN_NAME}"
        ],
        "ExposeHeaders": []
    }
]
```

`https://${PORTAL_DOMAIN_NAME}` は、ハイブリッドマネージャー用に構成されたドメインです。

GCS の S3 interoperability layer により、GenAI Builder は S3 互換オブジェクトストアとして GCS を使用できます。

1. GCS コンソールの **設定** の下で、s3 の相互運用性をオンにします。
2. ストレージ管理者およびサービスアカウントトークン作成者ロールを使用して更新または create a service account を作成します。

3. サービスアカウントの HMAC key pair を作成します。

4. ハイブリッドマネージャーエンドポイントをポイントする CORS 構成を使用して構成ファイルを作成します。

```
cat cors-config.json
[
  {
    "origin": ["https://${PORTAL_DOMAIN_NAME}"],
    "method": ["GET", "PUT", "POST", "DELETE", "HEAD"],
    "responseHeader": ["*"],
    "maxAgeSeconds": 3600
  }
]
```

`https://${PORTAL_DOMAIN_NAME}` は、ハイブリッドマネージャー用に構成されたドメインです。

5. CORS 構成を、前に作成したバケットに適用します。

```
gsutil cors set cors-config.json gs://<bucket name>
```

プロバイダーの管理コンソールまたは CLI を使用して、ハイブリッドマネージャーでクロスオリジンリソース共有 CORS を構成します。

シークレットを構成し、専用バケットを作成したら、Hybrid Manager installation に進みます。

Fernet キーのバックアップ

Hybrid Manager 管理者は、Fernet キーを安全に保管し、バックアップする必要があります。

キーを取得します。

```
kubectl get secret -n upm-griptape fernet-secret -o yaml
```

キーは安全に保管してください。

Enabling data catalogs on Hybrid Manager

Hybrid ManagerHM データカタログでは、Lakehouse データストレージを管理するための追加構成が必要です。これには、Lakekeeper サービスが暗号化されたデータを保存するために使用できる交絡キーを使用して Kubernetes シークレットを作成することが含まれます。

キーを使用してシークレットを作成する

`eks-install-secrets.sh` スクリプトを使用した EKS インストールの場合、キーとシークレットを作成する必要はありません。その場合、この手順をスキップして**交絡キーのバックアップ** に進みます。

他の EKS セットアップとオペレーティングシステムでは、交絡キーを使用した手動シークレットの作成が必要です。

1. 交絡キーを作成し、変数に保存します。

```
PG_CONFOUNDING_KEY=$(dd if=/dev/urandom bs=32 count=1 2>/dev/null | base64)
```

!!!note

- 交絡キーは、少なくとも 32 バイトの長さのランダム化された文字列です。 - ハイブリッドマネージャー展開ごとに交絡キーを作成します。 .. ::

2. サービスの名前空間を作成します。

```
kubectl create namespace upm-lakekeeper
```

3. 新しい交絡キーを参照し、専用の名前空間に保存されるシークレットを作成します。

```
kubectl apply -f - <<EOF
  apiVersion: v1
  kind: Secret
  metadata:
    name: pg-confounding-key
    namespace: upm-lakekeeper
  stringData:
    PG_CONFOUNDING_KEY: ${PG_CONFOUNDING_KEY}
EOF
```

シークレットを構成したら、 Hybrid Manager installation に進みます。

交絡キーのバックアップ

Hybrid Manager 管理者は、交絡キーを安全に保管し、バックアップする必要があります。

> **警告**
>
> ディザスターシナリオでの交絡キーの損失は、Hybrid Manager データカタログによって管理される Lakehouse データにアクセスするメカニズムが存在しない状況をもたらします。代わりに、管理者は新しいキーを作成して保存し、*upm-lakekeeper /lakekeeper* ワークロードを再起動し、既存のすべてのデータカタログを削除せずに注意深くリビルドする必要があります。この手順は非常に危険であり、EDB PG AI プロフェッショナルサービスチームのサポートが必要です。

キーを取得します。

```
kubectl get secrets -n upm-lakekeeper pg-confounding-key -o yaml
```

キーは安全に保管してください。

Customizing Migration Portal secrets for secure internal communication

移行ポータルは、内部コンポーネント通信にいくつかのシークレットを使用します。これらのシークレットは、すばやく機能するインストールのためにハイブリッドマネージャーリリースでハードコーディングされていますが、運用環境の場合、カスタムシークレットでオーバーライドすることを強くお勧めします。

カスタムシークレットを使用すると、シークレットが簡単に検出されず、侵害のリスクが削減されます。

Migration Portal のシークレットを理解する

各シークレットについて、 **Description** でそれが必要な理由を説明します。 **Helm value** は、構成ファイル内のどこを参照するかを示します。 **パラメーター**は、デフォルト以外の値を設定する方法を示しています。

カスタムシークレットを構成し、デフォルトのものをオーバーライドする方法の例については、デフォルトのシークレットのオーバーライド を参照してください。

Migration Portal データベースアカウント

説明ハイブリッドマネージャーが使用するデータベースサービスアカウントを構成し、移行ポータルシステムデータとスキーマ評価結果を専用データベースに保存します。

Helm 値 `parameters.edb-migration-portal.db_secrets`

```
apiVersion: v1
kind: Secret
metadata:
  name: <db-secret-name>
  namespace: edb-migration-portal
stringData:
  username: "<db username>"
  password: "<db password>"
type: kubernetes.io/basic-auth
```

パラメーター

- `metadata.name` 後で Helm 値で参照するシークレットの名前。
- `stringData.username` データベースに接続するユーザー。また、このユーザーを `parameters.edb-migration-portal.db_owner` シークレットをインストールに適用する に入力します。
- `stringData.password` このユーザーに生成されたパスワードを割り当てます。

Migration Portal データベースのスーパーユーザーアカウント

説明ハイブリッドマネージャーが移行ポータルとコパイロットデータベースをプロビジョニングするために使用するデータベーススーパーユーザーアカウントを構成します。

Helm 値 `parameters.edb-migration-portal.db_superuser_secrets`

```
apiVersion: v1
kind: Secret
metadata:
  name: <db-superuser-secret-name>
  namespace: edb-migration-portal
stringData:
  username: "postgres"
  password: "<db superuser password>"
type: kubernetes.io/basic-auth
```

パラメーター

- metadata.name 後で Helm 値で参照するシークレットの名前。
- stringData.username 常に postgres である必要があります。
- stringData.password このユーザーに生成されたパスワードを割り当てます。

Migration Copilot Migration Portal からの HTTP アクセス

説明コパイロットと移行ポータル間の HTTP 通信の認証に使用される資格情報を定義します。

Helm 値 parameters.edb-migration-portal.copilot_secrets

```
apiVersion: v1
kind: Secret
metadata:
  name: <copilot-auth-secret>
  namespace: edb-migration-portal
stringData:
  username: "<http username>"
  password: "<http password>"
type: kubernetes.io/basic-auth
```

パラメーター

- metadata.name HTTP 要求の認証に使用されるシークレットの名前。後で Helm 値で参照します。
- stringData.username コパイロットで認証するときに使用する移行ポータルのユーザー名を選択します。
- stringData.password このユーザーに生成されたパスワードを割り当てます。

移行コパイロットおよび移行ポータルのデータベースアカウント

説明コパイロットが移行ポータルデータベースと通信できるようにします。edb-migration-copilot コンポーネントは、初期化時に、このユーザーが作成され、コパイロットデータベースに正しい所有権と権限が設定されていることを保証します。

Helm 値 parameters.edb-migration-copilot.db_secrets

```
apiVersion: v1
kind: Secret
metadata:
  name: <copilot_mp_secret_name>
  namespace: edb-migration-copilot
stringData:
  username: "<db username>"
  password: "<db password>"
type: kubernetes.io/basic-auth
```

パラメーター

- metadata.name コパイロットに移行ポータルデータベースへのアクセスを提供するために使用されるシークレットの名前。
- stringData.username 移行ポータルデータベースにアクセスするときに使用するコパイロットのユーザー名を選択します。
- stringData.password このユーザーに生成されたパスワードを割り当てます。

モニタリングと移行コパイロット

説明ハイブリッドマネージャーは、Grafana を介してコパイロット展開の状態を監視できます。

Helm 値 parameters.edb-migration-copilot.metrics_auth_secrets

```
apiVersion: v1
kind: Secret
metadata:
  name: <copilot-metrics-secret-name>
  namespace: edb-migration-copilot
stringData:
  username: <metrics-http-username>
  password: <metrics http password>
type: kubernetes.io/basic-auth
```

パラメーター

- metadata.name コパイロットの監視を許可するために使用されるシークレットの名前。
- stringData.username これを、移行ポータルからの移行コパイロット HTTP アクセス用に構成されたユーザー名とは異なるユーザー名に設定します。
- stringData.password このユーザーに生成されたパスワードを割り当てます。

デフォルトのシークレットのオーバーライド

最初に、内部コンポーネント通信用のカスタムシークレットを作成します。次に、使用しているインストール方法に応じて、values.yaml 構成ファイルまたは HybridControlPlane カスタムリソースのカスタムシークレットを参照して、値をオーバーライドします。

この例では、カスタムシークレットを作成し、ターミナルで直接作業してそれらを使用するようにハイブリッドマネージャーを構成する方法を示しています。運用環境の場合、データを端末に公開しない適切な安全なプロセスを使用することをお勧めします。

シークレットの作成

1. 各サービスユーザーアカウントの安全なパスワードを生成するファンクションを作成します。

```
function generate_pw {
 dd if=/dev/urandom bs=32 count=1 2>/dev/null | base64 | tr +/ -_ | tr     -d =
}
```

1. シークレットに必要なすべてのユーザーとパスワードの環境変数を設定します。

```
HTTP_USERNAME=test_user_001
HTTP_PASSWORD=$(generate_pw)
DB_SUPERUSER_USERNAME=postgres
DB_SUPERUSER_PASSWORD=$(generate_pw)
PORTAL_DB_USERNAME=test_user_002
PORTAL_DB_PASSWORD=$(generate_pw)
COPILOT_DB_USERNAME=test_user_003
COPILOT_DB_PASSWORD=$(generate_pw)
METRICS_USERNAME=test_user_004
METRICS_PASSWORD=$(generate_pw)
```

1. カスタムシークレットを保存する名前空間を作成します。

```
kubectl create namespace edb-migration-copilot
kubectl create namespace edb-migration-portal
```

1. *Migration Portal* **データベースアカウント** を決定するシークレットを作成します。

```
kubectl create secret generic custom-db-secrets \
  --namespace=edb-migration-portal \
  --from-literal=username="${PORTAL_DB_USERNAME}" \
  --from-literal=password="${PORTAL_DB_PASSWORD}" \
  --type=kubernetes.io/basic-auth
```

1. *Migration Portal* **データベースのスーパーユーザーアカウント** を決定するシークレットを作成します。

```
kubectl create secret generic custom-db-superuser-secrets \
  --namespace=edb-migration-portal \
  --from-literal=username="${DB_SUPERUSER_USERNAME}" \
  --from-literal=password="${DB_SUPERUSER_PASSWORD}" \
  --type=kubernetes.io/basic-auth
```

1. コパイロットと移行ポータルの間で *authenticates HTTP communication* というシークレットを作成します。

```
kubectl create secret generic custom-edb-migration-copilot-auth \
  --namespace=edb-migration-portal \
  --from-literal=username=${HTTP_USERNAME} \
  --from-literal=password=${HTTP_PASSWORD} \
  --type=kubernetes.io/basic-auth
```

1. **移行コパイロットおよび移行ポータルのデータベースアカウント** を提供するシークレットを作成します。

```
kubectl create secret generic custom-ragchew-db-secrets \
  --namespace=edb-migration-copilot \
  --from-literal=username="${COPILOT_DB_USERNAME}" \
```

(次のページに続く)

(前のページからの続き)

```
  --from-literal=password="${COPILOT_DB_PASSWORD}" \
  --type=kubernetes.io/basic-auth
```

1. モニタリングと移行コパイロット を有効にするシークレットを作成します。

```
kubectl create secret generic custom-edb-migration-copilot-metrics-auth \
  --namespace=edb-migration-copilot \
  --from-literal=username=${METRICS_USERNAME} \
  --from-literal=password=${METRICS_PASSWORD} \
  --type=kubernetes.io/basic-auth
```

シークレットをインストールに適用する

Hybrid Manager のインストールに付属するデフォルトのシークレットをオーバーライドするには、values.yaml または HybridControlPlane カスタムリソース使用したインストール方法に応じて、新しいシークレットを参照します。

```
parameters:
  edb-migration-copilot:
    db_secrets: custom-ragchew-db-secrets
    metrics_auth_secrets: custom-edb-migration-copilot-metrics-auth
  edb-migration-portal:
    copilot_secrets: custom-edb-migration-copilot-auth
    db_owner: ${PORTAL_DB_USERNAME}
    db_secrets: custom-db-secrets
    db_superuser_secrets: custom-db-superuser-secrets
```

インストールプロセスを続行します。 helm upgrade コマンドを呼び出して values.yaml 構成ファイルを適用するか、HybridControlPlane を適用することによりブートストラッププロセスを実行すると、インストールでシークレットオーバーライドの使用が処理されます。

インストール後にシークレットをオーバーライドする場合は、必ず再度ブートストラップして変更を有効にしてください。

シークレットをバックアップする

次のようなシークレットを取得できます。

```
kubectl get secret -n <namespace> <secret_name> -o yaml
```

このコマンドは、シークレットの内容を YAML 形式で出力するため、安全にコピーして保存できます。作成した各シークレットでこれを繰り返します。

Enabling Key Management Systems for TDE

新しいクラスターを作成するときに、データベースに保存されている encrypt the data に対して**透過的データ暗号化 TDE** を有効にできます。

secure the data encryption key には、選択したパスフレーズを入力するか、 HashiCorp Vault や AWS Key Management Service などのキー管理プロバイダーから独自のキーを持ってくることができます。

!!!note

TDE は、バージョン 15 以降の EDB Postgres Advanced Server および EDB Postgres Extended Server で利用できます。

最初に、選択したキー管理システムプロバイダーが `values.yaml` で有効になっていることを確認します。

- *Adding KMS support*

次に、キー管理システムプロバイダーを構成します。

- *HashiCorp Vault*
- *AWS KMS*
- *Google Cloud KMS* **キーの作成**

Adding KMS support

ハイブリッドマネージャーを展開すると、ブートストラップキット構成 `values.yaml` には、デフォルトで TDE 実装にパスフレーズまたは HashCorp Vault キーを使用する機能が含まれます。

```
# - transparentDataEncryptionMethods: List of TDE Encryption Methods for the agent.
#     Supported options: passphrase, aws_kms, gcp_kms, hashicorp_vault.
#     Default values: passphrase, hashicorp_vault.
transparentDataEncryptionMethods:
  - passphrase
  - hashicorp_vault
```

TDE に別のキー管理システムを使用する場合は、インストール中にこのセクションを編集できます。または、ハイブリッドマネージャーをインストールした後にキープロバイダーを追加できます。その場合、変更した `values.yaml` で helm チャートを再適用して、システムに新しい構成を設定する必要があります。Helm チャートを適用する方法に関するインフラストラクチャ固有の手順については、インストールガイドを参照してください。

HashiCorp Vault

ハイブリッドマネージャー構成には、すぐに使える HashCorp のサポートが含まれています。

AWS KMS

Amazon の KMS、Amazon Key Management サービスを使用するには、 `aws_kms` をリストに追加します。

```
# - transparentDataEncryptionMethods: List of TDE Encryption Methods for the agent.
#     Supported options: passphrase, aws_kms, gcp_kms, hashicorp_vault.
#     Default values: passphrase, hashicorp_vault.
transparentDataEncryptionMethods:
  - passphrase
  - hashicorp_vault
  - aws_kms
```

Google KMS

Google の KMS クラウドキー管理サービスを使用する場合は、 `gcp_kms` をリストに追加します。

```
# - transparentDataEncryptionMethods: List of TDE Encryption Methods for the agent.
#     Supported options: passphrase, aws_kms, gcp_kms, hashicorp_vault.
#     Default values: passphrase, hashicorp_vault.
transparentDataEncryptionMethods:
  - passphrase
  - hashicorp_vault
  - gcp_kms
```

次のステップ

- *Configure HashiCorp Vault as the KMS for TDE*
- *Configure AWS Key Management Service as the KMS for TDE*
- *Configure Google Cloud Key Management Service as the KMS for TDE*

HashiCorp Vault

HashiCorp Vault キーを使用して TDE 対応クラスターを作成する前に、管理者は Hybrid Manager インスタンスの Vault をインストールして構成する必要があります。

この例では、ハイブリッドマネージャーと同じ Kubernetes 環境に Vault のカスタマイズバージョンをインストールし、Vault プロバイダーを構成して、キーを使用して TDE が暗号化を有効にした新しいデータベースクラスターをプロビジョニングできるようにします。

ベンダーニュートラルなキー管理ソリューションを目指している場合は、さまざまなクラウドプロバイダーとオンプレミス環境にわたって一貫した一元的なキー管理を提供し、ベンダーロックインを防止するため、HashCorp Vault を使用するのが現実的です。

Vault のインストール

ハイブリッドマネージャーを展開したのと同じ Kubernetes 環境上の Install HashiCorp Vault 必ず独自の `vault` 名前空間にインストールしてください。

1. Helm リポジトリを設定します。

```
helm repo add hashicorp https://helm.releases.hashicorp.com
```

2. インストールのカスタマイズオーバーライドを作成して、Vault がハイブリッドマネージャーのコントロールプレーンポート 8200 にインストールされるようにします。

```
cat > helm-vault-raft-values.yml <<EOF
injector:
 tolerations: |
   - key: "edbaiplatform.io/control-plane"
     effect: NoSchedule
     operator: Equal
     value: "true"

 affinity: |
   nodeAffinity:
     preferredDuringSchedulingIgnoredDuringExecution:
     - preference:
         matchExpressions:
         - key: edbaiplatform.io/control-plane
           operator: Exists
       weight: 1
server:
 tolerations: |
   - key: "edbaiplatform.io/control-plane"
     effect: NoSchedule
     operator: Equal
     value: "true"

 affinity: |
   nodeAffinity:
     preferredDuringSchedulingIgnoredDuringExecution:
     - preference:
         matchExpressions:
         - key: edbaiplatform.io/control-plane
           operator: Exists
       weight: 1
 ha:
     enabled: true
     raft:
        enabled: true
        setNodeId: true
        config: |
           cluster_name = "vault-integrated-storage"
           storage "raft" {
              path    = "/vault/data/"
           }
```

(次のページに続く)

(前のページからの続き)

```
          listener "tcp" {
             address = "[::]:8200"
             cluster_address = "[::]:8201"
             tls_disable = "true"
          }
          service_registration "kubernetes" {}
EOF
```

3. オーバーライドを使用して Vault をインストールします。

```
helm install -n vault vault hashicorp/vault --values helm-vault-raft-values.yml  --
↪create-namespace
```

4. 続行する前に、すべてのポッドが「正常」ステータスに到達するまで待ちます。

```
kubectl -n vault get pods
```

次に、base64 エンコードされた公開 PGP キーまたは Keybase ユーザー名を使用して Vault インスタンスを初期化し、マスターキーで封印を解除する必要があります。Vault のドキュメント Initialize and unseal Vault を参照してください。

Vault の構成とキーの作成

Vault サービスとハイブリッドマネージャー間の通信を有効にします。

1. Vault がインストールされているポッドにアクセスします。

```
kubectl -n vault exec --stdin=true --tty=true vault-0 -- /bin/sh
```

2. ボルトインスタンスの Kubernetes 通信を有効にします。

```
vault auth enable kubernetes
```

3. Kubernetes クラスターで認証するように Vault を更新し、ハイブリッドマネージャーが Vault を使用できるようにします。

```
vault write auth/kubernetes/config \
  token_reviewer_jwt="$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)" \
  kubernetes_host=https://${KUBERNETES_PORT_443_TCP_ADDR}:443 \
  kubernetes_ca_cert=@/var/run/secrets/kubernetes.io/serviceaccount/ca.crt
```

4. *Transit* シークレットエンジンを有効にして、キー管理機能をアクティブにします。

```
vault secrets enable transit
```

5. キーの Vault ポリシー定義を含む新しいファイルを作成します。

この例では、ポリシーを含むファイルの名前は `tde.txt` です。

```
cat > /tmp/tde.txt << EOL
path "transit/*" {
  capabilities = ["create", "read", "update", "patch", "delete", "list"]
}
EOL
```

6. 作成したファイルを使用して、HashCorp Vault のセキュリティポリシーを更新します。

```
vault policy write tde /tmp/tde.txt
```

7. Transit エンジン周辺キーローテーション、暗号化方法などと adjust according to your requirements のデフォルトパラメーター値を確認します。または、次の手順でキーの作成中に vault write コマンドを使用して、特定のキーのデフォルト値をオーバーライドできます。

8. トランジットエンジンを使用する新しいキーを作成します。

<my-tde-key> を選択したキー名に置き換えます。

```
vault write -f transit/keys/<my-tde-key>
```

9. Vault ポリシーを Kubernetes サービスアカウントにマッピングする tde というロールを作成します。

```
vault write auth/kubernetes/role/tde \
  bound_service_account_names="p-*" \
  bound_service_account_namespaces="p-*" \
  policies=tde \
  ttl=72h
```

!!!note

別の ttl 値を構成して、アプリケーションが再認証が必要になる頻度を指定できます。詳細については、token metadata に関する Vault のドキュメントを参照してください。

10. ポッドを終了します。

```
exit
```

Vault を構成し、キーを作成したので、PG AI コンソールのプロジェクトに追加します。

Vault キーのプロジェクトへの割り当て

コンソールを使用して、作成したキーをハイブリッドマネージャーのプロジェクトに追加します。

1. コンソールで、TDE 対応クラスターを作成するプロジェクトに移動します。プロジェクト内の複数のクラスターにキーを使用できますが、追加のハイブリッドマネージャープロジェクトごとに新しいキーをセットアップすることをお勧めします。

2. 左側のナビゲーションで、 **Settings** ☒ **Security** を選択し、 **+ Add a key** を選択します。

3. キーを構成する場所を選択します。

4. **選択したキー管理システムプロバイダー** の下で **HashCorp Vault** を選択します。
5. **Vault address** に、HashCorp Vault サーバーエンドポイントを入力します。

Vault が `vault` 名前空間、およびハイブリッドマネージャーと同じ Kubernetes 環境にインストールされている場合、Vault のデフォルトの Kubernetes エンドポイントは `http://vault.vault.svc.cluster.local:8200/` になります。設定が異なる場合は、次のように環境からエンドポイントを推定します`<pod_name>.<namespace>.svc.cluster.<host>:<port_configured_for_vault>` 。

6. **Vault Role** に、前に作成したロールの名前、`tde` を入力します。
7. **Vault Key** に、前に作成したキーの名前 `<my-tde-key>` を入力します。
8. 必要に応じて、 **(オプション) キーのフレンドリな名前を入力します**に、覚えやすい代替キーの名前を入力できます。
9. **Add Key** を選択します。

クラスターを作成するときに暗号化を有効にする場合、このキーを使用できるようになりました。追加されたキーは、*enable TDE during cluster creation* を実行するときにオプションとして表示されます。

AWS Key Management Service (KMS)

AWS Key Management Service キーを使用して TDE 対応クラスターを作成する前に、管理者は Hybrid Manager インスタンスの AWS KMS をインストールして構成する必要があります。

この例では、AWS で対称カスタマー管理キーを作成し、ハイブリッドマネージャーで KMS プロバイダーを構成して、そのキーを使用して TDE が有効にして、暗号化が有効になった新しいデータベースクラスターをプロビジョニングします。

Amazon Elastic Kubernetes ServiceEKS でハイブリッドマネージャーを実行している場合、AWS エコシステムのベストプラクティスとのネイティブ統合と調整を提供するため、キー管理に AWS KMS を使用することが現実的です。

前提条件

ハイブリッドマネージャーの `values.yaml` で AWS KMS のパラメーターを設定していることを確認します。有効にするには、*Adding KMS support* を参照してください。

AWS KMS キーの作成

ハイブリッドマネージャーがインストールされているのと同じ AWS アカウントとリージョンで AWS コンソール、AWS CLI または AWS API を使用して Create a symmetric customer-managed key 。

新しいキーには次の構成を使用します。

- **キータイプ** 対称
- **キーの使用法** 暗号化と復号化
- **エイリアス**キーの使いやすいエイリアスを割り当てます機密情報は入力しないでください
- **キーポリシー** Kubernetes クラスターでハイブリッドマネージャー用に構成されたメイン IAM ユーザーまたはロールに、キーを暗号化および復号化する権限があることを確認します。

```
{
    "Sid": "Allow HM identity to use this KMS key",
    "Effect": "Allow",
    "Principal": {
        "AWS": "<IAM-user-or-role-arn>"
    },
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt"
    ],
    "Resource": "*"
}
```

顧客管理 KMS キーを作成したので、コンソールのプロジェクトに追加します。

AWS KMS キーのプロジェクトへの割り当て

作成したキーをプロジェクトに追加します。

1. コンソールで、TDE 対応クラスターを作成するプロジェクトに移動します。プロジェクト内の複数のクラスターにキーを使用できますが、追加のハイブリッドマネージャープロジェクトごとに新しいキーをセットアップすることをお勧めします。
2. 左側のナビゲーションで、 **Settings** ☒ **Security** を選択し、 **+ Add a key** を選択します。
3. キーを構成する場所を選択します。
4. **キー管理システムプロバイダーの選択** で **AWS** を選択します。
5. **Enter AWS Key Management System ARN** に、AWS KMS キー ARN を入力します。 AWS KMS ドキュメントの Find the key ID and key ARN を参照してください。
6. 必要に応じて、 **(オプション) キーのフレンドリな名前を入力します**に、覚えやすい代替キーの名前を入力できます。
7. **Add Key** を選択します。

クラスターを作成するときに暗号化を有効にする場合、このキーを使用できるようになりました。追加されたキーは、*enable TDE during cluster creation* を実行するときにオプションとして表示されます。

Google Cloud Key Management Service (Cloud KMS)

Google Cloud Key Management サービスキーを使用して TDE 対応クラスターを作成する前に、管理者は Hybrid Manager インスタンスの Google Cloud Key Management をインストールして構成する必要があります。

この例では、Cloud KMS で顧客管理キーを作成し、ハイブリッドマネージャーでキープロバイダーを構成して、そのキーを使用して TDE が暗号化を有効にした新しいデータベースクラスターをプロビジョニングできるようにします。

Google Kubernetes EngineGKE でハイブリッドマネージャーを実行している場合、Google エコシステムのベストプラクティスとのネイティブ統合と調整を提供するため、キー管理に Cloud KMS を使用することが

現実的です。

前提条件

ハイブリッドマネージャーの `values.yaml` で Google Cloud KMS のパラメーターを設定していることを確認します。有効にするには、*Adding KMS support* を参照してください。

Google Cloud KMS キーの作成

ハイブリッドマネージャーがインストールされているプロジェクトとリージョンで Google Cloud コンソール、CLI、または API を使用して Create a customer-managed key 。キーは、特定の場所に関連付けられた 1 つのキーリングにのみ存在します。

新しいキーリングには次の構成を使用します。

- **キーリング名** キーリングの特定の名前を選択します
- **ロケーションタイプ** リージョン
- **リージョン**ハイブリッドマネージャーが展開されているのと同じリージョンを選択します

新しいキーリングに関連付けられたキーを作成します。

- **キー名** キーの特定の名前を選択します
- **保護レベル** ソフトウェア
- **目的** 対称暗号化/復号化

キーのデフォルト設定たとえば、キーローテーション、保護レベル、キーマテリアル、バージョンなどと adjust according to your requirements を確認します。

キーの権限を構成します。

- キー権限を編集し、ハイブリッドマネージャーが使用する Google サービスアカウントのアクセスを付与します。これは、ハイブリッドマネージャーのインストール中に構成されたアカウントです。
- 暗号化および復号化の権限が付与されていることを確認しますたとえば、Google ビルトインの Cloud KMS CryptoKey Encrypter および Decrypter ロールを使用します。

Google Cloud KMS キーが作成されたので、Hybrid Manager コンソールのプロジェクトに追加します。

Google Cloud KMS キーのプロジェクトへの割り当て

作成したキーをプロジェクトに追加します。

1. コンソールで、TDE 対応クラスターを作成するプロジェクトに移動します。プロジェクト内の複数のクラスターにキーを使用できますが、追加のハイブリッドマネージャープロジェクトごとに新しいキーをセットアップすることをお勧めします。
2. 左側のナビゲーションで、 **Settings** ☒ **Security** を選択し、 **+ Add a key** を選択します。
3. キーを構成する場所を選択します。
4. **選択したキー管理システムプロバイダー** で **Google Cloud** を選択します。
5. **Region** に、ハイブリッドマネージャーが展開されるリージョンを入力します。us-west-2.

6. **Project ID** に、キーリングを含むプロジェクトの ID を入力します。
7. **Keyring name** に、Google Cloud アカウントで定義されたキーリングの正確な名前を入力します。
8. **Key name** に、Google Cloud アカウントで定義された正確なキー名を入力します。
9. 必要に応じて、 **(オプション) キーのフレンドリな名前を入力します**に、覚えやすい代替キーの名前を入力できます。
10. **Add Key** を選択します。

クラスターを作成するときに暗号化を有効にする場合、このキーを使用できるようになりました。追加されたキーは、*enable TDE during cluster creation* を実行するときにオプションとして表示されます。

Deleting a key

キーが必要でなくなった場合は、削除できます。

以前にキーを追加したプロジェクトに移動します。左側のナビゲーションで、**Settings** ⊠ **Security** を選択し、キーを削除します。

!!!note

既存のクラスターの暗号化に使用されているキーは削除できません。

第6章　Using Hybrid Manager

このセクションでは、日常のクラスター管理から高可用性、ディザスターリカバリーまで、さまざまなユースケースで EDB Postgres AI ハイブリッドマネージャー HM を操作する方法を理解するのに役立ちます。

トピックは対象者ごとに構成されており、一般的なプラットフォームユーザーアプリケーションチーム、データベースエンジニア、プラットフォーム所有者から始まり、クラスターのセキュリティ、オブザーバビリティ、および統合を担当する Kubernetes 管理者まで続きます。

6.1 どこから始めるべきか

基本的なセクションから始めます。

- *Hybrid Manager Terminology* は、主要な HM 概念を紹介します。
- Hybrid Manager overview は、Postgres、Lakehouse、および AI ワークロードに重要なコンテキストを提供します。

そこから、自分の役割と責任に基づいてトピックを探索します。

6.2 ユーザー向け

これらのセクションでは、ハイブリッドマネージャーでの日常的なタスクについて説明します。

- *Migration Portal* **プロジェクトの所有者** —アラートや使用状況レポートを含むプロジェクトレベルの設定を作成、管理、および構成します。
- Cluster Management —バックアップ、クエリ診断、テンプレートを含む Postgres クラスターを作成、監視、および管理します。
- Backup and Restore —バックアップスケジュール、復元ポイント、およびディザスターリカバリー構成を管理します。
- **オプション** *1edbctl* **の使用推奨** —CLI を使用して、展開、構成、およびデータ操作を自動化します。
- **ステップ** *5API* **アクセスを構成する** — HM API を使用してプラットフォーム機能を自動化します。
- *Upgrading Hybrid Manager* —Postgres および PGD のメジャーおよびマイナーアップグレードの手順に従います。
- **モデルイメージの移行** —Oracle、RDS、またはその他の Postgres インストールからスキーマとデータの移行を実行します。
- Image Management —コンテナイメージ、リポジトリ、および AI ナレッジベースアセットを管理します。

- Enable Migration Copilot —Copilot を使用したガイド付き移行ワークフローのロックを解除します。

6.3 Kubernetes 管理者の場合

ハイブリッドマネージャーの下のインフラストラクチャを管理している場合

- *Managing users in Hybrid Manager* —ネイティブユーザーを構成します。
- **オペレーションモニタリング**—エージェントをセットアップし、ビルトインダッシュボードを探索し、Prometheus と Grafana で可観測性を拡張します。
- *Hybrid Manager Multi-DC deployment guide* - 2 つのデータセンター、プライマリとセカンダリをセットアップします。
- *Upgrading Hybrid Manager* - HM プラットフォームを次のマイナー/メジャー/イノベーションリリースバージョンにアップグレードします
- *HA/DR planning and best practices* —実稼働クラスターの高可用性とディザスターリカバリーを構成します。
- *Troubleshooting AI Factory on Hybrid Manager* —すべてのプラットフォーム領域にわたって問題を調査および解決します。

6.3.1 Hybrid Manager Terminology

これらの用語を確認して、ハイブリッドマネージャー HM 機能を理解および活用します。

ハイブリッドマネージャー

Hybrid Manager は、Postgres および Lakehouse データベースとソブリン AI システムの展開、管理、監視、自動化のための EDB のソリューションに与えられた名前です。

これは、最新のプライベートクラウド、パブリッククラウド、マルチクラウド、またはオンプレミスのインフラストラクチャを最大限に活用することにより、展開されたデータベースクラスターのライフサイクルを簡素化する Kubernetes ベースのソフトウェアスタックであり、開発者がプライベートを介してアプリケーションのデータベースを自己管理できますまたは Sovereign Database-as-a-ServiceDBaaS モデル。

プライベート Database-as-a-Service DBaaS

ハイブリッドマネージャーの重要な機能は、Kubernetes でデータベースをプロビジョニングおよび管理できる機能です。この機能は、Private Database-as-a-Service DBaaS と呼ばれます。

この機能により、ハイブリッドマネージャーは、インフラストラクチャ内の自己管理の内部データベースサービスになり、プライベート環境の制御とセキュリティを備えたパブリッククラウド DBaaS の機敏性を提供します。

このアプローチにより、開発者はセルフサービスのプロビジョニング、可観測性、およびトラブルシューティングを実行できます。

エンジニアードシステム

Engineered System は、ボックス内のハイブリッドマネージャーであり、ラックにスロットしてコンテナ化されたアプライアンスとして展開できるすぐに使える物理サーバーです。この展開方法は、ハイブリッドまたはマルチクラウドの世界でデータベース操作を合理化しようとしている組織に、柔軟でポータブルなソリューションを提供します。

PG AI Console

PG AI コンソールは、ハイブリッドマネージャーの集中的な対話ポイントをユーザーに提供する Web ベースのグラフィカルインターフェイスです。このダッシュボードから、データベースの展開と管理、外部データベースの接続と監視、データの移行の実行、ユーザー権限の管理、AI ワークロードの展開などを行うことができます。

PG AI Console の機能の概要については、 *About the PG AI Console* を参照してください。

プロジェクト

ハイブリッドマネージャーでは、 **プロジェクト**は、リソースの管理と作成のための分離を提供する基本的な組織単位として機能します。これらの個々のプロジェクト内でクラスターを作成および管理できます。プロジェクトを使用して、チームごとにエンジニアリング、オペレーション、データサイエンス、環境ごとに開発、ステージング、プロダクション、製品ごと、顧客ごとなどにリソースを管理します。

詳細については、 *Migration Portal* **プロジェクトの所有者** を参照してください。

エステート

ハイブリッドマネージャーでは、 **エステート**は、アクセスするすべてのデータベースリソースの基本的な概要として機能します。ハイブリッドマネージャーでデータベースまたは分析クラスターを作成した場合でも、モニタリングのために外部データベースに接続した場合でも、それはエステートの一部になります。

詳細については、 **エステートインジェスター** *estate ingester* を参照してください。

Launchpad

Launchpad は、ハイブリッドマネージャーの機能を強化するために使用できる多数のサードパーティサービスへのアクセスを提供します。

詳細については、 *Launchpad* を参照してください。

クラスター明確化

コンピューティングにおける **クラスター**とは、共通の目標を達成するために統合システムとして連携する相互接続されたコンポーネントのグループを指します。多くのテクノロジーがこのアプローチを採用するため、**クラスター**という用語は、複数の個々のコンポーネントが統合された方法で管理されるシステムを指すように広く普及しています。

Kubernetes クラスター

ハイブリッドマネージャーのスコープ内で、Kubernetes クラスターは、ハイブリッドマネージャーソフトウェア自分自身が自己完結型インスタンスとして展開される基盤インフラストラクチャを表します。Hybrid Manager は、この基になる Kubernetes クラスターの機能を活用して、Kubernetes で Postgres をオーケストレーションするためのツールとして CloudNativePG を使用して、Postgres データベースの展開を管理します。

**Postgres* cluster* または **Database* cluster* と混同しないでください。

Kubernetes ストレッチクラスター

これは、ハイブリッドマネージャーの展開にも使用できる Kubernetes クラスターのサブタイプです。

!!!note

この機能は、後のバージョンで追加される予定です。

通常、Kubernetes クラスターのすべてのノードは、単一のアベイラビリティーゾーンにあります。高可用性 HA、高度な高可用性 AHA、または分散高可用性 DHA が有効になっているハイブリッドマネージャー展開の場合、プラットフォームは、スタンバイノードによるシームレスなテイクオーバーを開始することにより、プライマリノードの障害を自動的に処理します。

ただし、このビルトインフェイルオーバーメカニズムは、アベイラビリティーゾーン全体が停止した場合には不十分です。ここで、**Kubernetes ストレッチクラスター**が重要になります。これらは、Kubernetes クラスターを複数のアベイラビリティーゾーンまたは地理的に離れた場所に拡張または「拡張」し、1 つのゾーンに障害が発生しても、他の場所の正常なノードがアプリケーションのワークロードを継続して提供できます。この設定では、AZ に依存しない共有ストレージを使用して、AZ 障害が発生している間のデータアクセスを維持する必要があります。

Postgres クラスター

Postgres では、クラスターは単一の *instance of Postgres* であり、Postgres サーバーとしても知られています。Postgres インスタンスのこの単一のオカレンスには、1 つ以上の異なるデータベースを含めることができます。たとえば、データベース *HR* 、 *FINANCE* 、および *LEGAL* を使用する MYCOMPANY のサーバー。

このコンテキストでの「クラスター」の使用があいまいになる可能性があるため、この用語はハイブリッドマネージャーのドキュメントでは使用されません。ここでは、Postgres インスタンスと呼びます。

**Kubernetes* cluster* または **Database* cluster* と混同しないでください。

データベースクラスター

ハイブリッドマネージャーのドキュメントでは、 **データベースクラスター**は、レプリケーション機能を使用してデータベースの復元力をサポートし、パフォーマンスを向上させる Postgres の複数のインスタンスを指します。このようなクラスターベースのシステムは、現代のエンタープライズ Postgres 操作の重要な機能であるため、ハイブリッドマネージャーはデータベースクラスターを中心に構築されています。このため、シングルノードの Postgres インスタンスでもクラスターとして表されます。

たとえば、MYCOMPANY の postgres サーバーにデータベース *HR* 、 *FINANCE* 、および *LEGAL* が含まれ、同じサーバーが冗長性とフェイルオーバーのためにレプリケートされる場合、システム全体をデータベースクラスターと呼びます。2 つのインスタンスを使用するシナリオでは、 *HR* 、 *FINANCE* 、および *LEGAL* を使用したデータベースサーバ A と、 *HR* 、 *FINANCE* 、および *LEGAL* を使用したレプリカサー

バー B が存在します。サーバー A と B は同じデータを持っています。このレプリケーションセットアップの各サーバーは、独自のデータノードに存在します。

データベースクラスターは、ハイブリッドマネージャー内で作成するか、外部で作成できますが、可観測性のためにハイブリッドマネージャーに接続します。ハイブリッドマネージャーを作成または接続できるすべてのクラスタータイプの概要については、*Database cluster types in the Hybrid Manager* を参照してください。

**Kubernetes* cluster* または **Postgres* cluster* と混同しないでください。

ハイブリッドマネージャーのデータベースクラスタータイプ

Hybrid Manager インスタンスは、複数のデータベースクラスターカテゴリをホスト、管理、および監視できる一元化されたガラス画面です。一部のデータベースクラスターはハイブリッドマネージャー内で作成され、完全に維持されますが、他のデータベースクラスターは外部で作成されますが、ハイブリッドマネージャーに接続され、監視と移行機能を提供します。

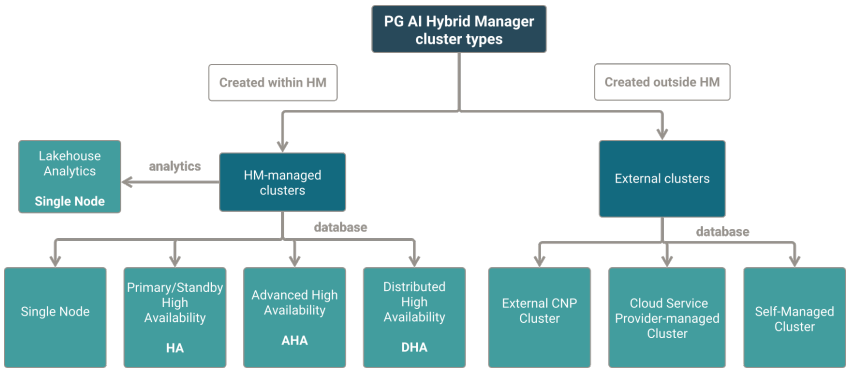


図 1: Cluster Types in Hybrid Manager

現在、次のことができます。

- *HM* **管理対象クラスター** の作成

または、

- **外部クラスターの要件** を接続します。

HM 管理対象クラスター

HM 管理対象クラスターは、ハイブリッドマネージャーで作成したデータベースサーバーです。これらは、プロビジョニング機能を使用して、ハイブリッドマネージャーと同じ Kubernetes クラスターに展開されます。**Estate** ページでは、HM 管理クラスターの **Management** 列の下に **Hybrid Manager** ラベルがあります。

例 展開された PostgreSQL、EDB Postgres Advanced Server、または EDB Postgres Extended Server を使用した単一ノード、プライマリ/スタンバイの高可用性 HA、Advanced High AvailabilityAHA、または分散高可用性 PGD データベースインスタンスハイブリッドマネージャーを使用して。

EDB は、次のデータベーストポロジオプションを提供します。

単一ノードこのオプションは、単一の Postgres インスタンスを作成します。詳細については、*Single-node cluster* を参照してください。

プライマリ/スタンバイの高可用性 HA とも呼ばれるこのトポロジオプションは、Postgres インスタンスをホストし、少なくとも 1 つの追加ノードでそれをレプリケートします。詳細については、*Primary/standby high availability cluster* を参照してください。

高度な高可用性 AHA とも呼ばれるこのトポロジは、単一リージョンのアベイラビリティーゾーンを超えて Postgres インスタンスとそのレプリカの少なくとも 2 つのグループをホストします。詳細については、*Advanced high availability cluster* を参照してください。

分散高可用性 DHA とも呼ばれるこのトポロジは、複数のリージョンのアベイラビリティーゾーンにまたがって Postgres インスタンスとそのレプリカの少なくとも 2 つのグループをホストします。詳細は、*Distributed high availability cluster* を参照してください。

レイクハウス分析-シングルノードハイブリッドマネージャーを使用して *Lakehouse Analytics* クラスターを作成することもできます。PG AI Lakehouse Analytics クラスターは、オブジェクトストレージ内のオープンレイクハウスフォーマット Delta Lake などで保存されているデータを照会および分析するための環境を提供します。Hybrid Manager の統合コントロールプレーンを活用することで、ユーザーは、高速分析クエリ用に最適化されたこれらの特殊なクラスターを簡単に展開および管理できます。

Hybrid Manager は、これらの分析エンジンの展開と操作を簡素化し、データドリブンの意思決定を加速します。

外部クラスター

外部でプロビジョニングされたクラスターを接続できます。

自己管理クラスター

自己管理クラスターは、外部 Postgres または Oracle クラスター Hybrid Manager の外部で作成されます。これらは、ハイブリッドマネージャー以外のプロビジョニング機能などを使用して展開され、多くの場合、ハイブリッドマネージャーとは別のマシンに展開されます。

Estate ページでは、自己管理クラスターの **Management** 列の下に **Self-Managed** ラベルが表示されます。

Agent でのオブザーバビリティのために自己管理クラスターをハイブリッドマネージャーに接続し、*Installing EDB Data Migration Service agent on Linux* で移行を実行できます。

例 物理サーバー、仮想マシンなどに自分自身で展開された PostgreSQL、EDB Postgres Advanced Server、EDB Postgres Extended Server、または Oracle データベースインスタンス。仮想マシンの場合、クラウドインフラストラクチャで実行できますが、データベースクラスターのプロビジョニングはクラウドサービスプロバイダーではなくお客様が管理します。

!!!note

ハイブリッドマネージャーに接続する自己管理 Oracle データベースは、**Estate > Postgres** の下に表示されません。情報を表示するには、**Estate > Migrations** に切り替えます。

ハイブリッドマネージャーに接続する自己管理の Postgres クラスターは、**Estate > Postgres** と **Estate > Migrations** の両方に表示されます。

クラウドサービスプロバイダー管理のクラスター

クラウドサービスプロバイダー管理クラスターまたは CSP 管理クラスターは、Postgres インスタンスの 1 つ以上のオカレンスをホストする外部クラスターハイブリッドマネージャーの外部で作成されます。これらはクラウドサービスプロバイダー CSP プロビジョニング機能を使用して展開され、主要な操作タスクは CSP によって処理されますバックアップ、アップグレード、フェールオーバーなど。ただし、アーキテクチャ、開発者、およびアプリケーション固有のアクティビティは自分自身が処理します。

Estate ページでは、クラウドホスト型データベースクラスターの **Management** 列の下に **Cloud Hosted Databases** ラベルがあります。

connect CSP-managed clusters to Hybrid Manager for observability 、 *Installing EDB Data Migration Service agent on Linux* を使用して移行を実行できます。

例 AWS RDS の Postgres。

外部 CNPG クラスター

外部 CNPG データベースクラスターは、CloudNativePG を使用してハイブリッドマネージャーの外部の Kubernetes 環境で作成された外部データベースクラスターです。これらは、オープンソースの CloudNativePG、EDB CloudNativePG Cluster または EDB CloudNativePG Global Cluster のプロビジョニング機能を使用して展開されます。

エステート ページでは、外部 CNPG クラスターの **管理** 列の下に**外部 CNP クラスター** ラベルがあります。

例 CloudNativePG、EDB CloudNativePG クラスターまたは EDB CloudNativePG グローバルクラスターで展開された PostgreSQL、 EDB Postgres Advanced Server、または EDB Postgres Extended Server データベースインスタンス。

その他の用語

データノード

「データノード」という用語は、ハイブリッドマネージャーによって管理される Postgres データベースインスタンスをホストするように特に指定された Kubernetes ワーカーノードを参照するために使用されます。

Postgres インスタンス

複数のデータベースをホストできる Postgres サーバー。これは、そのインスタンスを実行するために必要なすべてのデータとファイルを含む特定のデータディレクトリで起動します。高可用性展開では、少なくとも 1 つの追加インスタンスが別のアベイラビリティーゾーンに複製されます。

6.3.2 Hybrid Manager console

Hybrid Manager コンソールとは何ですか?

Hybrid ManagerHM コンソールは、Postgres クラスターをプロビジョニングおよび管理するためのユーザーインターフェイスです。これは、データベースのライフサイクルを表示、監視、および管理するための集中的な場所です。

このコンソールでは、リソースを **プロジェクト**に整理します。プロジェクトは、組織に関連するリソースを関連付けるため、管理とアクセスが簡素化されます。

HM コンソールでは何ができますか?

コンソールを使用して、次のことができます。

- Postgres クラスターの作成、表示、および管理
- プロジェクトの作成、表示、および管理
- Migration Portal プロジェクトを作成し、移行を実行する
- 監視データベースの移行
- Lakehouse クラスターの作成、表示、および管理
- カタログの表示と管理
- モデルクラスターのステータスの表示
- アセットライブラリを管理する
- HM 管理対象および外部クラスターの監視

HM コンソールの構成

コンソールは、HM のすべてのページに表示されるページ名を選択してアクセスする 3 つのメインページを中心に構成されています。

-

Projects
、プロジェクトの作成と管理、クラスターの作成と管理、およびクラスターと各プロジェクトを管理するタスクを実行します。

- *Estate* 、リソースをモニタします。クラスターを作成するための開始点として使用することもできます。

-

Asset Library
、コンテナイメージとナレッジベースを表示し、リポジトリを管理できます。

サードパーティサービスのローンチ

HM *Launchpad* を使用して、HM を終了せずに関連するサードパーティサービスを起動できます。

Exploring the Hybrid Manager console

コンソールへのアクセス

アカウントを取得したら、Hybrid ManagerHM コンソールにアクセスし、管理者が提供する `https://<your-domain>/` フォームの URL を使用します。管理者から提供された資格情報を使用してポータルにログインします。

デフォルトでは、コンソールが開き、アカウントの概要ページと、 EDB Postgres AI HM サービスの **単一ウィンドウ**の概要が表示されます。ポータルのどこからでもこのビューに戻るには、EDB ロゴを選択します。

このビューでは、 *Migration Portal* プロジェクトの所有者 および エステート内のリソースのモニタリング の概要を表示できます。

Projects 、 **Estate** 、および **Asset ライブラリー**タブを選択して、さまざまなリソースを監視および作成できるページに移動します。

プロジェクトでリソースを整理する

リソースを作成および管理するには、リソースを保存するプロジェクトが必要です。プロジェクトは、組織に関連するリソースを関連付けるため、管理とアクセスが簡素化されます。

詳細については、 *Migration Portal* プロジェクトの所有者 を参照してください。

エステート内のリソースのモニタリング

HM コンソールでは、 *estate* は、アクセスできるすべてのリソースを指します。物理的資産が不動産の一部と考えられるのと同様に、データベースのライフサイクルに関連するリソースは HM 資産の一部です。HM での不動産の管理とは、リソースの状態を監視するか、安全に保つか、必要に応じて新しいものを作成するかどうかを管理することを意味します。

リソースをエステートとして表示することは、すべてのリソースの単一の統合ビューを提供するため、プロジェクトで表示することとは異なります。

リソースはさまざまなカテゴリにグループ化されています。

- ***Postgres clusters***
-

Migrations

- ***Lakehouses***
-

Catalogs

- **モデルクラスター**

エステートの機能を探る

エステートの **Postgres** タブでは、次のことができます。

- HM 管理か外部かにかかわらず、Postgres クラスターを監視します。
- HM で作成した Postgres クラスターにアクセスします。テーブル内のデータベースクラスターを選択して、詳細ビューを開きます。
- 移行の宛先クラスターを作成します。

エステートの **移行** タブでは、次のことができます。

- HM に接続した

assessment of the databases を表示して、移行の準備をします。

- 特定のデータベースを選択し、

an assessment of its schemas を表示します。

- 特定のデータベースと

perform a schema migration with Migration Portal integration を選択します。

- 移行の宛先クラスターを作成し、データベースの移行を実行します。

不動産の **レイクハウス** タブでは、次のことができます。

- 分析ワークロードに特化した HM クラスターである

analytics clusters
をモニタします。

不動産の **カタログ** タブでは、次のことができます。

- HM に接続されたまたは HM で作成された Apache Iceberg ® テーブルをすべて表示します。

エステートの **モデルクラスター** タブでは、次のことができます。

- AI モデルを提供するクラスターを表示およびアクセスします。

クイックアクションメニュー

エステートページのどこからでも、 **クイックアクション**メニューの一般的なアクションにアクセスできます。アクションには次のものが含まれます。

- **EDB Postgres AI クラスター** を作成します** 既存のプロジェクトで Create an EDB Postgres AI cluster 。
- **新しいレイクハウスクラスターの追加** —*Add a new Lakehouse cluster* を既存のプロジェクトに追加します。
- **自己管理クラスターの追加** —新しい自己管理クラスターを既存のプロジェクトに追加します。

Launchpad

コンソールでは、Launchpad を使用して、HM の機能を強化するために使用できるサードパーティサービスにアクセスできます。たとえば、Launchpad から *Customizing Migration Portal secrets for secure internal communication* にアクセスできます。これらのサービスは、展開されているプラットフォームに関係なく、HM のすべてのユーザーが利用できます。

Launchpad にアクセスするには、ユーザーメニューから **Launchpad** を選択します。

利用可能なすべてのサービスが Launchpad に表示され、各サービスの簡単な説明が表示されます。検索フィルタは、探しているサービスを見つけるのに役立ちます。サービスには次のものが含まれます。

- Grafana
- プロメテウス
- サノス - クエリ
- Thanos - ストアゲートウェイ
- Thanos - Compactor

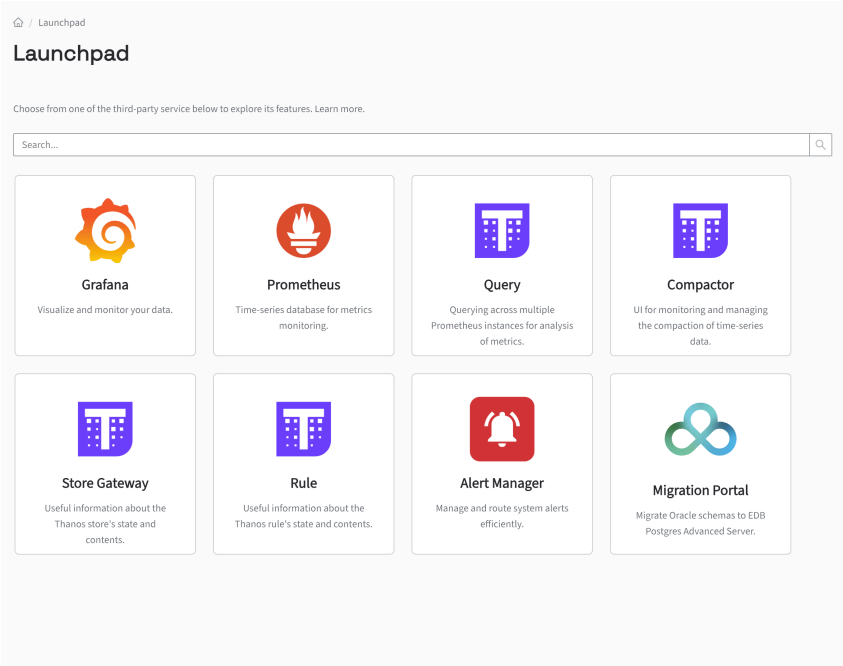


図 2: Launchpad

- サノス - ルール
- アラートマネージャー
- EDB Migration Portal

ユーザーメニュー

ユーザーメニューから、一部のポータル関連のアクションと一部のアカウント関連のオプションに集中的にアクセスできます。

ポータル関連のオプション

- **Launchpad** —*Launchpad* を開き、HM にインストールされているサードパーティサービスにアクセスします。
- **設定** — 組織名 **プロフィール** を介して *set your monitoring notification preferences* を設定できます。
- **Projects** — Projects overview を開きます。
- **Tags** — **Tags** ページを開き、タグ名を追加し、ディスプレイ内のタグ/バッジの色を選択できます。
- **User Management** —ユーザーを追加し、ロールを割り当てできる *User Management* ページを開きます。
- **使用レポート** —ディスク IO、vCPU、メモリ、ディスクおよびアップタイムを含む *information about each cluster's use of resources* を表示します。
- **アクティビティログ** —HM で管理されているリソースの作成、削除、更新など、実行されたアクションのリストを表示します。

アカウントオプション

- **マイアカウント** —プロフィールの表示、アクセスキーの表示と管理、およびアカウント、組織、およびプロジェクトの通知の管理ができます。
- **ログアウト**

Postgres clusters

アカウント内のすべてのプロジェクトのすべてのクラスターに関する情報を表示するには、**Estate** ページから **Postgres** タブを選択します。

概要統計の表示

Postgres タブの上部に、Postgres クラスターの次の側面を要約したカードを表示できます。

- **アクティブなアラート** —エステートで監視されているすべての Postgres クラスターにおける、高、中、および低の重大度のアラートの集計数。
- **上位 CPU 使用率** —エステートで監視されているすべての Postgres クラスターの中で、最も多くの CPU 使用率を示した上位 9 位のクラスター。カードの下部にあるコントロールを使用して、クラスターをページ移動できます。
- **リソース使用量** —エステートで監視されているすべての Postgres クラスターの集計された CPU、メモリ、およびストレージの使用量。

クラスターテーブル

クラスターテーブルは、プロジェクト全体で使用可能な Postgres クラスターのリストであり、パフォーマンス、状態、ステータス、およびその他のメトリックを監視するのに役立ちます。ビューを変更して表示する列の数を変更したり、フィルターを適用して目的のクラスターのみを表示したりできます。

表示する列の選択

デフォルトでは、テーブルには次の列が表示されます。

- Project Name — クラスターが属するプロジェクト。
- Cluster — クラスター名。
- Management — クラスターが管理される方法アプライアンスまたは自己管理。
- Cluster Type — クラスターのタイプ PGD、シングルノード、または HA。
- Status — クラスターの現在のステータス OK、Error、 注意、または Working 。
- Engine —このクラスターが実行している現在の Postgres ディストリビューションタイプ Postgres、EDB Postgres Advanced Server、または EDB Extended Server。
- Version —クラスターが実行している Postgres ディストリビューションのメジャーバージョン 13、14、15、16、または 17。
- Alerts — クラスターの高、中、および低の重大度のアラートの数。

- TPS —クラスターによって行われた 1 秒ごとのトランザクションと、最近の傾向を要約した折れ線グラフ。
- CPU —クラスターによって現在使用されている CPU の割合と、最近の傾向を要約した折れ線グラフ。
- Memory —クラスターによって現在使用されているメモリの割合と、最近の傾向を要約した折れ線グラフ。
- Disk —クラスターが現在使用しているディスク領域の割合と、最近の傾向を要約した折れ線グラフ。
- Tags — クラスターに関連付けられたタグ。

表示する列を選択することもできます。歯車を選択して、上部に検索ボックスを備えたメニューを開きます。検索ボックスを使用して、テキスト一致を使用して表示する列をフィルタリングします。返された列名のリストから、テーブルに表示する列名を選択します。

これらは表示できる他の列です。

- アラートの数 — クラスターのアラートの合計数。
- コア数 — クラスターが使用するコアの合計数。
- Conn.Usage — 接続の使用率。
- IO Wait % — 入力/出力タイプの待機イベントの割合。
- IOPS — クラスターの 1 秒ごとの入出力操作の数。
- カーネルバージョン — 自己管理クラスターのプラットフォームタイプ。
- Last Backup — クラスターが最後にバックアップされた時の ISO 8601 タイムスタンプ。
- OS バージョン —自己管理クラスターのオペレーティングシステムバージョン。
- Provider — CSP プロバイダーの ID。
- Region — アプライアンスクラスターのロケーション ID。
- Rep. Lag — ミリ秒単位のレプリケーションラグ。
- TDS — ギビバイト単位のデータベースの合計サイズ。

> **Tip**
>
> 列をデフォルトのセットにリセットするには、歯車メニューから**Reset Columns**を選択します。

表示するクラスターのフィルタリング

次の 2 つの手法のいずれかを使用して、表示されるクラスターのリストを絞り込んで、関心のあるクラスターを含めることができます。

- テーブルの **Search** ボックスを使用して、表示するクラスターを名前で検索します。
- **Filter** コントロールを使用して、次のプロパティに一致するクラスターを表示します。各プロパティには、プロパティに関連する特定のタイプまたは値を選択するためのサブメニューがあります。
- クラスタータイプ

- Engine
- ヘルススコアインデックス
- ヘルススコア統計
- 削除されたクラスターを含む b_tran_14 ロケーション
- Postgres バージョン
- プロジェクト名
- リソースタイプ、以下を含む Hytran Manager —edb_idClusters3).
  - 自己管理 Postgres — 自己管理であるクラスター.
  - 自己管理 Postgres データベース —自己管理であるデータベースを含むクラスター.
- ステータス

データの古さの監視

テーブルのデータが更新される頻度を確認するには、 Metrics latency の **Estate page > Postgres テーブル**で提供されている値を参照してください。

テーブルの下のメッセージは、テーブルが新しいデータで最後に更新された時刻を示します。

Lakehouse clusters

Lakehouse クラスターを表示するには、**Estate** ページから **Lakehouse** を選択します。Lakehouse クラスターを表示するときに、フィルターを適用して結果を絞り込むことができます。アクセスできる Lakehouse クラスターの数、クラスターのメタデータとメトリックを確認できます。

Lakehouse クラスターの追加

Lakehouse クラスターをエステートに追加するには、 **Quick Actions** メニューから **Add New Lakehouse Cluster** にカーソルを合わせ、新しいクラスターに関連付けるプロジェクトを選択します。**Create Analytics Cluster** ページが開きます。このページでは、クラスターを追加する 2 つの方法があります。テンプレートからビルドするか、カスタムビルドを作成します。

- テンプレートからビルドするには、使用可能なテンプレートがない場合、**Create New Template** を選択して作成します。既存のテンプレートを使用するには、 **Use Template** を選択します。
- カスタムビルドを作成するには、 **Start from Scratch** を選択します。

分析クラスターのテンプレート作成ワークフローとカスタムビルドワークフローは、通常のクラスター作成ワークフローと同じです。Creating clusters を参照してください。

Hybrid ManagerHM の Analytics / Lakehouse 機能の使用の詳細については、Analytics/Lakehouse を参照してください。

Lakehouse クラスターテーブル

clusters テーブルは、プロジェクト全体で使用可能な Lakehouse クラスターのすべてのリストです。パフォーマンス、状態、ステータス、およびその他のメトリックを監視するのに役立ちます。ビューを変更して表示する列の数を変更したり、フィルターを適用して目的のクラスターのみを表示したりできます。

表示する列の選択

デフォルトでは、テーブルには次の列が表示されます。

- プロジェクト名 クラスターが属するプロジェクト。
- Cluster クラスター名。
- 管理 クラスターが管理される方法アプライアンスまたは自己管理。
- クラスタータイプ クラスターのタイプ PGD、シングルノード、または HA。
- ステータスクラスターの現在のステータス OK、エラー、注意、または作業中。
- エンジンこのクラスターが実行している現在の Postgres ディストリビューションタイプ Postgres、EDB Postgres Advanced Server、または EDB Extended Server。
- バージョンクラスターが実行している Postgres ディストリビューションのメジャーバージョン 13、14、15、16、または 17。
- アラート クラスターの高、中、および低重大度のアラートの数。
- ヘルススコア
- TPS クラスターによって行われた 1 秒ごとのトランザクションと、最近の傾向を要約した折れ線グラフ。
- CPU クラスターによって現在使用されている CPU の割合と、最近の傾向を要約した折れ線グラフ。
- Memory クラスターによって現在使用されているメモリの割合と、最近の傾向を要約した折れ線グラフ。
- Disk クラスターによって現在使用されているディスク領域の割合と、最近の傾向を要約した折れ線グラフ。
- Tags クラスターに関連付けられたタグ。

表示する列を選択することもできます。歯車を選択して、上部に検索フィールドを備えたメニューを開きます。検索フィールドを使用して列をフィルタリングし、テキスト一致を使用して列名前を表示します。列名のリストから、テーブルに表示するものを選択します。

これらは表示できる他の列です。

- アラートの数 クラスターのアラートの合計数。
- コア数 クラスターが使用するコアの合計数。
- Conn. 使用状況 パーセンテージでの接続使用状況。
- IO Wait % 入力/出力タイプの待機イベントの割合。
- IOPS クラスターの 1 秒ごとの入出力操作の数。

- カーネルバージョン 自己管理クラスターのプラットフォームタイプ。
- OS バージョン 自己管理クラスターのオペレーティングシステムバージョン。
- Provider CSP プロバイダーの ID。
- リージョン アプライアンスクラスターのロケーション ID。
- TDS ギビバイト単位のデータベースの合計サイズ。

> **Tip**
>
> 列をデフォルトのセットにリセットするには、歯車メニューから**Reset Columns**を選択します。

表示するクラスターのフィルタリング

次の 2 つの手法のいずれかを使用して、表示されるクラスターのリストを絞り込んで、関心のあるクラスターを含めることができます。

- テーブルの **Search** ボックスを使用して、表示するクラスターを名前で検索します。
- **Filter** コントロールを使用して、次のプロパティに一致するクラスターを表示します。各プロパティには、そのプロパティに関連する特定のタイプまたは値を選択するサブメニューがあります。
  - Engine
  - 削除されたクラスターを含める s
  - Location
  - Management
  - Postgres バージョン
  - プロジェクト名
  - ステータス

データの古さの監視

テーブルのデータは 30 秒ごとに更新されます。テーブルの下のメッセージは、テーブルが新しいデータで最後に更新された時刻を示します。

6.3.3 For Users

Projects

Hybrid ManagerHM では、プロジェクトを使用して、組織に関連するリソースを関連付けます。プロジェクトにより、これらのリソースの管理とアクセスが簡素化されます。

プロジェクトのリストを表示するには、 **Projects** タブを選択します。各プロジェクトカードには次のものが表示されます。

- プロジェクト名

- プロジェクト内のクラスターの数
- プロジェクトにアクセスできるユーザーの数
- プロジェクトにタグを追加するためのコントロール、リソースを整理するのに役立つキーと値のペア
- プロジェクトで構成されたアクティブなクラウドサービスプロバイダー CSP のログ。

各カードの省略記号メニューから、 プロジェクトの詳細の表示時に実行できる機能にすばやくアクセスできます。

所有者権限がある場合は、このビューからプロジェクトを作成することもできます。

プロジェクトの作成

1. **Projects** タブを選択します。
2. **Create New Project** を選択します。
3. プロジェクトの名前を入力します。
4. オプションで、タグをプロジェクトに追加します。
5. **Create New Project** を選択します。

プロジェクトを作成すると、プロジェクトの詳細が表示されます。あなたは新しいプロジェクトの所有者であり、リソースの追加を開始できます。

プロジェクトの詳細

プロジェクトの詳細ビューでは、プロジェクト管理タスクを実行します。詳細ビューを開くには、プロジェクトを作成するか、**プロジェクト**ページのプロジェクトのリストからプロジェクトを選択します。このビューには、プロジェクト内の各クラスターの説明と、メモリ使用量、CPU 使用量、ストレージ使用量、ディスク IOPS とトランザクションレート、データベースサイズを含むメトリックが含まれています。

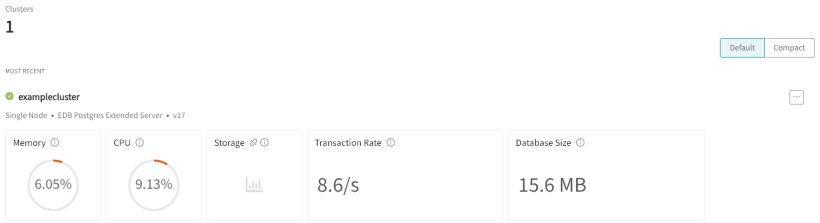


図 3: Project Overview

プロジェクトでは、次のことができます。

- Create, view, and manage clusters 。
- *Manage storage locations* 。
- カタログを追加します。
- *View and manage the users in the project* 。

- アクティビティログを表示およびダウンロードします。アクティビティログは、クラスターの作成と削除、ソフトウェアのインストールなど、現在のプロジェクトで実行されたすべてのアクションの記録です。
- Manage project settings 通知、セキュリティ、アラートを含む
- *View a usage report* 。
- View and manage database migrations 。

クラスターごとに、カードの右上隅にある省略記号メニューを使用して、次のこともできます。

- **Edit Cluster** から クラスターを編集する を選択します。
- **資格情報を選択して、接続 URL、ユーザー名、データベース名を含むクラスターの接続の詳細を表示します。これらの詳細を使用して、Postgres クライアントからクラスターに接続できます。パスワードは表示されませんが、必要に応じてパスワードをリセットできます。詳細については、レジストリの資格情報 を参照してください。

プロジェクトの管理

プロジェクトの名前を変更したり、削除したりできます。プロジェクトの管理 を参照してください。

Managing projects

プロジェクトの名前を変更したり、削除したりできます。

プロジェクトの名前の変更

プロジェクトの一括ビューと **プロジェクト**タブでプロジェクトの名前を変更できます。プロジェクトのいずれかのリストから

1. プロジェクト名の右側にある省略記号メニューから、 **Edit** を選択します。
2. 新しいプロジェクト名を入力し、 **Update Project** を選択します。

プロジェクトの削除

プロジェクトの一括ビューと **プロジェクト**タブでプロジェクトを削除できます。プロジェクトのいずれかのリストから

1. プロジェクト名の右側にある省略記号メニューから、 **Delete** を選択します。

削除の確認を求められます。

1. プロジェクト内にクラスターがある場合、プロジェクトを削除する前にそれらを削除する必要があります。ディスプレイには、残りのクラスターが表示され、各クラスターの横にチェックボックスが表示されます。削除する各クラスターのチェックボックスを選択するか、**Select All** を選択します。
2. 削除するクラスターを選択したら、テキストボックスに `delete clusters` を入力して削除を確認する必要があります。このテキストを入力すると、[**Delete All Clusters**] ボタンが有効になります。
3. **Delete All Clusters** を選択して、選択したクラスターを完全に削除します。

4. プロジェクトにクラスターがない場合、ダイアログボックスが表示され、チェックボックス付きのプロジェクト名をリストします。チェックボックスを選択し、テキストフィールドに `delete project` と入力します。このテキストを入力すると、[**Delete Project**] ボタンが有効になります。

5. **Delete Project** を選択して、プロジェクトを完全に削除します。

Settings

通知、セキュリティ、アラートなどのプロジェクト固有の設定を構成できます。**Settings** メニューを使用して、これらの設定にアクセスします。

- **プロファイル**を選択して、プロジェクトの名前をいつでも変更できます。プロジェクト名は、ハイブリッドマネージャー HM インスタンスで一意である必要があります。
- ***Notifications*** を選択して、プロジェクトの通知の送信方法を構成します。
- **Security** を選択して、透過的データ暗号化 TDE の外部管理キーをプロジェクトに追加します。後のクラスター作成中にこれらのキーを使用して、データの暗号化を追加できます。KMS 固有の手順については、 *Enabling Key Management Systems for TDE* を参照してください。
- ***Alerts*** を選択して、プロジェクトのアラート設定を表示し、アラートを送信する方法と時期を制御します。

Notifications

Hybrid ManagerHM を使用すると、受信する通知の種類をカスタマイズできます。2 種類の通知チャネルがあります。

- **ユーザー通知**電子メール、アプリ内の受信ボックス、またはその両方を介して通知を受信できます。これらの通知は、ロールで許可されている場合に有効にしているすべてのユーザーを対象としています。
- **システム通知**電子メールメールグループ、Webhook、ポケベルデューティ、または任意の組み合わせを介して通知を受信できます。これらの通知は、主に組織内のタスクを自動化するために、イベントごとに 1 回トリガーされます。

HM でのロールにより、通知設定へのアクセスが決まります。次の種類の通知が使用できます。

- アカウントユーザー通知チャンネル用。次の通知を受け取ることができます。
  - 新しいプロジェクトロールがあなたに割り当てられました。
  - プロジェクトロールがあなたから割り当て解除されました。
  - 個人アクセスキーの有効期限が間もなくエキスパイアと。
- グローバルユーザーおよびシステム通知チャネル用組織の所有者にのみ可視。次の通知を受け取ることができます。
  - 新しいプロジェクトが作成されました。
  - マシンアクセスキーの有効期限が間もなくエキスパイア。
- プロジェクトユーザーおよびシステム通知チャネル用。次の通知を受け取ることができます。
  - アクティブなクラスターは一時停止されました。

- 一時停止されたクラスターが再アクティブ化されました。

通知の設定

プロジェクトの所有者/編集者および組織の所有者/管理者は、表示できるイベントの通知を有効または無効にできます。通知は、アプリ内の受信ボックス、電子メール、またはその両方で受信できます。

ユーザーが行った通知設定は、そのユーザーにのみ適用されます。電子メール通知が有効になっている場合、電子メールはユーザーのログインに関連付けられた電子メールアドレスに送信されます。

システム通知チャネルは、組織の所有者がグローバルレベルで、またはプロジェクト所有者がプロジェクトレベルで構成できます。デフォルトでは、各組織のプロジェクトはグローバルレベルのシステム通知チャネルを継承します。ただし、プロジェクトレベルでシステム通知チャンネルを有効/無効または再構成して、デフォルト設定をオーバーライドすることができます。

グローバルレベルで通知を設定する

組織の所有者のみがグローバル通知を設定できます。

Webhook の場合

1. プロフィールメニューから、 **設定** を選択します。
2. **Notifications** タブを選択します。
3. **Webhook** を選択して、次を入力します。

- webhook のホスト URL b_tran_2 Webhook の方法-POST または PUT

1. オプションで、次の認証パラメーターのいずれかを提供します。
    - 基本認証の場合 ユーザー名とパスワード
    - ベアラートークン
2. **Save** を選択します。

PagerDuty の場合

PagerDuty 通知チャンネルを構成する前に

- 昇格した権限を持つ PagerDuty アカウントへのアクセスが必要です。
- PagerDuty サービスは、 Events API v2 統合を使用して作成および構成する必要があります。
- PagerDuty アカウントで、目的のサービスに移動し、その **統合**設定にアクセスします。
- 新しい Events API v2 統合を追加して構成します。
- 新しく作成した統合用に PagerDuty が提供した統合キールーティングキーをコピーします。

PagerDuty 通知チャンネルを構成するには

1. プロフィールメニューから、 **設定** を選択します。
2. **Notifications** タブを選択します。
3. **PagerDuty** を選択します。

4. ルーティングキー統合キーを設定します。
5. **Save** を選択します。電子メールの場合
6. プロフィールメニューから、 **設定** を選択します。
7. **Notifications** タブを選択します。
8. **電子メール** を選択し、次のように入力します。
    - 電子メールの宛先 - オプションで、システムメール通知の場合電子メールグループ
    - 電子メールの送信元 - 電子メールを送信している電子メールの場合
    - ホスト - 電子メールの URL サービス
    - ポート - ポート email service
    - ユーザー名 - 電子メールサービスのユーザー名
    - パスワード - 電子メールサービスの
9. **Save** を選択します。

プロジェクトレベルで通知を設定する

プロジェクトレベルの通知設定は、グローバル設定をオーバーライドできます。各プロジェクトは、一意の構成を持つことができます。

1. **Projects** タブを選択します
2. 任意のプロジェクトを選択します。
3. プロジェクトの概要ページに移動します。
4. 左側のメニューで **Settings** を選択します。
5. **通知** を選択します。
6. 各システム通知チャネルに対して グローバルレベルで通知を設定する の下の手順に従います。

通知の表示

次のロールのユーザーは通知を表示できます。

- 組織の所有者はグローバル通知を表示できます。
- プロジェクトの所有者/編集者は、プロジェクトの通知を表示できます。
- すべてのユーザーが自分のアカウント通知を表示できます。

各通知は、HM で複数のロールが割り当てられているユーザーが属しているレベルとプロジェクトを示します。

アプリ内通知を表示するには、HM コンソールの上部にあるベルを選択します。通知を読んだり、未読としてマークしたり、アーカイブしたりできます。

メール通知を表示するには、設定したメールアドレスの受信箱を確認します。

通知設定の管理

1. プロフィールメニューから、 **My Account** を選択します。
2. **Notifications** タブを選択します。
3. **Account** 、 **Global** 、または **Project** セクションまでスクロールして、管理するイベントを表示します。
4. セクションを展開して、イベントのリストを表示します。
5. 通知ごとに、次のことができます。 - イベント名の前にあるトグルボタンを使用して、イベントの通知の受信をオプトインまたはオプトアウトします。- チェックボックスを使用して、メールとアプリ内通知をカスタマイズします。

Alerts

アラートは、ハイブリッドマネージャー HM 可観測性エコシステムに不可欠です。Postgres クラスターのパフォーマンスに影響を与える重要なイベントについて積極的に通知します。重要なパフォーマンスメトリックを継続的に監視し、事前定義されたしきい値を超過するたびにアラートをトリガーします。この堅牢なアラートシステムは、HM エコシステムにシームレスに統合され、すべてのリージョンの Postgres クラスターとそのインフラストラクチャから収集されたリアルタイムデータを活用します。アラートは、HM で作成されたプロジェクトごとにデフォルトで有効になっています。

アラートシステムはタイムリーな通知を保証し、情報に基づいた迅速なアクションを可能にします。特定のしきい値と間隔を構成して、アラートエクスペリエンスをカスタマイズできます。モニタリングとアラートのこの統合により、潜在的な問題への迅速な対応が可能になり、最適なクラスターパフォーマンスを維持できます。

アラートを表示する

クラスターの詳細ページの **モニタリング**タブにあるアラートにアクセスし、特定のクラスターのアラートの詳細を表示できます。

HM クラスターの場合

1. HM コンソールで、 **Projects** を選択します。
2. リストからプロジェクトを選択します。
3. 左側のナビゲーションで、 **Clusters** を選択します。
4. 準備ができているクラスターを選択します。
5. クラスターの詳細ページで、 **Monitoring** タブを選択します。

外部 HM クラスターの場合

1. HM コンソールで、 **Estate** を選択します。
2. すべてのクラスターをリストしているテーブルに移動します。
3. **管理** が **自己管理**である準備ができているクラスターを選択します。
4. クラスターの詳細ページで、 **Monitoring** タブを選択します。

[**Monitoring**] タブには、アクティブなアラートリストが表に表示されます。このタブの上部に、アクティブなアラートの概要を表示できます。考えられる重大度レベルは、高、中、および低です。

次を使用して特定のアラートを検索できます。

- クエリベースの検索のための検索ボックス
- 特定の日付範囲内を検索する日付範囲セレクター
- 重大度レベルで検索するフィルタ

Node 列でノードを選択して、アラートがトリガーされたときの状態でチャートを表示します。ノードを選択すると、**Monitoring** タブのフィルタが適用され、関連するチャートが表示されます。

次のこともできます。

- **Download CSV** を選択して、アラートの詳細を CSV 形式でダウンロードします。
- 重大度、アラート開始時間、およびアラート名に基づいて、アラートリストを並べ替えます。
- **Refresh** を選択してリストを更新します。
- 列セレクターを使用して、テーブルに表示する列を選択します。

アラートの構成

プロジェクト管理者は、プロジェクトレベルでアラートを構成できます。アラートが構成および適用されると、システムがアラートを完全に処理するまでに数分かかります。

アラートを構成するには

1. HM コンソールの上部で、 **Projects** を選択します。
2. リストからプロジェクトを選択します。
3. 左側のナビゲーションで、 **Settings** を選択します。
4. **アラート** を選択します。

アラートごとに、 **Edit** を選択して設定します。

- Enable/Disable 特定のアラートを有効または無効にします。
- Notify After アラートがトリガーされるまでの時間を設定します。
- しきい値超過した場合にアラートをトリガーするパーセンテージまたは値を指定します。

これらのアラートはデータベースレベルで構成できます。

- 接続の使用状況
- 最近の WAL アーカイブは成功しませんでした
- ノードダウン
- バイト単位のレプリケーションラグ
- 合計接続数
- 非アクティブなレプリケーションスロットの合計

- トランザクションのロールバック
- WAL は成長し続けます
- WAL のディスク領域が不足しています

これらのアラートはホストレベルで構成できます。

- CPU 使用率
- ディスク IOPS
- ディスクの使用量
- メモリ使用量

!!!note-HM は、アラート構成に加えた変更を保存します。 - 技術的な制限のため、外部 HM クラスターではすべてのアラートが利用できるわけではありません。

Managed storage locations

管理対象ストレージの場所は、将来のリリースでプロジェクトのデータを保存するために使用されます。Hybrid ManagerHM コンソールで、プロジェクトと同じストレージを共有するストレージの場所を作成できます。ただし、HM サービスは管理対象ストレージ場所機能を使用していません。

Usage report

使用レポートページは、クラスターの次の履歴使用メトリックのビューを提供します。

Term	Definition	Measure
Cluster Name	The name of the cluster.	•
Cluster Type	The type of the cluster.	•
Database Type	The type of the database.	•
Disk Allocated (GiB-Hour)	The amount of disk storage allocated.	GiB-Hours
Disk IO (GiB)	The amount of data read from and written to disk storage.	GiB
Disk Used (GiB-Hour)	The amount of disk storage used.	GiB-Hours
Disk Used Percent	The total amount of time that the allocated storage (disk) has been available to your database system. This does not measure activity like Disk IO, but the total duration the storage has been provisioned and available	Hours
Memory Allocated (GiB-Hour)	The amount of memory allocated.	GiB-Hours

次のページに続く

表 1 – 前のページからの続き

Term	Definition	Measure
Memory Used (GiB-Hour)	The amount of memory used.	GiB-Hours
Memory Used Percent	The amount of RAM (memory) consumed in hours versus the amount allocated.	•
Project	The name of the project.	•
Uptime (Hour)	The total amount of time the database instance has been continuously running and available	Hour
vCPU Allocated (Core-Hour)	The amount of vCPU allocated.	Core-Hours
vCPU Used Percent	The number of virtual CPU hours consumed versus the number allocated.	•
vCPU Used (Core-Hour)	The amount of vCPU used.	Core-Hours

Working with clusters

Hybrid ManagerHM を使用して、データベースクラスターを作成および管理できます。データベースは、次の Postgres タイプのいずれかに基づくことができます。

- EDB Postgres Advanced Server
- EDB Postgres 拡張サーバー PGE
- PostgreSQL

クラスター管理には、新しいクラスターの作成、アクセスの制御、クラスターパフォーマンスの監視、およびクラスターの編集、復元、または削除などのライフサイクル操作の管理が含まれます。再利用可能なテンプレートを作成して、今後のクラスターの作成を簡素化することもできます。

何ができるでしょうか？

- Create a cluster

事前定義されたテンプレートから新しいデータベースクラスターを作成する方法、または最初から構成する方法を学びます。

- Manage a cluster

構成の変更、バックアップの復元、クラスターの削除、テンプレートの作成などのキー管理タスクを実行します。

- *Manage database access*

ユーザー、ロール、および権限を設定して、クラスターにアクセスおよび接続できる人を制御します。

- View clusters

すべてのデータベースクラスターのステータス、構成、およびアクティビティを監視および確認します。

Creating a database cluster

テンプレートがある場合からデータベースクラスターを作成するか、クラスターをスクラッチから作成できます。

スタートポイント

クラスターを作成するための開始点は、 **Estate** ページまたはプロジェクトページのいずれかです。 **Estate** ページから開始する場合、データベースを作成するプロジェクトを指定します。プロジェクトページから開始すると、そのプロジェクトにデータベースを作成します。

Estate ページから開始するには

1. **Quick Actions > Create New Database Cluster** を選択します。

複数のプロジェクトがある場合、プロジェクトを選択するように求められます。

2. プロンプトが表示されたら、データベースを作成するプロジェクトを選択します。

プロジェクトを選択した後、またはプロジェクトが 1 つだけの場合、 **Create Cluster** *[パスの選択] ページから* ページが開きます。

プロジェクトページから開始するには、これらのアプローチのいずれかを使用できます。プロジェクトページに移動するには、**Projects** ページからプロジェクトを選択します。

- プロジェクトページで、 **Create New > Database Cluster** を選択します。
- プロジェクトページで、**Clusters > Postgres Cluster** を選択します。[**クラスター**] ページで、[**Create New > Database Cluster**] を選択します。

どちらの場合も、クラスターを作成するための **Choose your path** ページが開きます。

道を選択してください

[**Choose Your Path**] ページでは、事前定義されたテンプレートからクラスターを作成するか、カスタムビルドとしてスクラッチから作成するかを指定します。

- 使用可能なテンプレートからクラスターを作成するには、 **Use Template** を選択します。 *Creating a cluster from a template* を参照してください。
- 使用可能なテンプレートがない場合、またはクラスターを最初から作成する場合は、**Custom Build** を選択します。

Create New Template を選択して、ここからテンプレートの作成を開始することもできます。*Create a database cluster template* を参照してください。

クラスターを最初から作成する

Custom Build を選択したら、新しいクラスターの指定を開始できます。

Cluster Info タブで、クラスタータイプを選択します。

- シングルノード
- 高可用性

- 高度な高可用性
- 分散高可用性

作業するタブを選択するか、**Next** を使用してタブを順番に実行します。**DB 構成** および **追加設定** タブの代わりに **Advanced High Availability** または **Distributed High Availability** を選択した場合、***Data Groups*** タブが表示されます。

- ***Cluster Info***
- ***Cluster Settings***
- ***DB Configuration*** オプショナル
- ***Additional Settings*** オプショナル
- ***Data Groups*** 分散高可用性クラスター用

> **注釈**
>
> 構成しないオプションタブの設定はデフォルト値を使用します。

クラスターを作成する

タブの構成が終了したら、[**Create Cluster**] を選択します。設定が正しくないまたは欠落しているタブには、赤いインジケーターが表示されます。エラーを修正し、**Create Cluster** を再度選択します。

有効な設定で **Create Cluster** を選択すると、クラスターはプロビジョニングを開始し、クラスターのリストに戻ります。ポップアップはクラスターの作成を確認し、新しいクラスターがクラスターのリストに追加されます。クラスターはバックグラウンドで作成されます。クラスターのリストで進行状況を確認できます。

クラスターの作成の進行中にクラスターの詳細を表示しようとすると、概要、バックアップ、ログ、プロパティのみを表示できます。

Cluster types to create

create your cluster を開始すると、[**Cluster Info**] タブが、作成するクラスターのタイプを選択するように求めます。このタブは、クラスターを編集しているときにも表示されます。

> **注釈**
>
> クラスターを編集するときは、シングルノードまたは高可用性クラスターと、高度なまたは分散高可用性クラスターを切り替えることはできません。

- ***Single Node*** は、1 つのプライマリとスタンバイレプリカを持たないクラスターを作成します。このタイプのクラスターは、高可用性が必要とされないテスト環境に適しています。PostgreSQL、EDB Postgres Extended Server、または EDB Postgres Advanced Server を実行するシングルノードクラスターを作成できます。

- ***High Availability*** は、異なるアベイラビリティーゾーンに 1 つのプライマリと複数のスタンバイレプリカを含むクラスターを作成します。PostgreSQL、 EDB Postgres Extended Server、または EDB Postgres Advanced Server を実行する高可用性クラスターを作成できます。

高可用性を選択すると、クラスターのスタンバイレプリカの数を指定するように求められます。デフォルトは 2 です。

- ***Advanced High Availability*** は、高速な自動フェイルオーバーを備えたシングルリージョン、マルチ AZ の復元力を提供します。オンラインアップグレード。プライマリ/スタンバイノードを介した論理レプリケーション。

Advanced High Availability を選択すると、タブ **DB Configuration** および **追加設定** は表示されません。代わりに [**Data Groups**] タブが表示されます。

- ***Distributed High Availability*** は、 EDB Postgres DistributedPGD を利用して、3 つのノードと 3 つのアベイラビリティーゾーンにまたがる最大 2 つのデータグループを使用してクラスターを作成します。この構成は、より高いパフォーマンスとより速いリカバリーを実現します。PostgreSQL、 EDB Postgres Extended Server、または EDB Postgres Advanced Server を実行する分散高可用性クラスターを作成できます。

分散高可用性 を選択すると、 **DB 構成** および **追加設定**タブが表示されません。代わりに [**Data Groups**] タブが表示されます。

詳細については、 Supported cluster types を参照してください。

Cluster setup

[**クラスター設定**] タブで、名前やパスワードなどのクラスター設定を構成します。

Cluster Name — クラスターの名前を入力します。

Password —クラスターのパスワードを入力します。これは、ユーザー edb_admin のパスワードです。このパスワードは、データベースに接続するために必要であり、ハイブリッドマネージャー HM には保存されません。

タグ — オプションで、 **+ Add Tag** を選択して、クラスターにタグを追加します。タグは、リソースを整理するのに役立つキーと値のペアです。

- 既存のタグを割り当てるには、 **タグ**の下の検索バーで、割り当てるタグが表示されるまでタグ名前の入力を開始します。クラスターに割り当てるタグを選択します。
- 新しいタグを追加するには、完全なタグ名を入力し、 **+ Add ``<tagname>``** を選択します。次に、タグの色を選択し、 **Create** を選択します。新しいタグがクラスターに割り当てられます。

シングルノードおよび高可用性クラスターの設定

次のオプションは、シングルノードおよび高可用性クラスターの場合にのみ**クラスター設定**タブに表示されます。他のクラスタータイプの場合は、 ***Data Groups*** タブでこれらのオプションを設定します。分散高可用性または高度な高可用性クラスターを作成する場合は、**Next: Data Groups** を選択してそのタブに移動します。

展開の場所 —クラスターを展開する場所を選択します。デフォルトの場所は、 EDB Postgres AI サービスをホストする Kubernetes クラスター内です。マルチ HM 環境で HM が構成されている場合、クラスターを

展開する場所を選択できます。

データベースタイプ — 使用する Postgres のタイプを選択します。

1. **Postgres Type** フィールドで、使用する Postgres のタイプを選択します。

- ** EDB Postgres Advanced Server は、EDB の Oracle 互換データベース製品です。マイグレーション を表示します。 EDB Postgres Advanced Server は、すべてのクラスタータイプと互換性があります。

-** EDB Postgres Extended Server は、高度な論理レプリケーションを使用する EDB の PostgreSQL 互換データベース製品です。

- ** PostgreSQL は、オープンソースのオブジェクトリレーショナルデータベース管理システムです。PostgreSQL は、シングルノードおよび高可用性クラスタータイプと互換性があります。

2. **Postgres Version** リストで、使用する Postgres のバージョンを選択します。

3. HM は、画像ライブラリに保存されている各 Postgres データベースの複数の画像をサポートしています。各イメージは、さまざまな拡張子を含むデータベースの構成です。クラスターに使用するイメージを選択します。詳細については、*Asset library* を参照してください。通常、名前の最後に`-full` がないと、画像に拡張子はありません。名前の最後に`-full` を付けると、画像には使用可能なすべての拡張子が含まれます。

インスタンスサイズ —クラスターの CPU の数とメモリの量を選択します。選択できる CPU の数とメモリの量は、Kubernetes クラスターで使用可能なリソースによって異なります。

ストレージ

次のストレージ設定を指定できます。

先行書き込みログ用に別のストレージボリュームを使用します —デフォルトでは、HM はデータベースストレージとログ先行書き込み WAL ファイルに別のストレージボリュームを使用します。このデフォルトでは、最高のパフォーマンスと信頼性が得られます。チェックボックスをオフにして、WAL ファイルをデータベースストレージと同じボリュームに保存します。

デフォルトでは、HM はデータベースストレージと WAL ファイルに別個のストレージボリュームを使用します。このデフォルトでは、最高のパフォーマンスと復元力が得られます。このオプションをクリアすると、WAL ファイルはデータベースストレージと同じボリュームに保存されます。

データベースストレージ - **データベースストレージ**の下で、データベースストレージクラスとデータベースストレージボリュームのサイズを GiB 単位で選択できます。使用可能なストレージクラスは、基になるインフラストラクチャに設定した内容によって異なります。たとえば、EKS 展開の場合は gp2 および gp3、または RHOS 展開の場合は ssd-csi および standard-csi など。

Write-Ahead Logs Storage — **Write-Ahead Logs Storage** の下で、WAL ストレージクラスと WAL ストレージボリュームのサイズを GiB 単位で選択できます。使用可能なストレージクラスは、基になるインフラストラクチャに設定した内容によって異なります。たとえば、EKS 展開の場合は gp2 および gp3、または RHOS 展開の場合は ssd-csi および standard-csi など。

ネットワーキング

Connectivity Type では、プライベートネットワークデフォルトまたはパブリックネットワークを使用するかどうかを選択します。

- **プライベート**ネットワークでは、プライベートネットワーク内の IP アドレスのみがクラスターに接続できます。
- **パブリック**ネットワークとは、任意のクライアントがインターネットを介してクラスターのパブリック IP アドレスに接続できることを意味します。

Database configuration parameters

シングルノードまたは高可用性クラスターを作成または編集するときは、[**DB Configuration**] タブを使用してデータベース構成パラメーターを指定します。

[**DB Configuration**] タブにリストされているデータベースパラメーターは、グランドユニファイド構成 GUC 変数とも呼ばれます。詳細については、What Is a GUC Variable? を参照してください。

パラメーターのリストは、クラスターを作成したときに [**クラスター設定**] タブで選択したデータベースのタイプに基づいて入力されます。データベースタイプのパラメーターの詳細については、

- PostgreSQL パラメーターについては、Postgres ドキュメントの

Setting Parameters および Server Configuration を参照してください。

- EDB Postgres Advanced Server の場合、Summary of configuration parameters および Configuration parameters を参照してください。
- パラメーターが受け入れる値の種類の詳細については、

Parameter Names and Values
を参照してください。これらのタイプは、PostgreSQL と EDB Postgres Advanced Server パラメーターの両方に適用されます。

> **注釈**
>
> クラスターの作成が完了した後、データベースまたはサーバーパラメーターを変更できます。基になるハードウェアの物理的制限内でメモリおよび CPU 関連のデータベースパラメーターを構成してください。一部のパラメーターを変更するには、再起動が必要です。デフォルトの `wal_segment_size` は、PSR クラスター展開の場合は 64MB、PGD クラスター展開の場合は 128MB に設定されます。

パラメーターの変更

1. 次の方法を使用して、変更するパラメーターを見つけます。
    - 検索フィールドを使用してパラメーターを検索します。
    - **Show Only** リストのオプションを選択して、リストされたパラメーターをフィルターします。
        - **Custom Values** —現在のセッションで、または以前のセッションで変更されたデフォルト値から変更されたパラメーターのみを表示します
        - **現在編集中の値** —現在の編集セッション中に変更されたパラメーターのみを表示します。

2. パラメーター値フィールドに新しい値を入力します。エントリーには、リテラル値、または他のパラメーターに基づいて値を計算する *Formulas for parameter values* を指定できます。

> **警告**
>
> 黄色の感嘆符アイコンで示されるパラメーターは、変更を保存するときに再起動をトリガーします。再起動により、開いている接続がすべて終了され、完了までに数分かかります。新しい接続が確立されるまでにさらに数秒かかります。このプロセス中に、データベースに接続することはできません。

Backup and security settings

[**追加設定**] タブでは、バックアップ保持期間と一部のセキュリティ機能を指定できます。

バックアップ

バックアップ セクションの **保持期間**コントロールを使用して、デフォルトのデータベースバックアップの保持期間 30 日を変更します。保持期間は日、週、または月単位で構成できます。1〜180 日、1〜25 週間、または 1〜6 か月の範囲で保持期間を設定します。

セキュリティ

必要に応じて、 encrypt your cluster's data を実行できます。

このオプションは、バージョン 15 以降の EDB Postgres Advanced Server および EDB Postgres Extended Server で利用できます。

Transparent Data Encryption TDE) トグルを使用して、TDE を有効にします。

!!!注

クラスターの作成後、TDE 構成を有効、無効、または変更することはできません。

次のオプションを設定できます。

- **パスフレーズを入力します** 。TDE 対応クラスターを作成するときに、パスフレーズを使用したこのオプション secures the data encryption key created by Hybrid Manager (HM) 。
- HM インスタンスの管理者が以前に *enabled a key from a key management system* を使用していた場合、**Customer Key Management** を選択することもできます。次に、構成されたキーのいずれかを選択して、ハイブリッドマネージャーによって作成されたデータ暗号化キーを保護します。

TDE の詳細については、 Transparent Data Encryption を参照してください。

Creating data groups

[**Data Groups**] タブは、分散高可用性または高度な高可用性クラスターを作成または編集するときに使用できます。それを使用して、高度な分散高可用性クラスターの**データグループ**を作成および管理します。データグループは、通常、グループ内でデータを複製し、アプリケーションクエリに応答するように構成されたデータノードのリージョンに近いグループです。分散高可用性クラスターは、相互に複製する 1 つ以上のデータグループで構成でき、地理分散クラスターが有効になります。

データグループの作成

1. **Cluster Info** タブで **Advanced High Availability** または **Distributed High Availability** を選択した状態で、 **Data Groups** タブを選択します。
2. ***Node Settings*** タブを選択し、必要な情報を入力します。
3. ***DB Configuration*** タブを選択し、必要に応じてパラメーターを変更します。
4. **Create Data Group** を選択します。

新しいデータグループは [**Data Groups**] タブにリストされます。クラスターアーキテクチャ図を展開して、データグループの概要を表示します。

別のデータグループを追加するには、 **Add Data Group** を選択します。

> **注釈**
>
> 2 つのデータグループを作成する場合、監視グループが自動的に追加されます。データグループと同様に監視グループを編集できますが、オプションはより制限されています。グループの名前、展開場所、データベースタイプ、バージョン、イメージ、インスタンスサイズを設定できます。

ノード設定

[**ノード設定**] タブを使用して、データグループを設定します。

ノード — グループ内のノードの構成を選択します。

推奨される 3 データノード構成または 2 データノード構成を選択できます。高度な高可用性クラスターの場合、1 ノード構成を作成できます。ノードアーキテクチャの詳細については、*Distribution high availability* を参照してください。

展開の場所 —クラスターを展開する場所を選択します。デフォルトの場所は、 EDB Postgres AI サービスをホストする Kubernetes クラスター内です。マルチ HM 環境でハイブリッドマネージャー HM が構成されている場合、クラスターを展開する場所を選択できます。

データベースタイプ — 使用する Postgres のタイプを選択します。

1. **Postgres Type** フィールドで、使用する Postgres のタイプを選択します。

- EDB Postgres Advanced Server は、EDB の Oracle 互換データベース製品です。EDB Postgres Advanced Server は、すべてのクラスタータイプと互換性があります EDB Postgres Extended Server は、高度な論理レプリケーションを使用する EDB の PostgreSQL 互換データベース製品です。
- PostgreSQL は、オープンソースのオブジェクトリレーショナルデータベース管理システムです。PostgreSQL は、シングルノードおよび高可用性クラスタータイプと互換性があります。

2. **Postgres Version** リストで、使用する Postgres のバージョンを選択します。
3. HM は、画像ライブラリに保存されている各 Postgres データベースの複数の画像をサポートしています。各イメージは、さまざまな拡張子を含むデータベースの構成です。クラスターに使用するイメージを選択します。詳細については、*Asset library* を参照してください。通常、名前の最後に `-full` のない画像には拡張子はありません。名前の最後に `-full` を付けると、画像には PostGIS を除く使用可能なすべての拡張子が含まれます。

インスタンスサイズ —クラスターの CPU の数とメモリの量を選択します。選択できる CPU の数とメモリの量は、Kubernetes クラスターで使用可能なリソースによって異なります。

ストレージ

次のストレージ設定を指定できます。

先行書き込みログ用に別のストレージボリュームを使用します —デフォルトでは、HM はデータベースストレージとログ先行書き込み WAL ファイルに別のストレージボリュームを使用します。このデフォルトでは、最高のパフォーマンスと信頼性が得られます。チェックボックスをオフにして、WAL ファイルをデータベースストレージと同じボリュームに保存します。

データベースストレージ - **データベースストレージ**の下で、データベースストレージクラスとデータベースストレージボリュームのサイズを GiB 単位で選択できます。使用可能なストレージクラスは、基になるインフラストラクチャに設定した内容によって異なります。たとえば、EKS 展開の場合は gp2 および gp3、または RHOS 展開の場合は ssd-csi および standard-csi など。

Write-Ahead Logs Storage — **Write-Ahead Logs Storage** の下で、WAL ストレージクラスと WAL ストレージボリュームのサイズを GiB 単位で選択できます。使用可能なストレージクラスは、基になるインフラストラクチャに設定した内容によって異なります。たとえば、EKS 展開の場合は gp2 および gp3、または RHOS 展開の場合は ssd-csi および standard-csi など。

ネットワーキング

Connectivity Type では、プライベートネットワークデフォルトまたはパブリックネットワークを使用するかどうかを選択します。

- **プライベート**ネットワークでは、プライベートネットワーク内の IP アドレスのみがクラスターに接続できます。
- **パブリック**ネットワークとは、任意のクライアントがインターネットを介してクラスターのパブリック IP アドレスに接続できることを意味します。

バックアップ

バックアップ セクションの **保持期間**コントロールを使用して、デフォルトのデータベースバックアップの保持期間 30 日を変更します。保持期間は日、週、または月単位で構成できます。1〜180 日、1〜25 週間、または 1〜6 か月の範囲で保持期間を設定します。

セキュリティ

このセクションは、データベースタイプとして **EDB Postgres Advanced Server** または **EDB Postgres Extended Server** を選択し、バージョンとイメージを選択すると表示されます。ここでの変更はすべてのデータグループに適用されます。

透過的データ暗号化 TDE を有効にして、独自の暗号化キーを使用します。このオプションは、バージョン 15 以降の EDB Postgres Advanced Server および EDB Postgres Extended Server で利用できます。

Enable TDE チェックボックスを選択して **TDE** を有効にします。

TDE サポートの詳細については、Transparent Data Encryption を参照してください。

データベースパラメーター構成

DB Configuration タブを選択して、データベースのパラメーターを構成します。

Parameters セクションでは、必要に応じてデータベース構成パラメーターの値を更新できます。

[**DB Configuration**] タブにリストされているデータベースパラメーターは、グランドユニファイド構成 GUC 変数とも呼ばれます。詳細については、What Is a GUC Variable? を参照してください。

パラメーターのリストは、クラスターを作成したときに [**クラスター設定**] タブで選択したデータベースのタイプに基づいて入力されます。データベースタイプのパラメーターの詳細については、

- PostgreSQL パラメーターについては、Postgres ドキュメントの

Setting Parameters および Server Configuration を参照してください。

- EDB Postgres Advanced Server の場合、Summary of configuration parameters および Configuration parameters を参照してください。
- パラメーターが受け入れる値の種類の詳細については、

Parameter Names and Values を参照してください。これらのタイプは、 PostgreSQL と EDB Postgres Advanced Server パラメーターの両方に適用されます。

> **注釈**
>
> クラスターの作成が完了した後、データベースまたはサーバーパラメーターを変更できます。一部のパラメーターを変更するには、再起動が必要です。

パラメーターの変更

1. これらの方法を使用して、変更するパラメーターを見つけます。
   - 検索フィールドを使用して特定のパラメーターを検索します。
   - **Show Only** リストのオプションを選択して、リストされたパラメーターをフィルターします。
     - **Custom Values** —現在のセッションで、または以前のセッションで変更されたデフォルト値から変更されたパラメーターのみを表示します
     - **現在編集中の値** —現在の編集セッション中に変更されたパラメーターのみを表示します。
2. パラメーター値フィールドに新しい値を入力します。エントリーには、リテラル値、または他のパラメーターに基づいて値を計算する *Formulas for parameter values* を指定できます。

> **警告**
>
> 黄色の感嘆符アイコンで示されるパラメーターは、変更を保存するときに再起動をトリガーします。再起動により、開いている接続がすべて終了され、完了までに数分かかります。新しい接続が確立されるまでにさらに数秒かかります。このプロセス中に、データベースに接続することはできません。

データグループの管理

データグループを複製、編集、および削除できます。

- グループを複製するには、 **Duplicate** を選択します。次に、新しいグループを編集して元のグループと区別します。
- **Edit** を選択して、データグループを調整します。
- データグループを削除するには、 **Delete** を選択します。削除の確認を求められます。
- データグループを追加するには、 **Add Data Group** を選択します。

Creating a cluster from a template

プロジェクトにある場合は、テンプレートを使用してデータベースクラスターを作成できます。テンプレートを使用すると、時間を節約し、環境間の一貫性が保証されます。

テンプレートを作成するには、 *Create cluster templates* を参照してください。

スタートポイント

Choose Your Path ページまたは **Cluster Templates** ページのテンプレートからクラスターの作成を開始できます。

[パスの選択] ページから

Estate ページまたはプロジェクトページから **Choose Your Path** ページにアクセスできます。

エステート ページから

1. **Quick Actions > Create New Database Cluster** を選択します。

複数のプロジェクトがある場合、プロジェクトを選択するように求められます。

2. プロンプトが表示されたら、データベースを作成するプロジェクトを選択します。

プロジェクトを選択した後、またはプロジェクトが 1 つだけの場合、 **Choose Your Path** ページが開きます。

プロジェクトページから、これらのアプローチのいずれかを使用できます。プロジェクトページに移動するには、**Projects** ページからプロジェクトを選択します。

- プロジェクトページで、 **Create New > Database Cluster** を選択します。
- プロジェクトページで、**Clusters > Postgres Cluster** を選択します。[**クラスター**] ページで、[**Create New > Database Cluster**] を選択します。

Choose Your Path ページが開きます。

[Choose Your Path] ページのテンプレートを使用する

1. **Choose Your Path** ページで、 **Use Template** を選択します。
2. テンプレートを選択します。使用可能なテンプレートのリストから、ニーズに合ったものを選択します。

3. **Use Template** を選択します。

クラスター作成モードに入ります。 クラスターのカスタマイズと作成 を参照してください。

[クラスターテンプレート] ページから開始します

1. プロジェクトページの左側のナビゲーションで、 **Cluster > Cluster Templates** を選択します。
2. クラスターのベースとして使用するテンプレートの横にある **Use** アイコンを選択します。

クラスター作成モードに入ります。 クラスターのカスタマイズと作成 を参照してください。

クラスターのカスタマイズと作成

テンプレートを選択したら、作成しているクラスターの設定を確認および変更できます。詳細については、以下を参照してください。

- *Cluster types to create*
- *Cluster setup*
- *Database configuration parameters*
- *Backup and security settings*
- *Creating data groups* 高度なまたは分散高可用性クラスター用

作業する別のテンプレートを選択することもできます。 **Cluster Templates** タブを選択して、使用可能なテンプレートのリストを表示します。

タブを使用してクラスターの構成が終了したら、[**Create Cluster**] を選択します。設定が正しくないまたは欠落しているタブには、赤いインジケーターが表示されます。エラーを修正し、**Create Cluster** を再度選択します。

有効な設定で **Create Cluster** を選択すると、クラスターはプロビジョニングを開始し、クラスターのリストに戻ります。ポップアップはクラスターの作成を確認し、新しいクラスターがクラスターのリストに追加されます。クラスターはバックグラウンドで作成されます。クラスターのリストで進行状況を確認できます。

クラスターの作成の進行中にクラスターの詳細を表示しようとすると、概要、バックアップ、ログ、プロパティのみを表示できます。

Creating a cluster programmatically

Postgres クラスターの作成を自動化することにより、再現可能なビルドを合理化できます。

Bash スクリプトとハイブリッドマネージャー API を使用して、再現可能なビルドの Postgres クラスターの作成を自動化します。すべての構成をソース管理にチェックして、変更を追跡し、ドキュメントを維持します。

マシンユーザーを作成する

1. HM コンソールの右上隅にあるドロップダウンメニューに移動します。
2. ドロップダウンメニューから **User Management** を選択します。
3. 右上隅にある **Add new user** を選択します。
4. **Add new user** が表示されるポップアップウィンドウに、名前を入力し、チェックボックス **Create access key** をチェックします。
5. フィールド **有効期限** に入力します。
6. ポップアップウィンドウで、[**Add user**] を選択します。
7. ポップアップウィンドウからアクセスキーをコピーします。
8. **キーは安全に保存されました** を選択します。

マシンにユーザー権限を付与します

> **注釈**
>
> マシンユーザー権限を正しく設定するには、左側のナビゲーションバーから Postgres クラスターを作成するターゲットプロジェクトを選択します。

1. 左上隅で **Projects** を選択し、プロジェクトに移動します。
2. 左側のナビゲーションバーで、 **Users** を選択します。
3. 検索バーで、作成したマシンユーザーを探します。
4. **Machine user** の右側にある鉛筆編集アイコンを選択します。
5. **役割の割り当て** を選択します。
6. **プロジェクトのロールの割り当て** の下で、チェックボックス**プロジェクトの所有者** をチェックします。
7. **Submit** を選択します。

env.sh ファイルを作成する

プロジェクト ID を `env.sh` ファイルと API ホストに追加することにより、シェルスクリプトの記述を簡素化できます。

コピーしたアクセスキーを `env.sh` という名前のファイルに追加します。

```
export HCP_ACCESS_KEY="baak_xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
export HCP_API_ACCESS_KEY=${HCP_ACCESS_KEY}
export HCP_PROJECT_ID=prj_xxxxxxxxxxxxxx
export HCP_API_HOST="https://portal-uat-eks.aws.hcp.enterprisedb.network"
```

AIDB および PGFS を使用して Docker イメージを見つけます

場所とその場所で使用可能な Docker イメージのリストを取得するには、01-get-locations.sh というスクリプトを作成して実行します。次のコンテンツが表示されます。

```
# !/bin/bash

set -e
set -u
set -o pipefail

#  list locations

source env.sh

curl \
  --insecure \
  --header "Content-Type: Application/JSON" \
  --header "x-access-key: ${HCP_API_ACCESS_KEY}" \
  --request GET \
  ${HCP_API_HOST}/api/v1/projects/${HCP_PROJECT_ID}/locations \
  | jq
```

次のスクリプトを実行します。

```
./01-get-locations.sh > locations.json
```

locations.json を調べて、pgExtensions 、aidb 、および pgfs の下にリストされているイメージを探します。

Postgres クラスターを作成する

前の手順で取得したイメージ名を含む create-psr-cluster.json という名前のファイルを作成します。

```
{
  "projectId": "${HCP_PROJECT_ID}",
  "psr": {
    "clusterName": "small-ai-api-001",
    "password": "111111111111",
    "locationId": "managed-devspatcher",
    "clusterData": {
      "instances": 1,
      "resourceRequest": {
        "memory": "2Gi",
        "cpu": "2"
      },
      "storageConfiguration": {
```

(次のページに続く)

(前のページからの続き)

```
        "primaryStorage": {
          "size": "4",
          "storageClass": "hcp-aws-gp3"
        },
        "walStorage": {
          "size": "4",
          "storageClass": "hcp-aws-gp3"
        }
      },
      "image": {
        "url": "upmdev.azurecr.io/postgresql:17.6-2510160831",
        "digest":
↪"sha256:6bda61e6d75c3b9d6a886c547d789c46b88ed807ae28d8d06fa1510a0cc5afae"
      },
      "networkAccessType": "NetworkAccessTypePublic"
    }
  }
}
```

create-psr-cluster.json を使用し、クラスターを作成するように 02-create-psr-cluster.sh というシェルスクリプトを作成します。

```
# !/bin/bash

set -e
set -o pipefail
set -u

source env.sh

cat create-psr-cluster.json | envsubst > /tmp/create-psr-cluster.json

curl \
  --insecure \
  --header "Content-Type: Application/JSON" \
  --header "x-access-key: ${HCP_API_ACCESS_KEY}" \
  --data-binary "@/tmp/create-psr-cluster.json" \
  --request POST \
  ${HCP_API_HOST}/api/v1/projects/${HCP_PROJECT_ID}/clusters
```

シェルスクリプトを実行します。

```
./02-create-psr-cluster.sh | tee ./psr-cluster.json
```

HM コンソールでクラスター作成の進行状況を確認できます。

新しいクラスターに AIDB および PGFS 拡張機能を作成します

AIDB および PGFS の機能を、プログラムで作成したクラスターに拡張機能として追加できます。

PostgreSQL 認証資格情報を含めるように env.sh を構成する

以下を env.sh ファイルに追加します。例

```
export PGPASSWORD=your_postgresql_password
export PGUSER=edb_admin
export PGHOST=your-postgresql-hostname
export PGPORT=5432
export PGDATABASE=edb_admin
export PGSSLMODE=require
```

クラスター拡張機能を作成する

次の内容を使用して 03-create-ai-extensions.sh という名前の bash ファイルを作成します。

```
# !/bin/bash

set -e
set -o pipefail
set -u

source env.sh

psql -X <<EOF
create extension aidb cascade;
create extension pgfs cascade;
EOF
```

スクリプトを実行します。

```
$ chmod +x 03-create-ai-extensions.sh
$ ./03-create-ai-extensions.sh
```

クラスター検証

HM コンソールを介してクラスターの作成を確認した後、API を介して新しいクラスターを取得して、設定を確認できます。JSON 形式でクラスターを取得することは、必要な JSON フィールドを指定するのにも役立ち、自動クラスター作成のブループリントとして機能し、各設定フィールドを個別に検索することを回避できます。

注釈

Create-cluster JSON と Get-cluster JSON は、異なる情報を含めることができる 2 つの異なる機能です。
!!!

次を含む get-existing-cluster.sh という名前のファイルを作成します。

```
# !/bin/bash

set -e
set -u
set -o pipefail

source env.sh

HCP_PG_CLUSTER_ID=p-fj3ki6e0x0

curl \
  --insecure \
  --header "Content-Type: Application/JSON" \
  --header "x-access-key: ${HCP_API_ACCESS_KEY}" \
  --request GET \
  ${HCP_API_HOST}/api/v1/projects/${HCP_PROJECT_ID}/clusters/${HCP_PG_CLUSTER_ID} \
  | jq
```

HCP_PG_CLUSTER_ID はクラスター ID を指定します。

クラスターのコンテンツを JSON ファイルに保存します。

```
./get-existing-cluster.sh > existing-cluster.json
```

注釈

クラスター作成の必須フィールド、命名規則、および JSON 構造の詳細については、

API documentation を確認してください。

Managing clusters

いくつかの管理タスクは、ハイブリッドマネージャー HM のクラスターに関連付けられています。

クラスター作成タスク

クラスター作成プロセスの一部として、次のことを行うことができます。

- 新しいテンプレートのベースとして使用できる *Create a cluster template*
- *Create a formula* 特定の構成パラメーターの値として使用する。

既存のクラスターでの作業

create a cluster 後、クラスター名の右側にある 3 点メニューを使用して、次のことを行うことができます。

- クラスターを編集する 構成の大部分を変更します。
- クラスターパスワードを変更します。
- *Delete a cluster* 。
- クラスターをアップグレードします。マイナーアップグレードまたはメジャーアップグレードを実行できます。
- *Restore a deleted cluster* 。

メニューには、現在のプロジェクトユーザーが使用可能なオプションのみが含まれます。

Postgres クラスターの実行状況を分析する

クラスターの実行中に、Postgres クラスターが実行されている Kubernetes インフラストラクチャのステータスを確認する必要がある場合があります。これは、*tracing the cluster back to its Kubernetes resources* で行うことができます。たとえば、Grafana を使用して、リソースがどのように使用されているかをより深く理解できます。

Create a database cluster template

将来のクラスターのベースとして使用するクラスターテンプレートを作成できます。テンプレートの作成は、クラスターの作成に似ています。すべて同じ情報を指定する必要があります。

スタートポイント

開始点は、テンプレートを作成するプロジェクトのランディングページです。そこに到達するには、**プロジェクト** ページからプロジェクトを選択します。

プロジェクトページの左側のナビゲーションで、 **Clusters > Cluster Templates** を選択します。

Cluster Templates ページには、Lakehouse Analytics テンプレートを含むプロジェクト内のテンプレートがリストされます。

Create Template > Create Database Cluster Template を選択します。

この点から、クラスター構成の指定は、クラスターを作成する場合と同じです。**Next** を使用してタブを順番にステップ実行するか、作業するタブを選択できます。詳細については、以下を参照してください。

- *Cluster types to create*
- *Cluster setup*
- *Database configuration parameters*
- *Backup and security settings*
- *Creating data groups* 高度なまたは分散高可用性クラスター用

クラスターテンプレートの構成が終了したら、[**Create Template**] を選択します。

新しいテンプレートがテンプレートのリストに追加されます。

テンプレートの削除

テンプレートは、不要になった場合は **クラスターテンプレート**ページから削除できます。

> **警告**
>
> 削除されたテンプレートを復元または復元することはできません。

1. テンプレートを削除するプロジェクトのページの、左側のナビゲーションで、**クラスター > クラスターテンプレート** を選択します。
2. 削除するテンプレートの横にある **Delete** アイコンを選択します。
3. 確認ボックスに `delete template` と入力し、 **Delete** を選択します。

テンプレートの使用については、 *Creating a cluster from a template* を参照してください。

Deleting clusters

クラスターは、 **Postgres クラスター**ページおよびクラスターの詳細ビューから削除できます。

- プロジェクトページから、 **Clusters > Postgres Clusters** を選択します。削除するクラスターの横にある 3 点メニューから、 **Delete Cluster** を選択します。
- プロジェクトページから、 **Clusters > Postgres Clusters** を選択します。削除するクラスターを選択します。クラスターの詳細ビューから、**Quick Actions > Delete Cluster** を選択します。

削除の確認をするプロンプトが表示されたら、 **Delete** を選択します。

Editing a cluster

Postgres クラスターページまたはクラスターの詳細ビューからクラスターの編集を開始できます。

- **Postgres Clusters** ページで、編集するクラスターの横にある **Edit** を選択します。
- クラスターの詳細ビューを表示するには、 **Postgres Clusters** ページで、クラスター名前を選択します。 [**Cluster Details**] ページで、[**Quick Actions > Edit Cluster**] を選択します。

> **注釈**
>
> クラスターの作成が完了した後、データベースまたはサーバーパラメーターを変更できます。一部のパラメーターを変更するには、再起動が必要です。

クラスターの編集

Edit Cluster ページは、 **Create Cluster** ページに似ています。作業するタブを選択するか、 **Next** を使用してタブを順番に実行します。

- ***Cluster Info***
- ***Cluster Settings***
- ***DB Configuration*** オプショナル

- ***Additional Settings*** オプショナル
- ***Data Groups*** 分散高可用性クラスター用

> **注釈**
>
> クラスターを編集するときに、シングルノードまたは高可用性クラスターと、高度なまたは分散高可用性クラスターの間で切り替えることはできません

編集の保存と確認

クラスター構成の変更が完了したら、[**Save**] を選択します。

構成が有効な場合、クラスターに行う変更のリストが表示されます。変更によりサーバーが再起動される可能性があることを示す警告が表示される場合があります。**confirm Changes** を選択して変更をクラスターに適用し、元のページに戻ります。

> **注釈**
>
> ページの上部にある**戻る**矢印を選択すると、変更が破棄され、元のページに戻ります。

Restoring clusters

削除されたクラスターをバックアップから復元できます。バックアップは、復元するクラスターと同じプロジェクトにある必要があります。復元プロセスは、技術的には、削除されたクラスターと同じ構成で新しいクラスターを作成します。これは実際には新しいクラスターであるため、新しい名前と新しいパスワードを指定する必要があります。

復元されたクラスターは、削除されたクラスターと同じプロジェクトに作成されます。バックアップが利用可能な限り、削除されたクラスターを復元できます。バックアップは、クラスターごとに設定された保持期間の間利用可能なままです。保持期間が経過すると、バックアップは削除されます。

バックアップからのクラスターの復元

1. **クラスター> Postgres クラスター** を選択します。
2. **Deleted** タブを選択します。
3. 復元する削除されたクラスターの横にある **Restore** アイコンを選択します。
4. **Restore Cluster** ダイアログボックスで、復元するバックアップを選択し、 **Restore** を選択します。

Restore Cluster ページが開きます。このページは、 **Create Cluster** ページに似ています。クラスター名前、クラスタータイプ、および復元するために選択したバックアップが表示されます。必要に応じて、クラスター名とクラスタータイプを変更できます。

5. 新しいパスワードを入力します。
6. [**クラスター設定**] タブで、オプションでクラスター設定を変更できます。 **Restore Type** の下で、実行する復元のタイプを選択します。

- **完全な復元** - データベースの最後の完全なバックアップからクラスター全体を復元します。このオプションを選択すると、その下のリストから復元するバックアップを選択できます。
- **ポイントインタイムリストア** - クラスターを特定のポイントインタイムに復元します。バックアップの保持期間内の日付と時刻を選択します。範囲は日付と時刻ピッカーの下に表示されます。

7. 必要に応じて他のタブを使用します。詳細については、*Cluster creation settings* を参照してください。
8. **Restore Cluster** を選択します。

Postgres Clusters ページが開き、復元操作の進行状況を確認できます。復元されたクラスターは、元のクラスターと同じプロジェクトにあります。そのクラスターと同じ構成ですが、新しい名前とパスワードがあります。

Tracing clusters back to their Kubernetes resources

Hybrid ManagerHM で Postgres クラスターを管理しているときに、Postgres クラスターが実行されている Kubernetes インフラストラクチャのステータスを確認する必要がある場合があります。これらは Kubernetes ワーカーノードです。たとえば、HM インターフェイスで問題が表示されているため、またはリソースが不足しているためにノードに障害が発生しているかどうかを確認することができます。

クラスターを Kubernetes リソースまでトレースするのに役立つ情報を取得するには、いくつかの方法があります。

- *HM を使用してクラスターをその名前空間に照合する* より
- *Grafana ハードウェア使用率ダッシュボード* より
- *Loki と Fluent Bit* を使用する
- あなたが管理者の場合 *kubectl を使用して基になるクラスターリソースを確認する* を使用する

分散高可用性クラスターをトレースバックする場合、高可用性またはシングルノードクラスターのトレースとのいくつかの違いに注意する必要があります。**分散高可用性クラスターのトレースバック** を参照

HM を使用してクラスターをその名前空間に照合する

HM を使用して作成された Postgres クラスターをその基になる Kubernetes リソースまでトレースバックする場合、最初に開始する場所は HM のクラスターの**概要** ページです。そのページに移動するには、**Estate** ページで、クラスターを選択します。そこで、ブラウザーの URL の最後を見て、クラスターに関連付けられている Kubernetes 名前空間を見つけます。

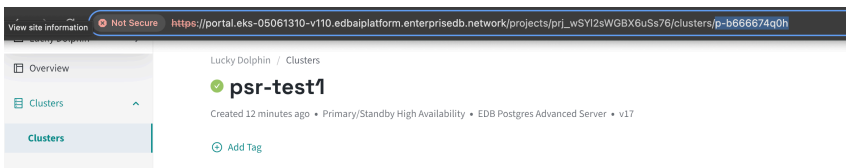


図 4: Browse cluster

この例では、クラスター psr-test1 は名前空間 `p-b666674q0h` に関連付けられます。クラスターをそのリソースまでさらにトレースするには、関連する名前空間がパズルの重要な最初のピースです。それを使用すると、さらに多くのことを見つけることができます。

探索しているクラスターの名前空間を取得したら、次の目的はビルトインの Grafana ダッシュボードです。

Grafana ハードウェア使用率ダッシュボード

Grafana ダッシュボードにアクセスするには、HM ランチパッドを使用します。プロフィールメニューから、**Launchpad** を選択します。ラウンチパッドの **Grafana** タイルで、**View Service** を選択します。

Grafana のランディングページの **Search** フィールドに、`Hardware Utilization` と入力します。**ハードウェア使用率**ダッシュボードでは、基になる Kubernetes ワーカーノードと、一部で実行されている Postgres ワークロードを含む Kubernetes ポッドを表示できます。

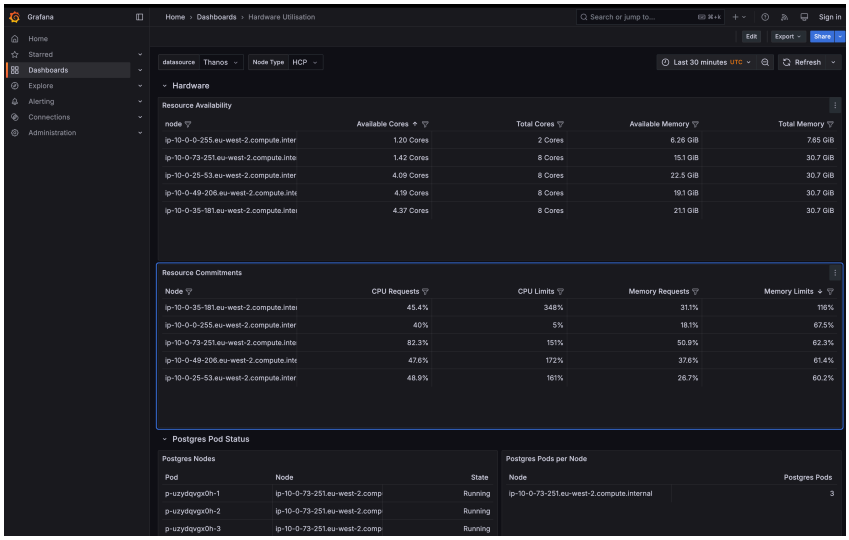


図 5: Hardware Utilization dashboard

ポッド

Postgres Pod Status セクションでは、調査している Postgres クラスターに関連付けられた名前空間を使用して、Postgres クラスターが関連付けられている名前空間と同じ指定で名前が始まるポッドを特定します。ワークロードがポッド間で分割されるため、これらのポッド名には`-1`、`-2`、`-3` などのサフィックスが付けられます。 Postgres Nodes テーブルから、State 列にポッドの状態を確認し、Node 列にポッドが実行されているノードを確認することもできます。

リソースの可用性

ポッドと基になる Kubernetes ワーカーノードを照合した後、**リソースの可用性**の下のノード名を使用して、これらのノードで使用可能なリソースに関する詳細情報を取得できます。

- **利用可能なコア**ワークロードが必要な分を使用した後に残っている利用可能なコアの数。
- **合計コア**ワークロードから使用される前にノードが必要とするコアの合計数。ワーカーノードには複数のポッドが存在する場合があります。
- **利用可能なメモリ**ワークロードが必要な分を利用した後に利用可能なメモリの量。
- **合計メモリ**何も使用していないワークロードのないノードの合計メモリ。

リソースのコミットメント

Postgres ワークロードを調べたい Kubernetes ワーカーノードがある場合、ダッシュボードのもう 1 つの役立つセクションは**リソースコミットメント**テーブルです。ここでは、関心のあるポッドの基になるノードを含む、各ノードでどのような要求が実行されているかを確認できます。

- **CPU 要求**プロビジョニングされたワークロードによって要求されたノードの CPU 量、または HM によってプロビジョニングされています。
- **CPU 制限** ノード上のポッドからの CPU 制限の合計。
- **メモリ要求**プロビジョニングされたワークロードによって要求されたノードからのメモリ量、または HM によってプロビジョニングされています。
- **メモリ制限** ノード上のポッドからのメモリ制限の合計。

リクエスト

CPU とメモリ要求は両方とも、その上で実行されているポッドによるノードの要求を満たすことが保証されていますノードがオーバープロビジョニングされていないことを前提としています。これは、各ポッドが一度に 25%を要求する必要があるという意味ではありません。たとえば、1 つ以上がアイドル状態である可能性があり、その時点で必要な CPU ははるかに少なくなります。

ただし、これは、各ポッドが一度に CPU リソースの 25 %を使用する可能性があることを意味します。ノード A でポッド 1、2、および 3 のそれぞれに CPU 要求があり、3 つのポッドのそれぞれがノードで 25%の CPU 要求があるとします。ノード A で他のポッドが CPU 要求を行っていないことを考慮すると、ノードの合計要求要求は 75%であり、ノードが CPU 負荷を処理できることを意味します。

他のクラスターの他のポッドもノード A で要求を行っていた場合、たとえば、別のポッドがノードで 30 の要求を持っていた場合、ノードは 105 でオーバープロビジョニングされます。

リミット

CPU とメモリの制限は両方とも、ポッドが使用できるノードのリソースの CPU とメモリの上限です。ポッド 1、2、および 3 の制限がそれぞれ 50 の CPU であり、すべてノード A であると仮定します。これは、ノード A の CPU 制限が 150 であることを意味します。一度に 100 %を超える CPU を提供できないため、3 つのポッドすべてが 50 %にバーストすると、結果として調整されます。ポッド 1 が 50 の CPU 制限にバーストし、他の 2 つが通常のオペレーションでそれぞれ 25 %のみを使用している場合、ポッド 1 は必要な 50 %を取得します。

メモリの場合、ポッド 1、2、および 3 がノード A でそれぞれ 50%のメモリである同様のシナリオを検討します。3 つすべてが 50%のしきい値までバーストしようとします。ポッド内のコンテナは、Linux カーネルによって killed になる場合があります。

ハードウェア使用率ダッシュボードを使用した潜在的なリソース問題のトラブルシューティング

プロビジョニングを要求した Postgres クラスターがあり、85 %完了した状態で 15 分以上フリーズしているように見えるとします。HM コンソールから、Web ブラウザーの URL の最後のビットを使用して、クラスターの名前に関連付けられた名前空間を見つけることができます。

この場合、名前空間 `p-b666674Q0h` を調べているとします。**ハードウェア使用率**ダッシュボードを使用して、最初にその名前空間に関連付けられているポッド-ここでも `p-b666674Q0h-1` 、`p-b666674Q0h-2` 、お

よび p-b666674Q0h-3 とします-を見つけ、これらのポッドのステータスを確認できます。1 つ以上のポッドが Pending 状態になっている可能性があります。たとえば、 p-b666674Q0h-1 および p-b666674Q0h-2 が実行されている間、 p-b666674Q0h-3 はまだ保留中の場合があります。次に、 **Postgres Pod Status** の下の Postgres ノードテーブルから p-b666674Q0h-3 が実行されているノードを特定できます。

p-b666674Q0h-3 がノード ip-10-0-73-251.eu-west-2.compute.internal で実行されていると仮定します。 **リソースの可用性**テーブルの ip-10-0-73-251.eu-west-2.compute.internal を確認すると、合計 8 コアのうち 0.5 コアが利用可能であり、合計 20.3 のうち 0.8Gi のメモリが利用可能なことがわかります。このことから、ノードには別のポッドのリソースがあまり残されていないと結論付けることができます。

Resource Commitments テーブルの ip-10-0-73-251.eu-west-2.compute.internal を確認すると、ノードに 129 の CPU 要求と 106 のメモリ要求があることに気づいたとしましょう。要求が 100%を超えていることを考慮すると、ノードがオーバープロビジョニングされていると結論付けることができます。これは、ポッド p-b666674Q0h-3 が Pending 状態でスタックしている理由を説明しています。より多くのリソースがなければ、この Postgres クラスターは完了できません。HM を使用して、プロビジョニングするクラスターを編集して、必要な CPU とメモリを削減できます。プロビジョニングプロセスは続行されます。クラスターを小さくすることがニーズを満たさない場合は、より多くのデータベースワーカーノードを Kubernetes クラスターに手動で割り当てるか、オートスケーラーの最大ノードを割り当てて、より多くのプロビジョニングを行うように認識する必要があります。

Loki と Fluent Bit

Fluent Bit サービスログで Loki を使用して、ポッドとそのステータスに関する詳細情報を見つけることができます。このビューを見つけるには

1. Grafana のナビゲーションパネルから、 **Explore** を選択します。
2. **Outline** の横にあるメニューから、 **Loki** を選択します。
3. **Label filters** メニューを使用して、左側に **service_name** を選択し、右側に **fluent-bit** を選択します。
4. その名前空間に関連付けられているポッドの Loki の Fluent Bit ログで検索クエリを実行します。さらに分析するクラスターに関連付けられた名前空間をコピーして、**Line contains** フィールドに貼り付け、 **Run Query** を選択します。

検索した名前空間を含むログが収集された時間を確認できます。その下には、実際のログが表示されます。

Pretty JSON をオンにして、検索された名前空間の各インスタンスをログ全体で強調表示して表示します。

検索しているポッドに関連付けられている Can't find target pod など、検索している名前空間に関連するエラーメッセージが全体で表示される場合があります。または、reason: FailedScheduling の下に、次のようなメッセージが表示される場合があります。

```
"0/43 nodes are available: 1 node(s) had untolerated taint (CriticalAddonsOnly: true),
 3 node(s) had untolerated taint (edbaiplatform.io/control-plane: true), 39␣
↪Insufficeint cpu, 4 Insuffcient memory. preemtion: 0/43 nodes are available: 39 No␣
↪preemption vitims found for upcoming pod, Preemption is not helpful for scheduling."
```

このメッセージは、ポッドが完了に失敗した正確な理由、つまり CPU とメモリの不足を示します。

分散高可用性クラスターのトレースバック

分散高可用性 DHA クラスターのトレースは、高可用性クラスターまたはシングルノードクラスターのトレースとは少し異なります。これは、DHA クラスターが 1 つ以上のデータグループに分割され、これらの各データグループにはグループごとに 2〜3 つのノードがあるためです。

たとえば、それぞれ 3 つのノードを持つ 2 つのデータグループ A および B を含む DHA クラスターがあるとします。クラスターのメインページの **Monitoring** タブに移動し、そのタブの **Monitoring** セクションに移動すると、2 番目の値をデフォルトの **クラスターレベル**から **ノードに変更できます。レベル** 。値を **ノードレベル**に変更した後、右側の最後のメニューを展開して、クラスターの 6 つのノードをそれぞれ表示できます。データグループ A の 1、2、および 3、およびデータグループ B の 1、2、および 3。

Kubernetes レベルに戻るには、Grafana の *Grafana* **ハードウェア使用率ダッシュボード** に移動します。

次に、 **ハードウェア使用率** ダッシュボードの左下近くにある **Postgres ノード** セクションを確認します。クラスターの **モニタリング**タブで、今表示した 6 つのノードは、<id-of-cluster/namespace>-a-1-1 、<<id-of-cluster/namespace>-a-2-1> などの形式でそれぞれのポッドで表されます。ポッドの指定は、ポッドが属するデータグループ A または B と、ポッドのどのノードである 1、2、または 3 を呼び出します。

ポッドを指定すると、実行しようとしているノードとポッドの状態の両方を確認できます。ノード ID を使用すると、ダッシュボードの ***Resource Availability*** および ***Resource Commitments*** セクションで、そのノードのリソースの可用性とリソースコミットメントの両方を表示できます。ノードに残っているコアの数、メモリ量などを確認できます。

シングルノードまたは HA クラスターと同様に、ここから一部のリソースの問題をトラブルシューティングできます。または、**Explore** に移動し、 *Loki* と *Fluent Bit* を使用して、関心のあるポッドのログレベルを調べることができます。

kubectl を使用して基になるクラスターリソースを確認する

Kubernetes へのアクセスを持つ管理者は、kubectl を使用して、Postgres クラスターの基になる Kubernetes ワーカーノードを分析できます。

まず、kubectl で HM に接続しているときに、最初に `kubectl get clusterwrappers` コマンドを使用します。このコマンドは、トレースバックしている Postgres クラスターの名前を、クラスターが属している名前空間と照合し、クラスターの経過期間と現在の状態に関する高レベルの情報を提供します。

```
kubectl get clusterwrappers
```

このコマンドは表示します。

- 名前空間ごとのクラスター。クラスターが関連付けられているプロジェクト ID を含む
- クラスターの名前空間と照合するために使用できるクラスター名
- クラスターの状態と年齢

```
NAME           PROJECTID              CLUSTERNAME        STATE     AGE
p-6gtrowr94d   prj_X9s5sGg5Zl2tDl1v   dbcluster-1        Healthy   2d2h
p-ha6mloybrf   prj_X9s5sGg5Zl2tDl1v   dbcluster-2        Healthy   5d23h
p-p93ou61mgw   prj_X9s5sGg5Zl2tDl1v   dbcluster-3        Healthy   6d
```

dbcluster1 をその Kubernetes リソースまでトレースしようとしているとします。たとえば、dbcluster1 が p-6gtrowr94d 名前空間にあることがわかりました。各 Postgres クラスターには、名前空間が 1 つだけあります。クラスターの状態は現在正常で、その経過期間は 2 日と 2 時間であることがわかります。

関心がある Postgres クラスターの名前空間の名前がわかっている場合は、それを使用して、その名前空間で実行されているポッドを表示できます。

```
kubectl get pods --namespace p-6gtrowr94d
```

すべてが正常に機能していると、出力として次のようなものが表示されます。

```
NAME             READY   STATUS    RESTARTS   AGE
p-6gtrowr94d-1   3/3     Running   0          2d2h
p-6gtrowr94d-2   3/3     Running   0          2d2h
p-6gtrowr94d-3   3/3     Running   0          2d2h
```

Postgres クラスター保留中

92%完了したプロビジョニング状態の HM を使用して作成した新しいクラスターがあるとします。新しいクラスターの名前は psrtest-1 で、Kubernetes レベルで何が起こっているのかを確認したいと考えています。最初に kubectl get clusterwrappers を使用して、HM の PG クラスターの名前をその名前空間と照合します。

```
kubectl get clusterwrappers
__OUTPUT__
NAME           PROJECTID              CLUSTERNAME        STATE     AGE
p-gime4xeamp   prj_X9s5sGg5Zl2tDl1v   psrtest-1          Pending   9m40s
```

クラスターのステータスが保留中であることは既にわかります。HM がまだプロビジョニング中であることを示したため、これは理にかなっています。クラスターの名前空間を取得したため、より詳しく確認し、ポッドのプロビジョニングの状況を確認することもできます。

```
kubectl get pods --namespace p-gime4xeamp
__OUTPUT__
AME                           READY   STATUS      RESTARTS   AGE
p-gime4xeamp-1                 0/3     Pending     0          7m17s
p-gime4xeamp-1-initdb-g6d9b    0/1     Completed   0          9m46s
```

1 つのポッドが正常に作成され、別のポッドがまだ作成中であることがわかります。現時点では、このクラスターが実際に停止しているようには見えません。まだプロビジョニング中です。

しばらくしても、ポッドの保留中ステータスがまだある場合は、リソースが使い果たされたと考える十分な理由があります。例

```
kubectl get pods --namespace p-gime4xeamp
__OUTPUT__
NAME                           READY   STATUS      RESTARTS   AGE
```

(次のページに続く)

(前のページからの続き)

```
p-gime4xeamp-1                0/3     Pending     0          28m
p-gime4xeamp-1-initdb-g6d9b   0/1     Completed   0          32m
```

リソースが不足したかどうかを確認するには、 `kubectl describe pod` を使用して、何が起こっているのかを正確に確認できます。

```
kubectl --namespace p-gime4xeamp describe pod p-gime4xamp-1
```

このコマンドは、ポッドに関する不要な多くの情報を結果ます。ただし、出力の最後では、何が問題になっているのかについての詳細を提供します。例、この場合、次のように表示されます。

```
Normal   NotTriggerScaleUp  4m15s (x181 over 34m)  cluster-autoscaler  pod didnt␣
↪trigger scale-up: 7 max node group size reached
```

これは、オートスケーラーが最大ノードグループサイズに到達したことを示しています。その結果、ポッドを完了できません。

警告

別のクラスターに十分なリソースがある場合でも、同時に作成しようとする数が多すぎると、システムはすべてを一度に作成しようとします。この場合、負荷が分散されるため、どれも完全にプロビジョニングできない可能性があります。

リソースの追加

ポッドを完了できないような最大ノードグループサイズに到達したことを学習した後、最大ノードグループサイズを変更して、より大きくなります。AWS EKS 展開では、AWS コンソールに移動して EKS に移動してこれを行います。

1. 作業している EKS クラスターを選択し、 **Compute** タブを選択します。
2. **ノードグループ** セクションで、PGNodeGroup を選択し、**Edit** を選択します。
3. ノードグループの **最大サイズ**の値を以前よりも大きい値に変更します。HM は、クラスターのプロビジョニングを続行します。

一度に多くのクラスターを作成しようとした場合、それに応じて **最大**サイズを拡張する必要がある場合があります。

Formulas for parameter values

***DB Configuration** tab* のパラメーターに特定の値を入力することに加えて、一部のパラメーターでは、値を計算するための式を指定できます。演算子`+ - / * > >= < <= == != && || ! ? : ( )` を使用して、*shared buffer example* などの三項フォーミュラで整数および実数タイプのパラメーターのフォーミュラを使用できます。`?` および`:` を使用します。必要に応じて、`( )` を使用して操作の順序を指定します。数式で使用される GUC も、整数または実数型である必要があります。ターゲット GUC が整数型で実数でない場合、すべての算術演算は整数の結果に丸められた 64 ビット浮動小数点値で実行されます。

Hybrid ManagerHM には、方程式の作成を支援する **擬似** *GUC* と呼ばれるものもあります。これらの読み取り専用 GUC は次のとおりです。

- `cpu_cores` — クラスターノードを実行している VM が提供するコアの数。
- `disk_iops` —プライマリデータボリュームをバッキングするディスクに割り当てられた IOPS の数。
- `disk_size` —プライマリデータボリュームをバッキングするメモリ量 KB 単位。
- `ram` —クラスターノードを実行している VM によって提供されるメモリ量 KB 単位。
- `replica_count` — クラスター内のノードの合計数。

フォーミュラは、クラスターに対して選択した仕様に相対して、または他の非擬似 GUC に相対して値を計算できます。クラスターのサイズを変更すると、HM は擬似 GUC の新しい値を使用して数式を再計算します。

操作の順序

* および/ は、演算子の優先順位が同じです。 + および- は、演算子の優先順位が同じです。 * および/ は、+ および- の前に評価されます。かっこを使用して、操作の追加の順序を指定します。

ユニット

GUC は、8 などの裸の数値か、GB などの単位を持つことができます。ユニットは、サイズや期間などのユニットの種類に分類されています。

ターゲット GUC のユニットの種類は、式で使用されるユニットの種類と一致する必要があります。たとえば、3 キロバイトと 2 秒を乗算する数式 `2s *3KB` は使用できませんが、スケールが異なる場合でも 2 つのサイズを乗算する数式 `2KB* 8GB` は使用できます。

ターゲット GUC の単位は、式で使用される単位と一致する必要があります。たとえば、2 秒に 4 日を掛けることはできますが、結果を KB 単位の GUC に割り当てることはできません。

ターゲット GUC に単位がある場合、方程式内の裸の数値はターゲット GUC のデフォルトの単位であるとみなされます。

再帰

再帰の深さは約 100 レベル以下をお勧めします。

例

`shared_buffers` パラメーターに特定の値を入力する代わりに、GUC 方程式を使用して、`shared_buffers` の値が RAM の特定のパーセンテージであるが、特定のサイズ以下であることを指定できます。例

```
((ram *0.25) > 8 GB) ? 8 GB : (ram* 0.25)
```

`wal_buffers` を `shared_buffers` の 3%、最大 16MB までにするとします。式は次のとおりです。

```
((shared_buffers *0.03) > 16MB) ? 16MB : (shared_buffers* 0.03)
```

フォーミュラは `shared_buffers` を参照し、この例では独自のフォーミュラがあります。この例は、GUC が他の GUC を参照できる方法を示しています。

一般的なエラーとエラー処理

不正な数式はエラーになります。次にいくつかの例を示します。

- 2 つの単位の数値の後に続く、例

```
work_mem = 10 GB kB
```

この構文では次のエラーが発生します。

```
unexpected token "kB" at end of input (line 1, col 7)
```

- 自分自身を参照する GUC を持つことにより、無限評価ループを作成します。例

```
effective_cache_size = effective_cache_size
```

この構文では次のエラーが発生します。

```
cycle in equation; GUC "effective_cache_size" already seen
```

- ユニットタイプの混合、例

```
work_mem = 10kB * 8s
```

この構文では次のエラーが発生します。

```
both size and time units used in calculation; cannot proceed
```

Managing database cluster access

ID プロバイダーを使用してデータベースクラスターアクセスハイブリッドマネージャーアクセスではなくを構成する場合、*Configuring LDAP* を参照してください。

LDAP authentication for database cluster access

データベースクラスター認証用に LDAP を構成できます。

- 適用対象ハイブリッドマネージャーバージョン 1.2 以降

前提条件

- API を使用して LDAP 構成を使用してクラスターを展開する前に、環境に LDAP サーバーが存在する必要があります。
- ユーザーが LDAP を介して認証できる前に、ユーザーは PostgreSQL に存在する必要があります。

API を介して LDAP を構成する

LDAP サーバーをセットアップした後、API を使用してクラスターを展開し、LDAP 認証を構成します。

LDAP を使用する新しいクラスターの作成

REST API を使用する前に、有効なキーがあること、およびクラスター CRUD 操作を実行するための適切なロールが割り当てられていることを確認します。

新しいクラスターの場合、クラスターの作成中に curl を使用して LDAP サーバーを構成します。要求内の${URL}、KEY、URL、project_id、およびその他のパラメーターを更新します。

```
curl -H "x-access-key: ${KEY}" -H "Content-Type: application/json" --insecure -s
↪"https://.${URL} edbaiplatform.enterprisedb.network/api/v1/projects/prj_
↪pJ9Ug36492SQhdEQ/clusters" -d {"projectId": "prj_pJ9Ug36492SQhdEQ",
 "psr": {
   "clusterName": "ldap-cluster",
   "password": "111111111111",
   "locationId": "managed-devspatcher",
   "clusterData": {
     "instances": 1,
     "resourceRequest": {
       "requests": {
         "memory": "2Gi",
         "cpu": "1"
       }
     },
     "storageConfiguration": {
       "primaryStorage": {
         "size": "4"
       }
     },
     "image": {
       "url": "<pg_image_url>",
       "digest": "<pg_image_sha>"
     },
     "ldapConfig": {
       "enabled": true,
       "config": {
         "hostname": "ldap-service.default.svc.cluster.local",
         "scheme": "ldap",
         "bindAuth": {
           "prefix": "uid=",
           "suffix": ",dc=example,dc=org"
         }
       }
     }
   }
 }}
```

LDAP を使用するように既存のクラスターを変更する

既存のクラスターの場合、 PATCH を使用して LDAP 認証用に構成します。

```
curl -X PATCH -H "x-access-key: ${KEY}" -H "Content-Type: application/json" --
↪insecure -s "https://portal.eks-05220612-main.edbaiplatform.enterprisedb.network/
↪api/v1/projects/prj_sXNNRhwj1YddHslR/clusters/p-lfdghq5gxe" -d {"projectId": "prj_
↪sXNNRhwj1YddHslR","clusterId": "p-lfdghq5gxe",
"psr": {
  "ldapConfig": {
   "enabled": true,
   "config": {
    "hostname": "ldap-service.default.svc.cluster.local",
    "scheme": "ldap",
    "bindAuth": {
     "prefix": "uid=",
     "suffix": ",dc=example,dc=org"
    }
   }
  }
 }
}
```

検証

pg_hba.conf ファイルが認証用の LDAP エントリーで更新されたことを確認します。

```
#
#  LDAP CONFIGURATION (optional)

#
host all all 0.0.0.0/0 ldap ldapserver="ldap-server.default.svc.cluster.local"␣
↪ldapscheme="ldap" ldapprefix="uid=" ldapsuffix=",dc=example,dc=org"
```

バインドアンドサーチモード構成

search+bind モードを使用する場合、次のパラメーターが必要です。

- server
- prefix
- suffix

search+bind モードを使用する場合、次のパラメーターはオプショナル。

- searchFilter
- searchAttribute searchAttribute が指定されない場合、デフォルト値は uid に設定されます。
- scheme LDAP スキーム
- port

この例は、検索+バインドモードの構成を示しています。

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

バインドモード構成

バインドモードを使用する場合、次のパラメーターが必要です。

- server
- baseDN
- binDN
- bindPassword ldap パスワードを含むシークレット

バインドモードを使用する場合、次のパラメーターはオプショナル。

- scheme LDAP スキーム
- port

この例は、バインドモードの構成を示しています。

```
postgresql:
  ldap:
    server: "openldap.default.svc.cluster.local"
    bindAuth:
      prefix: "uid="
      suffix: ",dc=example,dc=org"
```

LDAP over TLS

TLS を使用して、PostgreSQL サーバーと LDAP サーバー間のトラフィックを暗号化できます。TLS 証明書を使用するように LDAP サーバーを構成できます。

SSL が構成されていない限り、PostgreSQL サーバーと PostgreSQL クライアント間の接続は暗号化されないままです。

詳細情報

- LDAP Authentication
-

LDAP Configuration

Cluster view

クラスタービューは、クラスターの構成、状態、モニタリング、クエリ診断、バックアップ、ログ、その他のプロパティを含むクラスターの詳細な概要を提供します。Cluster ビューを開くには、 Clusters テーブルからクラスターを選択します。

[クラスター] ビューには、クラスターの名前、作成時刻、クラスターの種類、使用される Postgres ディストリビューション、および Postgres ディストリビューションのバージョンが表示されます。作成時間は、クラスターが現在時刻と相対して作成された時間を示します。クラスター名前を選択して、クラスターの接続情報ページを開きます。

タグは、管理を簡単にするためにクラスターをグループ化するために使用されます。クラスタービューでは、タグがリンクのリストとして表示されます。タグを選択して、そのタグを持つすべてのリソースを表示する詳細ビューを開きます。

タグを追加または削除するには、 + を選択します。ダイアログボックスで、タグを追加および作成し、クラスターからタグを削除できます。

クラスタービューのタブは次のとおりです。

- **アーキテクチャの概要** — クラスターの簡単な概要。
- **接続オプション** — クラスターの接続情報。
- **オペレーションモニタリング** — クラスターのモニタリング情報。
- *Diagnostics [***クエリーの診断と推奨事項***] タブ* — クラスターの診断情報を照会します。
- *HM* **バックアップと** *DR* — クラスターのバックアップ情報。
- **メトリックとログへのアクセス** — クラスターのログ。
- *Cluster Properties* — クラスターのプロパティ。

Cluster overview

[**Cluster Overview**] タブには、クラスターの接続文字列、アクティブなアラート、および最近のクラスターアクティビティの概要が表示されます。

クラスターが単一ノードまたはプライマリ/スタンバイの高可用性クラスターの場合、概要には最初に *Quick Connect* セクションが表示されます。

クラスターが分散高可用性クラスターの場合、概要には最初に *Creating data groups* セクションが表示されます。

接続文字列の取得

Quick Connect セクションは、クラスター用に構成された接続文字列を含む完全な psql コマンドラインと、`edb_admin` データベースに接続するための edb_admin ロールを持つユーザーを提供します。このコマンドをコピーしてターミナルに貼り付けて、データベースに接続できます。データベースに接続するには、edb_admin ユーザーのパスワードが必要です。

データグループ

Data Groups の下で、クラスター内のデータグループと監視グループの概要を表示します。各グループは、ステータスバッジ、データグループの名前、および**データグループ** または **Witness Group** ラベルでリストされます。

ステータスバッジは、データグループのステータスを示します。白いチェックマークの付いた緑の円は、データグループが正常であることを示します。表示されるその他の情報には、作成からの時間、Postgres サーバータイプとバージョン、データグループ内のノード数が含まれます。

- データグループのノード設定とデータベース構成を編集するには、データグループ名の横にある **Edit Data Group** を選択します。
- データグループが複数ある場合、データグループを削除できます。すべてのデータグループを削除することはできません。省略記号メニューから **Delete Data Group** を選択します。
- 削除したデータグループを復元するには、省略記号メニューから **Restore Data Group** を選択します。

データグループの **Quick Connect** の下で、読み取り-書き込みおよび読み取り専用接続の psql コマンドライン接続文字列を取得できます。

Read-write または **Read-only** を選択して、データグループと `bdrdb` データベースに接続する edb_admin ロールを持つユーザー用に構成された接続文字列を含む psql コマンドラインを表示します。データベースに接続するには、このコマンドをコピーしてターミナルに貼り付けます。これを行うには、edb_admin ユーザーのパスワードが必要です。

> **注釈**
>
> **読み取り専用**接続文字列は、Hybrid Manager 1.1 では使用できません。

クラスターアーキテクチャ図の表示を切り替えるには、 **Cluster Architecture Diagram** 見出しを選択します。この図は、データグループ、監視グループ、プロキシを含むクラスターのアーキテクチャ、およびそれらの間の相互接続を示しています。

アクティブなアラート

現在アクティブなアラートの数は、高重大度、中重大度、および低重大度の 3 つのカテゴリにリストされます。表示されている数を選択すると、 ***Monitoring*** タブが開き、アラートの詳細を表示できます。

クラスターアクティビティログ

クラスターに記録された最近のアクティビティには、作成、構成の更新、およびクラスターのライフサイクル内のその他のメジャーイベントが含まれます。アクティブな要素はここには表示されません。イベントの詳細については、***Logs*** タブを選択してください。

Cluster connections

Cluster Connect タブを使用して、クラスターの接続情報と、クラスター内の `edb_admin` データベースに接続する edb_admin ロールを持つユーザーを取得します。

クラスターが分散高可用性クラスターの場合、接続情報はクラスター内のデータグループごとに提供されます。

接続情報

次の接続情報を表示できます。

- **読み取り/書き込み URI** 読み取り/書き込みアクセスでクラスターに接続するための接続文字列。これは、**Quick Connect** セクションの psql コマンドラインと同じです。
- **Dbname** 接続するデータベースの名前。これは常に `edb_admin` です。
- **読み取り/書き込みホスト**読み取り/書き込みアクセスのためのクラスターのホスト名または IP アドレス。
- **Port** 接続するポート。これは常に `5432` です。
- **User** 接続するユーザー。これは常に edb_admin です。

これらの各フィールドをコピーして、データベースへの接続試行を開始するために使用できます。

`.pgpass` blob

`.pgpass` blob をコピーして `.pgpass` ファイルに貼り付けることができるため、パスワードを入力せずにクラスターに接続できます。これは、スクリプトやアプリケーションなど、クラスターへの接続を自動化する場合に役立ちます。プレースホルダーを実際のパスワードに置き換えます。不正なアクセスを防ぐには、`.pgpass` ファイルに正しい権限が設定されていることを確認します。権限を `0600` ユーザーの読み取りおよび書き込みのみに設定します。

`.pgpass` ファイルの詳細については、PostgreSQL documentation を参照してください。

`.pg_service.conf` blob

`.pg_service.conf` blob をコピーして `.pg_service.conf` ファイルに貼り付けることができるため、**サービス名**で参照してクラスターに接続できます。これにより、クラスターへの接続を覚えやすくなります。`.pgpass` blob と組み合わせて、サービス名を指定して接続できます。`.pg_service.conf` ファイルの詳細については、

PostgreSQL documentation を参照してください。

`pgpass` および `.pg_service.conf` ファイルは、libpq を使用する Postgres クライアントアプリケーションでサポートされています。

Cluster monitoring

[**Monitoring**] タブは、ノードとクラスターの状態の詳細なビューを提供します。

モニタリング

クラスターの状態の概要については、**Monitoring** セクションを確認してください。テーブルの上部にあるモニタリングコントロールを使用して、テーブルに表示される内容を制御します。

表示されるデータの開始時期を指定できます。デフォルトは 15 分で、これは、過去 15 分間のデータを表示することを意味します。15 日〜5 分の値を指定できます。または、日付と時刻ピッカーを使用して、監視データをサンプリングするカスタム時間範囲を選択します。クラスターのデータが利用可能な場合にのみ時間範囲を選択できます。最初のメニューから時間を選択すると、指定した日付と時刻がクリアされます。

クラスター内のすべてのノードの集計としてデータを表示するかどうか**クラスタレベル** 、または各ノードとして表示するかどうか**ノードレベル**を指定することもできます。デフォルトでは、データをクラスターレベルで表示することです。

クラスターレベルでは、各リングチャートは、クラスター内のすべてのノードのデータの平均を示す単一のリングであり、グラフは同様の集計にもとづいています。

ノードレベルでは、各チャートは同心リングチャートです。各リングは、クラスター内のノードを表します。グラフはノードごとに個別に表示されます。

アクティブなアラート

アクティブなアラートの概要には、アクティブなアラートの数が高重大度、中重大度、低重大度の 3 つのカテゴリで表示されます。

アラートの検索可能なテーブルビューでは、検索/フィルターバーを使用して次のことができます。

- テキスト検索用語でフィルタリングします。 **Search** ボックスに値を入力します。
- 時間範囲でフィルタリングします。 **From** および **To** 日付/時刻ピッカーを選択します。
- 重大度によるフィルタリング。 **Filter > Severity** を選択し、1 つ以上の使用可能な重大度を選択します。
- 開始時間、重大度、またはアラートで昇順または降順で並べ替えます。
- 稲妻アイコンを選択して、自動更新を選択します。
- 表示されている列を選択します。歯車を選択し、メニューからテーブルに表示する列を選択します。
- 現在表示されているアラートを CSV 形式でダウンロードします。ダウンロードアイコンを選択します。

ホスト

グラフに表示されるオペレーティングシステム統計の概要については、**Host** セクションを確認します。

- **Memory** Postgres プライマリノードのメモリの平均メモリ使用率。
- **CPU** Postgres プライマリノードの平均 CPU 割合。
- **Storage** Postgres プライマリノードに使用される合計ストレージ。

- **Disk IOPS** 一定期間における 1 秒ごとのディスクの読み取り、書き込み、および合計操作の合計数。
- **ディスクスループット** Postgres ノードの 1 秒ごとにディスクとの間で転送されるデータの合計量。
- **ネットワークアクティビティ** Postgres ノードの期間、1 秒ごとにネットワークカードとの間で転送されるデータの合計量。

接続

- **Connections** クライアントアプリケーションと Postgres データベース間のタイプ別の現在の接続数。
- **待機タイプ別の平均アクティブセッション**プライマリ Postgres ノードが各待機イベントタイプに費やした時間は、平均アクティブセッション AAS を使用して計算されます。
- **ブロックされたバックエンドの数**クラスター内のすべての Postgres ノードにわたってロックで待機しているバックエンドの合計数。

トランザクション

- **Tuples In** Postgres ノードの 1 秒ごとに挿入、更新、および削除されたタプルの合計数。
- **Tuples Out** Postgres ノードの 1 秒ごとに取得および返されたタプルの合計数。
- **トランザクションレート** Postgres ノードの 1 秒ごとのコミット/ロールバック/合計トランザクションの合計数
- **バッファキャッシュヒット率**クラスター内のすべての Postgres ノードの平均バッファキャッシュヒット率。
- **最長実行トランザクション** Postgres ノードの 1 秒ごとの最長実行トランザクションの合計数。

クエリー

- **Query Rate** Postgres ノードの 1 秒ごとのクエリーの合計数。
- **Query Latency** クラスター内のすべての Postgres ノードのミリ秒単位の平均クエリレイテンシー。

ストレージ

- **データベースサイズ** 折れ線グラフプライマリ Postgres ノード全体のデータベースサイズの合計。
- **ディスク使用量**プライマリ Postgres ノードのディスク使用量のパーセンテージ。
- **WAL Size** プライマリ Postgres ノード全体の WAL ディレクトリの合計サイズ。
- **WAL 使用量** プライマリ Postgres ノードの WAL 使用率。
- **Live/Dead Tuples** Postgres ノードの Live/Dead タプルの合計数。
- **インデックス、テーブル、および一時的な推定サイズ** Postgres ノードのインデックス、テーブル、および一時の推定サイズ。

内部

- **最後の自動バキュームからの時間** Postgres ノードの秒単位の最後の自動バキュームからの時間。
- **Autovacuum Stats** Postgres ノードの 1 秒ごとの自動バキューム操作の合計数。
- **最後のチェックポイントからの時間** Postgres ノードの秒単位の最後のチェックポイントからの時間。
- **Checkpoint Stats** Postgres ノードの 1 秒ごとのチェックポイントの合計数。
- **最後の成功した WAL アーカイブからの時間** Postgres ノードの秒単位の最後の成功した WAL アーカイブからの時間。
- **WAL Archiving Stats** Postgres ノードの 1 秒あたりの WAL アーカイブ操作の合計数。

Cluster Query Diagnostics

クエリ診断

[Query Diagnostics] タブは、クラスターで実行されているワークロードに関する洞察を提供します。実行されたクエリに関する詳細な統計を収集するため、ユーザーは問題のある領域をすばやく特定できます。

[Query Diagnostics] タブは、4 つのグラフとクエリの表で構成され、クエリの実行に関するパフォーマンスの問題を診断するための役立つ統計を提供します。

チャート

チャートは次のとおりです。

- 「クエリレイテンシー」、クエリレイテンシーのパーセンタイル 50、90、および 99 を含む折れ線グラフ、
- 「実行されたクエリーの数」、実行されたクエリーの生の数を含む折れ線グラフ、
- クエリレイテンシーヒストグラム、クエリレイテンシー分布を含む棒グラフ、
- 「アクティブなセッションの数」アクティブなセッションの数と、I/O、CPU、IPC、およびロックの関連する待機状態を含む積み上げ面グラフ。

フィルタを使用すると、ユーザーは次の方法でデータを制限できます。

- 時間間隔、
- 開始日と終了日、
- 特定のノード。

クエリテーブル

グラフの下のテーブルには、収集されたすべてのクエリーとその情報と統計がリストされています。

- “Query ID” クエリの ID、
- “Query”、切り詰められたクエリテキスト、
- 「# of Executions」このクエリの実行数、

- “50th Percentile” クエリレイテンシーの 50 パーセンタイル
- 「90th Percentile」、クエリレイテンシーの 90 パーセンタイル、
- 「99th Percentile」 クエリレイテンシーの 99 パーセンタイル
- 「CPU %」 CPU での作業に費やされた時間の割合、
- 「Wait %」待機して費やされた時間つまりアクティブですが、CPU ではない時間の割合、
- “最小実行時間”、
- “最大実行時間”、
- “CPU AAS”、クエリによって CPU で費やされた平均アクティブセッション
- 「Wait AAS」、クエリーによって待機状態で費やされた平均アクティブセッション、
- “Total AAS”、クエリの実行に費やされた平均アクティブセッション、
- 「Count」、このクエリを少なくとも 1 回実行したユーザー、データベース、およびノードの数。

歯車アイコンを使用すると、列の表示/非表示が可能です。

フィルタを使用すると、ユーザーは次の方法でデータを制限できます。

- データベース,
- ユーザー。

右側のドロップダウンメニューを使用すると、テーブルの行を列ごとに昇順または降順で並べ替えることができます。

左側の検索バーを使用して、クエリテキストの部分文字列を検索できます。

クエリ診断について

クエリーとは何ですか?

クエリ診断を行うために、述語が削除された後、一意のクエリテキストとしてクエリを定義しました正規化とも呼ばれます。たとえば、`SELECT id FROM users WHERE name = "Bob"` は `SELECT id FROM users WHERE name = "Alice"` と同じクエリーです。これは、クエリーが別のデータベースで別のバージョンの `users` テーブルに対して実行される場合にも当てはまります。

平均アクティブセッション AAS とはどういう意味ですか?

平均アクティブセッションは、すべてのセッションにわたって特定の状態で費やされた時間を示すメトリックです。これは、すべてのセッションを合計して、目的の状態で費やされた各セッションの時間の割合として計算されます。

たとえば、単一のセッションが同じクエリを 1 時間継続して実行した場合、そのクエリはその時間に 1AAS を占有すると言われます。クエリの実行に 30 分のみを費やした場合、それは 0.5 AAS になります。

代わりに 10 のセッションがそれぞれ 30 分ずつクエリを実行している場合、それは 5 AAS になります。

平均アクティブセッション数は、「昨日のトップ 10 クエリーは何でしたか?」という質問に答えるために役立つメトリックです。定義では、最も高い AAS を持つ 10 のクエリーは、一日全体、すべてのセッションに

わたって、最も時間を占めたクエリーとなります。AAS の場合、クエリが 1 日に 100 万回実行されたか、1 回だけ実行されたかは関係ありません。合計実行時間を測定します。

データはどこから来たのですか?

クエリ診断用のデータは、2 つの拡張機能から取得されます。1 つだけ使用可能な場合、機能が制限されます。どちらも使用できない場合、クエリ診断は無効になります。

- EDB Wait States は、セッション状態からサンプリングデータを収集します。これは、AAS および関連メトリックを設定するために使用されます。
- EDB Stat Monitor は、クエリ実行レベルでデータを収集します。デフォルトでは、すべてのクエリ実行が記録されます。これは、実行数を入力し、レイテンシーメトリックを照会するために使用されます。

クエリ診断には、実行されたすべてのクエリが表示されますか?

はい、EDB Stat Monitor が有効になり、`edbsm_sample_rate` がデフォルトの 1 に設定されている場合、クエリ診断には、`SET` などのユーティリティクエリや PGPLSQL プロシージャ内で実行されるクエリを除き、実行されたすべてのクエリが含まれます。

EDB Stat Monitor が有効になっていない場合、表示されるクエリは、EDB 待機状態によって観察されたクエリのみです。これには、大量の AAS に寄与するクエリが含まれる可能性がありますが、これは保証できません。

Cluster recommendations

おすすめ

[**Recommendations**] タブには、Postgres クラスターのパフォーマンスの向上に役立つスコアカードと推奨事項のリストが表示されます。

スコアカードと推奨事項は直接関係しています。クラスターは、推奨事項が表示される場合にのみ Excellent より低いスコアとなります。推奨事項を適用または却下すると、スコアに Excellent と示されます。

スコアカード

スコアカードでは、スコアは各評価カテゴリの特定のクラスターに割り当てられます。

- インデックス
- 統計
- 構成
- セキュリティ
- ワークロード

> **注釈**
>
> 現在、最初のインデックスと統計のみが使用できます。

通常、グレードは A〜F スケールで割り当てられます。

Indexes カテゴリの場合、可能なグレードは次のとおりです。

- Excellent (A) インデックスを調整するための推奨事項が生成されなかった場合
- 生成された推奨事項の数に応じて、良い B、Fair、C、および悪い D

これらのスコアは、PSR の場合はプライマリノード、PGD / AHA の場合は書き込みリーダーのみを考慮して計算されます。さらに、スコアは、最もパフォーマンスの悪い PG クラスター内のデータベースに関連しており、そのようなクラスターで作成されたすべてのデータベースの平均ではありません。

統計カテゴリの場合、可能なグレードは次のとおりです。

- Excellent (A) 統計を追加するための推奨事項が生成されなかった場合
- 良好 B 少なくとも 1 つの推奨事項が生成された場合

> **注釈**
>
> - Fail F は現在使用されていません。これは、即時のアクションが必要な異常なクラスター状態のために予約されています。 - 不明 U は、クラスターを評価するために必要な情報が利用できないことを意味します。

カテゴリを選択すると、表示されるテキストは、スコアが計算された方法に関するコンテキストを提供することに加えて、推奨事項の関連性を理解するのに役立ちます。

レーダーチャートには、すべてのカテゴリのクラスターのグローバルスコアが表示されます。

おすすめリスト

スコアカードの下には、スコアカードで選択したカテゴリに応じて、推奨事項のリストが表示されます。

- ステータス 推奨事項のステータス New、Ignored、Applied 。
- データベース名。
- ノード ID。
- サイズの増加インデックスの場合 推定インデックスサイズ。
- Absolute Benefit インデックスの場合このインデックスを適用することにより生み出される絶対的なコスト削減。
- 合計コストインデックスの場合このクエリを実行するための実行時間とプラン時間。
- コスト削減インデックスの場合推定コスト削減パーセンテージは、(abs_benefit/total_cost) * 100 として計算され、データベースで実行されたすべてのワークロードに相対しています。
- 重み統計の場合 統計を適用することの潜在的な利点の大まかな推定。
- 推奨時間 この推奨事項の作成時間。
- クエリ インデックスまたは統計を作成するクエリのテキスト。

列を表示/非表示にするには、歯車アイコンを使用します。

各行の CTA ボタンでは、次のことができます。

- 推奨事項を完了済みとしてマークします
- 推奨を無視する
- 推奨事項のテキストをコピーします

!!!注 Web インターフェイスから推奨事項を適用することはできません。ターミナルまたは管理 UI でそれらを手動でコピーして実行する必要があります。

フィルタを使用して、データベース、ノード、またはステータスごとにデータを制限します。

推奨事項を理解する

推奨事項を適用/無視するものとしてマークします

applied オプションを使用して、推奨される変更を適用したことをハイブリッドマネージャー HM に伝えます。推奨事項を適用済みとしてマークすると、リストから消えます。ページが次にリロードされると、スコアが増加して変更を反映します。同じ推奨事項が再度表示される場合、これは変更が適用されないか、元に戻されたことを意味します。またスコアが下がってしまいます。

ignored オプションを使用して、推奨される変更を確認し、適用しないことにしたことを HM に伝えます。おすすめを無視するとしてマークを付けると、リストから消えます。次にページがリロードされると、スコアが増加します。同じ推奨事項を再度受け取った場合、それは無視され、再度表示されません。

このデータはどこから来たのですか?

インデックスと統計の推奨事項は、クエリアドバイザー拡張機能から取得されます。この拡張機能が監視対象データベースで有効になっていることを確認します。

将来のリリースでは、履歴メトリックやクエリ診断を含む他のデータソースからレコメンデーションが生成されます。

Backup Management

ハイブリッドマネージャーの特定のクラスターの下の [Backups] タブは、その PostgreSQL クラスターのバックアップを管理するための集中ビューとコントロールパネルを提供します。このページは、ユーザーがバックアップライフサイクルを効率的かつ安全に追跡、検証、および管理するのに役立ちます。

目的

[Backups] タブは、選択したクラスターに関連付けられているすべてのバックアップアクティビティを提供するように設計されています。過去のバックアップの監視と新しいバックアップの開始の両方をサポートしています。運用およびサイト信頼性エンジニア SRE チームがデータ保護を保証し、必要に応じて復旧プロセスを促進するために特に役立ちます。

主な機能

- **バックアップステータスの監視**

バックアップが正常に完了したか失敗したか、タイムスタンプと期間を簡単に確認できます。

- **手動バックアップの開始** [Create Backup] ボタンを使用して、クラスターの即時バックアップをトリガーします。

- **バックアップメタデータの確認**

開始時間と停止時間、バックアップ方法ボリュームスナップショットなど、使用されるストレージ場所などの詳細な洞察を取得します。

- **バックアップからの復元**

復元アイコン⌧を使用して、特定のバックアップポイントからリカバリーを開始します。

- **バックアップ ID をコピーします**

コピーアイコン⌧を使用して、オートメーション、サポートケース、またはログで参照するためにバックアップ識別子をコピーします。

- **バックアップの削除**

削除アイコン⌧を使用して、不要になったバックアップを手動で削除します。

[backups] タブでは、次の列を含むバックアップジョブの詳細情報を提供します。

Column	Description
Status	バックアップが正常に完了した Completed、失敗、またはまだ実行中であることを示します。
Name	バックアップジョブのシステム生成の名前。
Start Time / Stop Time	バックアップが開始および終了した時刻を示すタイムスタンプ。
Duration	バックアップが完了するまでにかかった時間。
Location	バックアップの保存先たとえば managed-devpatcher。
Method	バックアップの取得に使用されるテクノロジーたとえば、 VolumeSnapshot 。

ユーザビリティ

このタブは、次の場合に使用します。

- 自動またはスケジュールされたバックアップの正常な完了の確認
- メジャーアップグレードまたはメンテナンスタスクの前にバックアップを実行する
- データベースを以前の既知の良好な状態に復元する
- 履歴バックアップアクティビティの監査

Cluster Logs

ハイブリッドマネージャーで選択したクラスターの Logs タブは、クラスターの内部操作、システムイベント、および拡張レベルのアクティビティに関するリアルタイムおよび履歴の洞察を提供します。これは、クラスター内の個々のポッドの状態をデバッグ、診断、および監視するための重要なツールです。

目的

[Logs] タブは、ユーザーがクラスター内の各ポッドのイベント、警告、およびシステムアクティビティをトレースできるようにする集中ロギングインターフェイスとして機能します。データベース管理者や Kubernetes オペレーターが、誤った構成の拡張機能、ハートビート障害、欠落コンポーネントなどの運用上の問題を迅速に特定するのに役立ちます。

主な機能

- **ポッドレベルのログの表示**ポッドごとにログを表示するため、クラスターの特定の部分に影響を与える問題を簡単に分離できます。
- **時間範囲によるフィルタリング**

日付セレクターを使用して、特定の時間フレーム内のログをフィルタリングし、インシデント調査中の焦点を向上させます。

- **イベントの検索とフィルタリング**イベントタイプまたはキーワードでフィルタリングすることにより、関連するログエントリを簡単に見つけます。
- **ログレベルインジケーター**

Info や Warn などのラベルを介して、各イベントの重大度をすばやく評価します。

- **エラーの詳細な説明**

ログには、欠落している拡張機能や失敗したクエリなどの詳細なエラーメッセージが含まれており、根本原因の分析に役立ちます。

[logs] タブには、次のクラウドを含むログの詳細情報が表示されます。

Column	Description
Timestamp	ログイベントが発生した時 UTC を示します。
Pod	イベントが発生したポッド。
Event	拡張エラーやハートビートチェックなどのイベントの簡単な説明。イベントの情報、警告などの重大度ラベルのレベルも提供します。

ユーザビリティ

- モニタリング拡張機能 query_advisor、edb_stat_monitor などがアクティブであり、正しく構成されているかどうかを確認します。
- 欠落または非機能のデータベース拡張機能に関連するエラーの調査。
- ポッドのハートビートなどの定期的なシステムレベルのイベントの追跡。

- クラスター動作の最近の変更または異常の監査。

Cluster Properties

[**プロパティ**] タブは、選択した PostgreSQL クラスターの構成設定とリソース割り当ての包括的な概要を提供します。このタブは、スケーリング、編集、トラブルシューティングなどのクラスター操作を実行する前に、技術仕様を確認したり、セットアップパラメーターを検証したりする必要があるユーザーに重要です。

目的

このタブは、クラスターに関連するすべての静的プロパティの単一参照ポイントとして機能します。コンピューティングリソース、ストレージ構成、ネットワークアクセス、PostgreSQL バージョンなど。データベース管理者、サポートエンジニア、および開発者に役立ちます。

- クラスターの容量と制限を理解する
- 環境がワークロード要件を満たしていることを確認します
- コンプライアンスと監査チェックの促進
- 構成の不一致または展開エラーを特定する

[プロパティ] タブでは、次のプロパティを使用してクラスター構成設定に関する詳細情報を提供します。

Property	Description
Cluster Type	クラスターがシングルノードまたは高可用性 HA であるかどうかを示します。
Instances	クラスターにデプロイされた PostgreSQL インスタンスの数。
PgType	データベースタイプ PostgreSQL などを指定します。
PostgreSQL Version	現在クラスターにインストールされている PostgreSQL のバージョン。
Image	クラスターのプロビジョニングに使用されるコンテナまたは VM イメージタグ。
CPU Cores	クラスターに割り当てられた仮想 CPU コアの合計数。
Memory	クラスターに割り当てられた合計メモリ。
Primary Storage Class	永続ボリューム要求によって使用される基になるストレージクラスたとえば *gp3* 。
Primary Storage Size	プライマリインスタンスに接続されているストレージボリュームのサイズ。
Network Access	クラスターがパブリックまたはプライベートにアクセスできるかどうかを示します。
Connections	接続モード例 *Read/write* について説明します。
Retention Period	バックアップとログが保持される期間たとえば、30 日。

追加機能

- **Edit Cluster** 「Edit Cluster」リンクをクリックすると、ユーザーはリソースの割り当てやネットワーク構成などの編集可能なクラスタープロパティを変更できます。一部のプロパティでは、クラスターの再起動または再プロビジョニングが必要になる場合があります。
- **クイックアクション**スケーリングや削除など、クラスターに関連する状況依存のオプションにアクセスします。

Upgrading on Hybrid Manager

アップグレードは、データベースのメジャーアップグレードでもマイナーアップグレードでも、データベース管理の重要な部分です。

- *Postgres major version upgrades*
- *Postgres minor version upgrades on Hybrid Manager*

Postgres minor version upgrades on Hybrid Manager

マイナーバージョンのアップグレードは、クラスタータイプ単一ノード、プライマリ/スタンバイ、分散および Postgres データベースタイプ PostgreSQL コミュニティ、EDB Postgres Extended Server、 EDB Postgres Advanced Server に関係なく、同じ手順を使用します。

マイナーバージョンリリースには、バグ修正、セキュリティパッチ、および下位互換性の向上が含まれます。これらは通常、少なくとも 3 か月に 1 回発行されます。

PG のマイナーリリースは、新機能や重大な変更を導入しないため、リリースされたらすぐに採用することをお勧めします。PG クラスターが複数のマイナーバージョンを実行している場合でも、利用可能な最新のマイナーバージョンを採用することをお勧めします。

ユーザーは、ハイブリッドマネージャー HM を使用して、新しいバージョンを利用できる時期を完全に制御できます。検証および公開されると、新しいバージョンのイメージが HM のイメージライブラリで利用できるようになり、既存のクラスターでアップグレードに利用できます。

> **注釈**
>
> PG のマイナーバージョンのアップグレードには、データの再起動が必要です。最小限のダウンタイムが予想されます。

マイナーアップグレードは頻繁に行われ、リスクが低いのが理想的ですが、推奨されるベストプラクティスは、アップグレードを続行する前に、常に最新のバックアップとリカバリプランを実施することです。

開始するための前提条件は、HM が起動し、正常な状態で作成されたクラスターがあることです。

クラスターが正常な状態にあることを確認するには、クラスターのメインページから **Monitoring** タブを選択します。次に、クラスターとデータベースメトリックを確認し、すべてが動作していること、すべてのメトリックが定義された許容可能なパラメーターの範囲内にあることを確認します。

HM で作成した既存のインスタンス/クラスターの PG マイナーバージョンをアップグレードする

1. 最初に、クラスターのメインページから **Quick Action>Upgrade Cluster** を選択し、 **Minor Upgrade** を選択します。
2. [**アップグレード クラスター : マイナー**] ダイアログウィンドウが開きます。
3. 優先する新しいマイナーバージョンのイメージを選択します。
4. アップグレードパスを確認し、[**Continue**] ボタンを選択して確認します。
5. ダイアログウィンドウには、アップグレードで予想される特定のパッケージの変更が表示されます。これらを確認し、すべてが正しい場合は**アップグレードクラスター** を選択します。

6. アップグレードは自動的に適用されます。 次に、 **Monitoring** 、および **Logs** タブを使用して、プロセスをモニタしたり、予期しない動作のトラブルシューティングを行ったりできます。

> **注釈**
>
> まれに、特定の製品リリースのマイナーアップグレードが完了した後、追加の手順が推奨される場合があります。 問題の製品のリリースノートを参照して、あなたの場合にこれが必要かどうかを確認します。

ロールバックの考慮事項

このアプローチでは使用可能な自動ロールバックはなく、ダウングレードの実行はお勧めできません。マイナーアップグレードを元に戻すには、優先マイナーバージョンを実行し、アプリケーションワークロードを新しいクラスターにスイッチする新しいクラスターで最新のバックアップから復元します。次に、PG マイナーアップグレードされたクラスターを削除するか、さらにトラブルシューティングを行うことができます。

Postgres major version upgrades

メジャーバージョンのアップグレードにより、PostgreSQL の最新バージョンで導入された新機能、機能強化、およびパフォーマンスの向上にアクセスできます。ただし、下位互換性のない変更が含まれる場合があるため、注意深い計画と検証が必要です。

メジャーバージョンのアップグレードは通常毎年発生し、新しい SQL 機能、パフォーマンスの強化、最新の標準との互換性の向上などの重要な進歩をもたらす可能性があります。

スムーズなアップグレードプロセスを保証には、新しいバージョンに対してアプリケーションと拡張機能をテストし、非推奨または変更された機能のリリースノートを確認することが重要です。

Hybrid ManagerHM を使用すると、ユーザーは自分の裁量でメジャーバージョンのアップグレードを管理できます。新しいメジャーバージョンが検証および公開されると、HM のイメージライブラリですぐに利用できるようになり、既存のクラスターの制御されたアップグレードが有効になります。

> **注釈**
>
> PG メジャーバージョンのアップグレードには、データの再起動が必要です。ダウンタイムが予想されます。

メジャーアップグレードは大きな利点を提供しますが、注意深い準備が必要です。

続行する前に、クラスターの完全なバックアップを実行し、包括的な復旧計画を検証することを強くお勧めします。これにより、アップグレードプロセス中に互換性または操作の問題が発生した場合に、データを復元できます。

データベースクラスタータイプ別のメジャーバージョンのアップグレード

サポートされているクラスタータイプで Postgres メジャーバージョンを簡単にアップグレードできます。

- 単一ノードおよび高可用性クラスターは、インプレース、オフライン *upgrade procedure* を使用してアップグレードできます。
- Advanced 高可用性 AHA および分散高可用性 DHA クラスターは、インプレース、オンライン、*upgrade procedure* をサポートします。
- また、DHA クラスターは、 *blue-green deployment approach* を使用したゼロダウンタイムのメジャーアップグレードもサポートしています。

ロールバックの考慮事項

メジャーバージョンのアップグレードで使用できる自動ロールバックはなく、ダウングレードの実行はお勧めできません。メジャーアップグレードを元に戻すには、優先メジャーバージョンを実行し、アプリケーションのワークロードを新しいクラスターにスイッチする新しいクラスターで最新のバックアップから復元します。次に、PG メジャーアップグレードされたクラスターを削除するか、さらにトラブルシューティングを行うことができます。

Postgres major version upgrades on Hybrid Manager—single node or High Availability clusters

前提条件

- HM を起動して正常な状態でクラスターを作成します。クラスターが正常な状態にあることを確認するには、クラスターのメインページから **Monitoring** タブを選択します。次に、クラスターとデータベースメトリックを確認し、すべてが動作していること、すべてのメトリックが定義された許容可能なパラメーターの範囲内にあることを確認します。
- 続行する前に、クラスターの完全なバックアップを実行し、包括的な復旧計画を検証することを強くお勧めします。

HM で作成した既存のクラスターの PG メジャーバージョンをアップグレードする

1. 最初に、クラスターのメインページから **Quick Action>Upgrade Cluster** を選択し、 **Major Upgrade** を選択します。
2. **Upgrade Cluster : Major** ダイアログウィンドウが開きます。
3. 優先する新しいメジャーバージョンのイメージを選択します。
4. アップグレードパスを確認し、[**Continue**] ボタンを選択して確認します。
5. ダイアログウィンドウには、アップグレードで予想される特定のパッケージの変更が表示されます。これらを確認し、すべてが正しい場合は**アップグレードクラスター** を選択します。
6. アップグレードは自動的に適用されます。 次に、 **Monitoring** 、および **Logs** タブを使用して、プロセスをモニタしたり、予期しない動作のトラブルシューティングを行ったりできます。

注釈

まれに、特定の製品リリースのメジャーアップグレードが完了した後、追加の手順が推奨される場合があります。 問題の製品のリリースノートを参照して、あなたの場合にこれが必要かどうかを確認します。

Postgres major version upgrades on Hybrid Manager—Advanced High Availability or Distributed High Availability

アップグレードの考慮事項

- **アップグレードスキップ**直接メジャーバージョンスキップはサポートされていないことに注意してください。すべてのアップグレードは、一度に 1 つのメジャーバージョンごとにシークエンシャルに進行する必要があります。

正解 16 > 17 > 18 不正解 16 > 18

- **画像タグの一貫性**画像タグの一貫性を維持します。たとえば、<old-image-version>-<distribution>_full から<new-image-version>-<distribution>_full にアップグレードします。これを行わないと、パッケージ化された拡張機能のセットが変更される場合があります。
- **物理的結合** AHA および DHA クラスターは日々のデータ同期に論理レプリケーションを使用しますが、メジャーバージョンアップグレードのプロセスには、**物理的結合**物理レプリケーションが含まれ、間のバージョン変更を促進します。データグループ。これにより、データグループをインプレースでアップグレードでき、一時的に別のメジャーバージョンを実行できます。ただし、メジャーバージョンが異なる新しいデータグループを追加することはできません。

前提条件

- HM を起動して正常な状態でクラスターを作成します。クラスターが正常な状態にあることを確認するには、クラスターのメインページから **Monitoring** タブを選択します。次に、クラスターとデータベースメトリックを確認し、すべてが動作していること、すべてのメトリックが定義された許容可能なパラメーターの範囲内にあることを確認します。
- 続行する前に、クラスターの完全なバックアップを実行し、包括的な復旧計画を検証することを強くお勧めします。

HM で作成した既存のクラスターの Postgres メジャーバージョンをアップグレードする

高度な高可用性または分散高可用性クラスター複数のデータグループを持つクラスターのメジャーアップグレードの場合、すべてのデータと監視グループを順番にアップグレードする必要があります。グループの順序は重要ではありませんが、次のグループを開始する前に、各グループのアップグレードが完全に完了していることを確認します。

1. クラスターのメインページで、アップグレードする最初のデータグループの横にある 3 つのドットメニュー **Upgrade Data Group** を選択し、 **Major Upgrade** を選択します。

[**Upgrade Cluster : Major**] ダイアログウィンドウが開きます。

1. 優先する新しいメジャーバージョンのイメージを選択します。
2. アップグレードパスを確認し、[**Continue**] ボタンを選択して確認します。

3. ダイアログウィンドウには、アップグレードで予想される特定のパッケージの変更が表示されます。これらを確認し、すべてが正しい場合は**アップグレードクラスター** を選択します。

4. アップグレードは自動的に適用されます。PGD CLI を使用して、データグループが正常な状態になることを確認できます。すべてのチェックのステータスが `Ok` であれば、アップグレードは成功しています。

5. 最初のデータグループが正常にアップグレードされ正常になったら、クラスター全体が新しいバージョンになるまで、残りのデータと監視グループを 1 つずつアップグレードします。

> **注釈**
>
> まれに、特定の製品リリースのメジャーアップグレードが完了した後、追加の手順が推奨される場合があります。 問題の製品のリリースノートを参照して、あなたの場合にこれが必要かどうかを確認します。

Seamless major upgrades with blue-green deployments

メジャーアップグレードのブルーグリーンアプローチ

分散高可用性クラスターでは、信頼性が高く、安全で、即時のメジャーアップグレードが可能です。この方法は、さまざまなデータグループに基づいて、分散データベースに blue/green deployment アプローチを使用することで構成されています。

- 既存の `blue` データグループは、現在の安定した環境プロダクションデータベースなどを表します。
- 新しい `green` データグループがデータベースクラスターに作成され、次のメジャーバージョンにアップグレードされます。
- `green` データグループに接続することにより、アプリケーションで包括的なテストを実行します新しい接続文字列を利用できます。
- すべてのクライアントワークロードを `green` データグループにリダイレクトします。
- 最後に、アップグレードが完全に検証されたら、`blue` データグループを削除します。

前提条件

- ハイブリッドマネージャー上の古いバージョンの `blue` データグループを使用した分散高可用性クラスターが正常な状態で稼働していること。
- アセットライブラリーで次の PG メジャーバージョンを利用できるようにします。

一般的な推奨事項

- アップグレードプロセスを開始する前に、常に完全なバックアップを実行します。
- 可能であれば、実稼働前環境でアップグレードプロセスをテストします。
- アップグレードプロセスは短時間で完了することをお勧めします同日など。データグループで異なるメジャーバージョンを実行すると、予期しない非互換性の問題が発生する可能性があります。

クラスターのステータスを確認する

クラスターが正常な状態にあることを確認するには、クラスターのメインページから **Health Status** タブを選択します。次に、クラスターを確認します。

- **Raft Status** 「OK」と読み込まれていることを確認します。
- **Replication Slot Status** 「OK」と読み込まれていることを確認します。
- **クロックスキュー** 「OK」を読んでいることを確認します。
- **プロキシステータス** すべてのプロキシが稼働していることを確認します
- **ノードステータス** すべてのノードが起動していることを確認します
- **トランザクションレート**許容/予想されるパラメーター内であることを確認するため

次を含む各データグループとデータグループ内のノードに関する詳細をさらに調査できます。

- データグループの **プロキシステータス**
- データグループの **ノードステータス**
- グループ内の各ノードの次の詳細
  - **接続** の数
  - **ウォールサイズ**
  - **メモリ** の使用率 b_tran_5 **ストレージ** の使用率

最新の **レプリケーションステータス**の詳細を確認して、ノード間のレプリケーション遅延を確認できます。

さらに、PGD CLI を使用して view data group state またはレプリケーションピアで deeper monitoring を実行して、すべてのデータが予想通りに同期されていることを確認できます。

これらのメトリックのいずれかが許容可能な/予想されるパラメーターの範囲外である場合、アップグレードプロセスを続行する前にこれらの問題に対処する必要があります。

クラスターを編集する

ページの右上隅にある **Quick Actions** ドロップダウンメニューを使用して開始し、 **Edit Cluster** を選択してアップグレードプロセスを開始します。

- 次に、[**クラスター設定**] タブを選択し、パスワードを再度設定します。
- 次に、 **Data Groups** タブを選択し、古いバージョンの「blue」データグループを見つけて、**Duplicate** ボタンを選択します。これにより、既に持っているデータグループの下に新しいデータグループが表示されます。
- グループの名前をデフォルトから変更し、**ノード**アーキテクチャ 2〜3 ノードまたは 3〜2 ノード、および **展開場所** 、**データベースタイプ** を変更できます。 、 **インスタンスサイズ** 、**ストレージ** 、 **ネットワーク** 、 **バックアップ** 、および**セキュリティ**設定。このアップグレードプロセスを使用するほとんどの場合、古いバージョンのデータグループを、同じ負荷を処理できる別の新しいバージョンのデータグループに置き換えようとしているため、**グループ名**を除き、すべてを同じにしておきます。ただし、特定の状況で何かを変更する必要がある場合は、アップグレードプロセスのこの時点でそれを行うことができます。

- この時点までに奇数のデータグループのみがあった場合、[**Create Data Group**] ボタンを選択するとすぐに、デフォルト名 **Witness Group** の監視グループがセットに追加されます。データグループ。Witness グループは、raft コンセンサスを可能に保つために役立ちます。2 つのデータグループでこれが必要な理由は、偶数のデータグループで、いわゆる「スプリットブレイン」シナリオが発生する可能性があるためです。グループの半分が 1 つの決定を下したいと考えており、残りの半分が実行したい他の何か。監視グループは、これが発生するたびにタイブレーカーになることにより、このようなシナリオを防ぎます。
- **Save** ボタンを選択して、監視グループの構成を保存します。
- [**データグループ**] タブに、古い `blue` データグループ、新しい `green` データグループ、および監視グループの両方を表示できます。この画面で **Save** ボタンを選択すると、クラスターに行う変更をリストする **Return Changes** ポップアップが表示されます。このポップアップには、変更を適用するとクラスター全体でデータベースの再起動がトリガーされる可能性があることを示す警告も表示されます。これは事実ですが、変更の適用によりダウンタイムが発生することを意味するものではありません。この間、古いバージョンのデータグループの少なくとも 1 つのノードはアクティブなままです。
- システムのポップアップで **Jump Changes** ボタンを選択して、要求したクラスター変更の制定を実際に開始します。これは、基になるハードウェアとデータベースのサイズによっては、かなりの時間がかかる場合があります。

`green` データグループのデータを検証する

クラスター全体が クラスターのステータスを確認する にあることを確認します。

[**概要**] タブから、新しいデータグループの接続文字列をコピーできます。各データグループには異なる接続文字列が公開されるため、既存のデータグループに接続しているアプリケーションは、プロセスが完了するまで影響を受けません。新しい接続文字列を使用して、すべてのデータが同期されていることを検証するなどのテストを実行できます。

`green` データグループを次のメジャーバージョンにアップグレードする

- [**Overview**] タブから、新しいデータグループを特定し、3 つの点アクションボタンの下の **Upgrade data group** をクリックします。
- 次に、[ソフトウェアイメージ] ドロップダウンリストからターゲットバージョンを選択します。次のメジャーバージョンで利用可能な最新のマイナーにアップグレードすることをお勧めします。[続行] をクリックして、アップグレードの詳細を確認します。
- システムのポップアップで **Upgrade Cluster** ボタンを選択して、実際にデータグループのアップグレードを開始します。プロセスが完了するまで数分待ちます。

アップグレードの確認

概要 と **プロパティ**タブの両方に、データグループの新しいバージョンが表示されます。クラスターが クラスターのステータスを確認する にあることを確認します。

テストを計画する

「ブルー」環境から切り替える前に、新しいアップグレードバージョンを使用してアプリケーションとクライアントを厳密にテストすることを強くお勧めします。

テストを実行するために書き込み操作またはスキーマの変更が必要な場合、それらは古いバージョンのデータグループに自動的にレプリケートされます。これを管理するには、enabling or disabling the replication subscriptions によってクラスターのレプリケーション構成を変更できます。

たとえば、次の SQL コマンドを使用して、新しいアップグレードバージョンのデータグループから古いバージョンのデータグループへの変更を無効にできます。古いバージョンのデータグループのノードに接続したときに実行する必要がある SQL コマンドは、新しいデータグループから変更をレプリケートするサブスクリプションを無効にします。

```
SELECT bdr.alter_subscription_disable(sub_name, true, true)
    FROM bdr.subscription s, bdr.node n, bdr.node_group ng
  WHERE n.node_group_id = ng.node_group_id
    AND s.source_node_id = n.node_id
    AND ng.node_group_name = <new version data group name>;
```

スイッチオーバーを計画する

すべてのテストが実行されたら、新しい接続文字列を使用してワークロードを green データグループに切り替える準備が整います。これには、アプリケーション、クライアント、またはインフラストラクチャサービスの変更が必要になる場合があります。

新しいバージョンのデータグループと古いバージョンのデータグループ間の双方向同期を以前に無効にした場合、次のコマンドで再度有効にする必要があります。

```
SELECT bdr.alter_subscription_enable(sub_name, true)
    FROM bdr.subscription s, bdr.node n, bdr.node_group ng
  WHERE n.node_group_id = ng.node_group_id
    AND s.source_node_id = n.node_id
    AND ng.node_group_name = <old version data group name>;
```

クラスターが クラスターのステータスを確認する にあることを確認します。

blue データグループを削除します

アップグレード手順の最後のステップは、古いバージョンのデータグループを削除することです。クラスターページの**概要**タブで、削除する古いバージョンのデータグループのトリプルドットドロップダウンメニューを選択し、**Delete Data Group** アクションを選択します。

削除を確認するポップアップが表示されます。削除を確認し、持っている他の古いバージョンのデータグループまたは不要になった場合は Witness データグループに対して削除プロセスを繰り返し、アップグレードが完了しました。

クラスターが クラスターのステータスを確認する にあることを確認します。

ロールバックの考慮事項

古いバージョンのデータグループが削除されると、このアプローチでは使用可能な自動ロールバックはありません。メジャーアップグレードを元に戻すには、目的のメジャーバージョンを実行する新しいデータグループで最新のバックアップを復元し、アプリケーションとクライアントのワークロードをその新しいデータグループに切り替えることを検討します。

Migrating databases with Hybrid Manager

Hybrid ManagerHM は、外部データベースからお客様の環境への移行を実行するための統合ソリューションを提供します。この機能により、既存のデータ、スキーマ、アプリケーションを一元的な管理ハブに取り込むことができ、管理と監視が簡単になります。

データの移行、移行プロセスは主に EDB Data Migration ServiceDMS を活用します。移行ポータルや AI コパイロットなどの他の統合と連携して、データベースの評価、変換、移行のための包括的なワークフローを提供します。

HM で移行機能を使用するにはどうすればよいですか?

ソースデータベースと宛先データベースを HM コンソールに接続する一連の手順を使用して、移行プロセスを実行します。ワークフローを理解するには、*AI Factory Architecture on Hybrid Manager* を参照してください。

EDB Data Migration Service はどのような種類の移行をサポートしていますか？

EDB DMS は、外部データベースから HM 管理データベースへ、外部データベースから他の外部データベースへ、HM 管理データベースから外部データベースへの移行をサポートします。考えられる移行パスのエンドツーエンドの例については、

Choosing your migration path を参照してください。

サポートされているソースおよび宛先データベースについては、*Supported databases and operating systems* を参照してください。

サポートされている移行方法については、 **移行タイプ** を参照してください。

EDB Data Migration Service はどのようなテクノロジーを使用しますか?

DMS は、Apache Kafka とオープンソースの Debezium CDC プラットフォームに基づいて構築されています。変更データキャプチャ CDC とイベントストリーミングを使用して、ソースデータベースの行の変更が移行先にレプリケートされます。

Planning

Migration architecture

Hybrid ManagerHM は、さまざまなソースから HM 管理および外部管理の Postgres クラスターにデータベースを移行するための統合プラットフォームを提供します。移行プロセスは、スキーマ変換、データ転送、および継続的レプリケーションを処理するために連携するいくつかの主要コンポーネントに依存しています。

Oracle から

図 6: Migrations from Oracle databases

Oracle からの移行は、EDB Postgres Advanced Server クラスター HM 管理または外部管理をターゲットとするのに最適です。2 つのデータベースシステムの違いを考慮すると、これらの移行には、スキーマとデータの移行の両方を処理するための特定のコンポーネントのセットが必要です。EDB EDB Postgres Advanced Server は、この目的に最適化されています。

Oracle データベース移行するデータとスキーマを含むソースデータベース。

エージェント Oracle ソースに接続して、移行のスキーマを読み取りおよび抽出するコンポーネント。また、ソースを HM に登録すると、移行評価、およびスキーマの移行を有効にするために HM コンソールにソースデータベースのビューが提供されます。

リーダーモードの DMS エージェント Oracle ソースに接続して、移行用のデータを読み取りおよび抽出するコンポーネント。

ハイブリッドマネージャー HM 中央コントロールプレーン。

HM コンソールユーザーがすべての移行アクティビティを管理および監視する Web ベースのインターフェイス。

移行ポータル EDB Postgres Advanced Server との互換性を保証しながら、Oracle 固有のスキーマ評価、変換、および宛先データベースへのインポートを処理する HM のサービス。AI コパイロットを使用して、スキーマ関連の問題を支援することもできます。

データ移行サービス DMS 実際のデータ転送スナップショットと継続的変更レプリケーションストリーミングを実行するコアサービス。これには、HM 管理データベースへの移行のためのビルトインライタが含まれています。

HM 管理 EDB Postgres Advanced Server データベース HM によって作成および管理される宛先データベースクラスター。

ライタモードの DMS エージェント移行先データベースが HM の外部の自己管理データベースである場合にのみ必要なコンポーネント。CDC レコードを消費して、宛先データベースに変更を複製および適用します。

外部 EDB Postgres Advanced Server データベース自己管理データベースとも呼ばれる、移行の外部管理の宛先。このシナリオでは、宛先ホストに接続してデータを受信および適用するライタモードの追加の DMS エージェントコンポーネントが必要です。

移行を実行する方法を示すエンドツーエンドの例については、Oracle to HM-managed database を参照してください。

Postgres から

図 7: Migrations from Postgres databases

スキーマには既に互換性があるため、Postgres データベース間の移行はより合理化されたプロセスです。この移行は、主に効率的なデータ転送とレプリケーションに焦点を当てており、移行には特定のコンポーネントのセットが必要です。

外部 Postgres データベース移行するデータを含むソース Postgres データベース。

Migration Toolkit または pg_dump: スキーマ自分自身は変換を必要としないため、初期スキーマの移行に使用される外部ツール。

エージェントソースを HM に登録して、モニタリングのために HM コンソール内のソースデータベースのビューを提供するコンポーネント。

リーダーモードの DMS エージェントソース Postgres に接続してデータを読み取りおよび抽出するコンポーネント。

ハイブリッドマネージャー HM 中央コントロールプレーン。

HM コンソールユーザーがすべての移行アクティビティを管理および監視する Web ベースのインターフェイス。

データ移行サービス DMS 実際のデータ転送スナップショットと継続的変更レプリケーションストリーミングを実行するコアサービス。これには、HM 管理データベースへの移行のためのビルトインライタが含まれています。

HM 管理の Postgres データベース HM によって作成および管理される宛先データベースクラスター。 PostgreSQL、EDB Postgres Advanced Server、または EDB Postgres Extended Server です。

ライタモードの DMS エージェント移行先データベースが HM の外部の自己管理データベースである場合にのみ必要なコンポーネント。CDC レコードを消費して、宛先データベースに変更を複製および適用します。

外部 Postgres データベース自己管理データベースとも呼ばれる、移行の外部管理の宛先。このシナリオでは、宛先ホストに接続してデータを受信および適用する書き込みモードの追加の DMS エージェントコンポーネントが必要です。

移行を実行する方法を示すエンドツーエンドの例については、Self-managed Postgres to HM-managed database を参照してください。

Terminology

この用語は、EDB Data Migration Service DMS 機能を理解するために重要です。

変更データキャプチャ CDC

CDC は、 **デルタ**を計算することによりデータセットの変更を決定および追跡するために使用されるソフトウェアデザインパターンのセットです。EDB DMS は CDC を使用して、データベースクラスターから別のデータベースクラスターに変更をストリーミングします。

EDB DMS エージェント

DMS エージェントは、移行のためにデータベースを登録できるツールです。リーダーモードでは、データベースを移行のソースとして登録します。ライタモードでは、宛先として登録します。

ソースデータベースと宛先データベースを登録すると、DMS エージェントは DMS と連携してデータ移行を実行します。

!!!note Hybrid ManagerHM バージョン 1.3 以降では、 `cdcagent` としてパッケージ化された EDB Data Migration Service agent が、古い DMS Reader および DMS Writer を置き換えます。

互換性を保証には、すべての HM 1.3 以降の展開で DMS エージェントを使用します。バージョン 1.2 以前の HM 展開の場合、個別にパッケージされた EDB DMS Reader および EDB DMS Writer を使用します。

リーダーモードの EDB DMS エージェント

リーダーモードで実行されている場合、DMS エージェントは Debezium を使用してソースデータベースと Kafka /オブジェクトストレージで CDC 操作を実行し、変更イベントを含むメッセージを生成します。

ソースマシンに DMS エージェントをインストールします。また、ソースデータベースが、管理アクセスがないクラウドプロバイダーによって管理されるマシンで実行されている場合、ソースデータベースへのアクセスがある補助マシンにインストールすることもできます。

次に、このモードでソースに接続して読み取ることができるため、あらゆる種類の移行を実行するときにリーダーモードで構成して起動します。

ライタモードの EDB DMS エージェント

ライタモードで実行されている場合、DMS エージェントは Kafka/Object ストレージから CDC レコードを消費して、宛先データベースに変更を複製および適用します。

移行の宛先が HM 管理データベースの場合、HM に付属の DMS コンポーネントには既にビルトインライタが含まれているため、ライタモードで EDB DMS エージェントをインストールして起動する必要はありません。

宛先が自己管理データベースの場合、宛先マシンに追加の DMS エージェントをインストールします。次に、ライタモードで構成して起動します。そのモードでは、自己管理の宛先に接続してデータを移行できます。

HM エージェント

`beacon-agent` としてパッケージ化された HM agent は、データベースの可観測性、モニタリング、使用状況レポートなどを提供するツールです。移行のコンテキストでは、HM エージェントは、登録された各データベースの移行評価を実行します。登録された Oracle データベースの場合、*Migration Portal integration* での後続の処理のためにスキーマ抽出も実行します。

補助機

補助マシンは、データベース移行のために EDB DMS エージェントを実行する専用の Linux ベースのサーバーベアメタル、VM、またはローカルです。このマシンは移行エンジンとして機能し、ローカルまたはリモートにかかわらず、さまざまなソースおよび宛先データベースに接続できます。

このマシンは、ソースまたは宛先データベースがあるのと同じホスト、および HM が実行されているホストと同じにすることができます。唯一の要件は Linux オペレーティングシステムです。それが現在 EDB DMS エージェントをサポートしている唯一のプラットフォームであるためです。

移行タイプ

Creating a data migration with Hybrid Manager の場合、 **Snapshot** または **Snapshot + Streaming** オプションを使用して移行を実行するかどうかを選択できます。

スナップショットの移行

Snapshot オプションを選択すると、DMS は特定の瞬間にデータベースのイメージをキャプチャします。このポイント・インタイム・コピーは、移行に使用されます。スナップショットが作成された後にソースデータベースで実行された新しい挿入と更新は、宛先データベースにレプリケートされません。

スナップショット+ストリーミングの移行

Snapshot + Streaming オプションを選択すると、DMS はスナップショットから開始してベースラインコピーを確立し、常に CDC を使用して後続の変更を適用します。このアプローチにより、移行中のデータの一貫性が保証されます。DMS は、*Finalizing a streaming data migration* を行うまで、ソースデータベースで実行されたデータ更新を宛先データベースにストリーミングし続けます。

移行先のタイプ

Creating a data migration with Hybrid Manager の場合、管理対象 Postgres への移行を実行するか選択できます。

マネージド Postgres

管理対象 Postgres の宛先は、HM を介して作成および管理されるデータベースです。移行を実行しているプロジェクトで作成されたデータベースクラスターを移行先として選択できます。

外部 Postgres

外部 Postgres の宛先は、HM の外部で作成され、完全に管理するデータベースです。移行先として外部の Postgres データベースを選択するには、最初に *Configuring and running the EDB DMS agent in reader mode* をインストールして構成し、ライタモードで構成して、DMS が自己管理データベースに接続し、書き込み操作を実行できるようにする必要があります。

Apache Kafka

Apache Kafka は、オープンソースの分散イベントストリーミングプラットフォームであり、高性能データパイプライン、ストリーミング分析、データ統合、およびミッションクリティカルアプリケーションに数千の企業が使用しています。EDB DMS は、Kafka を使用して、ソースから宛先データベースへのデータストリーミングを管理します。

Debezium

Debezium は、Red Hat コミュニティによってサポートされている CDC 用の Java ベースのオープンソースプラットフォームです。リーダーモードで実行されている場合、EDB DMS エージェントは Debezium を使用して読み取り操作を実行し、ソースデータベースでデータ変更をキャプチャします。

古い用語

EDB DMS リーダー

HM バージョン 1.2 以前では、 `cdcreader` としてパッケージ化された *Installation phase of Hybrid Manager on Rancher RKE2* は、Debezium を使用してソースデータベースで CDC 操作を実行し、Kafka を使用して変更イベントを含むメッセージを生成するツールです。

ソースに接続して読み取ることができるため、あらゆる種類の移行を実行するときに DMS Reader をインストールして構成します。移行の宛先が HM 管理データベースの場合、*EDB DMS Writer* をインストールする必要はありません。宛先が別の自己管理データベースの場合、*EDB DMS Writer* をインストールします。

EDB DMS Writer

HM バージョン 1.2 以前では、 `cdcwriter` としてパッケージ化された *EDB Data Migration Service Writer* は、CDC レコードを使用して宛先データベースに変更をレプリケートおよび適用するツールです。

自己管理の宛先データベースへの移行を実行するときに、宛先マシンに DMS Writer をインストールして構成します。自己管理の宛先に接続できるようになります。任意の種類の移行の場合、*EDB DMS* **リーダー** もインストールします。

Migration features

Hybrid ManagerHM は、データベース移行のための包括的な一元化ソリューションを提供し、データ転送プロセスを簡素化および高速化するように設計された幅広い機能を提供します。

ソブリン移行ソリューション

オンプレミスまたはクラウド環境内に完全に展開された単一のプラットフォームから移行を実行します。このアプローチでは、データとインフラストラクチャを完全に制御でき、独自のネットワーク内での移行が主権的で安全であることが保証されます。

移行の評価

HM は、HM エージェントを活用して、Oracle データベースの移行に必要な複雑さの評価と推定作業量 LOE を生成します。登録されたソースデータベースを分析することにより、プラットフォームは潜在的な問題と移行の規模に関する早期の洞察を提供し、効果的な計画を支援します。

データベースとスキーマの管理

データベースの詳細ページからソースデータベースに関する詳細情報にアクセスします。そこでは、スキーマに関する情報と、移行評価要素の明確な内訳があります。

AI Copilot を使用した移行ポータル

移行ポータルは、Oracle から Postgres への移行のための強力なツールです。スキーマ、フラグ、非互換性の評価を自動化します。また、統合された AI コパイロットも提供し、複雑なスキーマの問題を迅速に特定して解決し、変換プロセスを合理化します。元々は EDB がホストする Web ツールでしたが、移行ポータルは HM に統合されました。この統合は、環境にサービスを展開し、展開の主権をサポートします。

柔軟な移行オプション

ニーズに最適な移行方法を選択してください

- スナップショット移行大量移行のための高速なワンタイムデータ転送。
- スナップショットとストリーミング最小限のダウンタイムを必要とする移行に適しています。変更データキャプチャ CDC を使用して、ソースから宛先データベースに進行中の変更を継続的にキャプチャおよびレプリケートします。

多様な移行先

このソリューションは、目的地を選択する際の柔軟性を提供します。HM でプロビジョニングされたデータベース HM 管理データベースまたは外部管理の Postgres データベース外部管理データベースに移行できます。

詳細な移行制御

移行する特定のデータベーステーブルと列を選択する機能を使用して、データの移行を正確に制御します。この機能を使用すると、必要なデータのみを移行できるため、移行時間とリソースの使用量を削減できます。

スキーマと列のマッピング

移行サービスは、柔軟なスキーママッピングを可能にし、宛先データベース内の別の名前を持つテーブルと列にデータを移行できます。この機能は、移行プロセス中にデータを再構成する場合に特に役立ちます。

Supported databases and operating systems

EDB Data Migration Service EDB DMS は、Hybrid ManagerHM の一部としてのみ利用できます。

サポートされているソースデータベース

次のデータベースバージョンがサポートされています。

Database version	Supported source
PostgreSQL 12 - 17	Y
EDB Postgres Advanced Server 12 - 17	Y
EDB Postgres Extended Server 12 - 17	Y
Oracle 19c	Y
Oracle 21c	Y
Oracle 23ai	Y
AWS RDS PostgreSQL 13-17	Y
AWS RDS Aurora (PostgreSQL compatible) 13-17	Y

サポートされている Postgres ソースデータベースには、外部管理自己管理データベースのみが含まれます。

!!!note

DMS エージェントは、Oracle 21c および 23ai データベースの移行をサポートしています。ただし、Migration Portal は行いません。これは、 21c および 23ai データベースを移行するには、スキーマの移行を手動で実行する必要があることを意味します。

Oracle

EDB DMS スタックでは、Oracle データベースでアーカイブログモードを有効にし、テーブルおよびデータベースレベルでサプリメンタルロギングデータを有効にする必要があります。詳細については、*Preparing your Oracle source for data migration* を参照してください。

コンテナデータベース CDB / PDB および非 CDB ソースがサポートされています。

Postgres

Postgres から Postgres へ移行する場合、クロスバージョンおよびクロスディストリビューションの移行がサポートされています。PostgreSQL 16 から PostgreSQL 17、または PostgreSQL 17 から EDB Postgres Advanced Server 17 など、PostgreSQL のあるバージョンから別のバージョンにデータを移行できます。例外は、ソースとして EDB Postgres Advanced Server から別のディストリビューション、たとえば、PostgreSQL または EDB Postgres Extended Server へ。

ソース Postgres インスタンスは、ベアメタルマシン、仮想マシン上、または AWS のようなクラウドサービスプロバイダー RDS または Aurora データベースでホストされる自己管理インスタンスにすることができます。

Postgres ソースには、レプリケーションを管理できるデータベースロールまたはユーザーが必要です。詳細については、

Preparing Postgres source databases を参照してください。

サポートされている宛先データベース

DMS は、HM 管理データベース HM で作成されたクラスター上と Postgres を実行している自己管理データベースの両方への移行をサポートしています。

Database version	Supported destination
PostgreSQL 11 - 17	Y
EDB Postgres Advanced Server 11 - 17	Y
EDB Postgres Extended Server 11 - 17	Y

!!!note

HM 管理データベースクラスターの場合、サポートはバージョン 16 および 17 のみです。

DMS は、表にリストされているすべてのデータベースへの移行をサポートしています。ただし、移行ポータルは、EDB Postgres Advanced Server バージョン 16 および 17 の場合のみスキーマの移行をサポートしています。バージョン 15 以前の場合、スキーマの移行を手動で実行する必要があります。

Oracle

EDB DMS は、Oracle 宛先への移行をまだサポートしていません。

Postgres

Postgres から Postgres に移行する場合、クロスバージョンおよびクロスディストリビューションの移行がサポートされています。たとえば、PostgreSQL 16 から 17、または PostgreSQL 17 から EDB Postgres Advanced Server 17 など。

Postgres の宛先には、レプリケーションを管理できるデータベースロールまたはユーザーが必要です。詳細については、

Preparing Postgres destination databases を参照してください。

サポートされているオペレーティングシステム

DMS エージェントは Linux で実行されます。詳細については、*Installing EDB DMS agent* を参照してください。

Known issues, limitations, and notes

これらは、次の既知の問題、制限、および注意事項です。

- データ移行の制限
- *Oracle* の制限事項
- *Postgres* の制限
- スキーマの取り込みの既知の問題

データ移行の制限

オブジェクト名の大文字と小文字の区別

EDB DMS は、現在、大文字と小文字を区別する名前を持つスキーマ、テーブル、列の移行をサポートしていません。

アップグレードされた宛先クラスター

HM インスタンスをバージョン 1.3 以降にアップグレードした後、Hybrid ManagerHM バージョン 1.2 以前でプロビジョニングされたクラスターを移行先として使用することはできません。アップグレードした環境で新しいクラスターを作成して、有効な移行先として使用する必要があります。

単一の宛先への複数の移行の考慮事項と制限

DMS は、1 つのソースから 1 つの宛先データベースへの複数のデータ移行の実行をサポートしていますが、特定の制限が適用されます。

- **テーブルの使用法**各宛先テーブルは、一度に 1 つの移行でのみアクティブに使用できます。
- **一意のリソース ID 外部宛先への移行に適用**外部宛先へ複数の移行を実行する場合、`DBCONFIG_DATABASES_0__RESOURCEID=${Resource_ID}` パラメーターを使用してライタモードで EDB DMS エージェントの各インスタンスに一意のリソース ID を割り当てる必要があります。このパラメーターを設定すると、同時移行が同じテーブルを同時に使用しようとすることを防ぎます。これにより、データの損失または重複が発生する可能性があります。

!!!note この resourceID 要件は、HM 管理の Postgres インスタンスをターゲットとする移行には適用されません。 .. ::

- **パフォーマンスプランニング外部宛先への移行に適用します** 一般に *System specifications* でパフォーマンスは強いままですが、同じ外部宛先データベースに 5 つ以上の移行を実行する場合、リソース競合を防ぐために、追加の CPU を割り当てることを検討します。宛先までのメモリ、およびボリューム IOPS/スループット。これは、宛先が外部データベースの場合に特に重要です。

- **リソース管理**各移行ジョブでは、必要な DMS エージェントコンポーネントの専用の非同時使用が必要です。

リーダーモードの DMS エージェントソース) 同じソースデータベースからの複数の同時移行には、リーダーモードで EDB DMS エージェントの単一の実行インスタンスを使用しないでください。HM コンソールは、ソースデータベースが移行でアクティブになると、利用可能なリストから削除します。ただし、後続の移行または別のソースデータベースからの同時移行には、リーダーモードで単一の DMS エージェントを使用できます。

ライタモードの DMS エージェント外部宛先移行が別の移行と同じ宛先を共有する場合でも、各移行にはライタモードの DMS エージェントの独自のインスタンスが必要です。

Oracle の制限事項

サポートされていない Oracle データ型

限られた数の Oracle データ型と機能は、EDB DMS でサポートされていません。

サポートされているデータ型に関する詳細なコメントについては、Debezium documentation を参照してください。

サポートされていない Oracle データ型は次のとおりです。

- BFILE
- ロング
- LONG RAW
- RAW
- UROWID
- ユーザー定義型 REF、Varray、ネストされたテーブル
- ANYDATA
- XMLTYPE
- 空間

EDB DMS は、 BLOB、CLOB、または NCLOB 列を含む Oracle テーブルのレプリケート、これらにも主キー制約がある場合にのみサポートします。テーブルに主キー制約がない場合、ストリーミングレプリケーションは INSERT 操作のみをサポートします。

`Nan` 、`+INF` 、および`-INF` 値を含む可能性のある Oracle の `BINARY_FLOAT` および `BINARY_DOUBLE` タイプは、EDB DMS ではサポートされていません。

EDB DMS は、Oracle の `VARCHAR` や `CLOB` のようなテキストフィールドタイプのヌル文字 0x00 を含む値をサポートしていません。

Oracle 21c データベース

DMS エージェントは、Oracle 21c データベースの移行をサポートしています。ただし、Migration Portal は行いません。これは、 21c データベースを移行するには、スキーマの移行を手動で実行する必要があることを意味します。

Oracle XE の互換性制限

Oracle XE Express Edition は、データベースレベルでのサプリメンタルロギングの有効化をサポートしていません。これは変更データキャプチャ CDC のコア要件であるため、EDB DMS はソースデータベースとしての Oracle XE と完全な互換性がありません。

- 予想されるエラー Oracle XE を使用する場合、DMS エージェントリーダーは通常、次のような例外をログに記録します。- Debezium コンシューマー$NAME は例外で停止しました。変更イベントプロデューサーで例外が発生しました。このコネクタは停止します。
- 機能制限スナップショットのみの移行のみが機能すると予想されます。スナップショットが成功した場合でも、Debezium エラーメッセージがログに表示される場合があります。
- 使用ポリシー Oracle XE でのスナップショットのみの移行は、テストまたは評価の目的にのみ適しています。この構成は、運用ユースケースではサポートされていません。

Postgres の制限

列でサポートされていないドメインタイプ定義

EDB DMS は、次のデータ型ドメインのデータ型としてユーザー定義ドメインを含む列を含むテーブルの移行をサポートしていません。

- 日付
- TIME
- タイムスタンプ
- インターバル
- UUID
- ENUM
- JSON
- XML
- POINT
- LTREE
- TSVECTOR

TSVECTOR タイプの移行の回避策

TSVECTOR タイプ列の移行はサポートされていませんが、この回避策を適用して移行を有効にすることができます。たとえば、ソースデータベースのこの messages テーブルには、TSVECTOR 列が含まれています。

```
CREATE TABLE messages (
    title       text,
    body        text,
    tsv         tsvector
);
```

移行を開始する前に、宛先データベースに TSVECTOR 列なしでテーブルを作成します。

```
CREATE TABLE messages (
    title       text,
    body        text,
);
```

移行を実行するときは、 tsv 列チェックボックスをクリアして **Review your Tables** ページでテーブルを確認するときに、HM コンソールの TSVECTOR 列を除外する必要があります。

移行が完了すると、title および body 列がターゲットデータベースに移行されます。これで、TSVECTOR 列をターゲットテーブルに追加できます。

```
ALTER TABLE messages ADD COLUMN tsv TSVECTOR DEFAULT NULL;
```

次に、次の UPDATE ステートメントを実行して、移行されたすべてのデータの列を設定します。

```
UPDATE messages SET tsv = to_tsvector(english, CONCAT(title,  , body)) WHERE tsv IS␣
↪NULL;
```

スキーマの取り込みの既知の問題

EDB Postgres Advanced Server ソース用に取り込まれた追加スキーマ

HM エージェントに登録したソース EDB Postgres Advanced Server データベースのスキーマのリストを表示すると、明示的に作成したスキーマに直接対応しない追加スキーマが観察される場合があります。これらの追加スキーマは、Oracle PACKAGE オブジェクトを表す Oracle 互換性機能セットの一部として EDB Postgres Advanced Server によって内部生成されます。これは既知の問題であり、将来のリリースで修正される予定です。それまでは、これらの生成されたスキーマは、移行の目的で無視できます。

その他の既知の問題

RHOS 環境のみでロードバランサーなしでサポートされているデータ移行

説明ハイブリッドマネージャー内で Kafka を公開するデータの移行に必要な主な方法は、ロードバランサーを介します。例外は RHOS 環境で、ネイティブルートを使用して Kafka を公開します。したがって、ロードバランサーが無効になっている非 RHOS 環境では、NodePort モードを使用して Kafka の公開をサポートするフォールバックメカニズムがないため、データ移行ワークフローは機能しません。

回避策この問題は、標準の Kubernetes ロードバランサーが使用できない場合でも Kafka にアクセスできるように、現在実装の対象となっています。RKE 環境でデータの移行を実行するには、現在ロードバランサーを構成する必要があります。

Optimizing migration performance

EDB Data Migration Service DMS は、Hybrid ManagerHM に含まれています。これを活用して、既存のデータベース Oracle、自己管理の Postgres などから HM 管理のデータベースクラスターにデータを移行できます。EDB DMS は、スナップショットとスナップショット+ストリーミングの 2 つの移行方法をサポートしており、最小限またはゼロのダウンタイムでデータを移行できます。

HM での移行ワークロードを理解する

EDB DMS は、HM が展開される場所と同じ Kubernetes 環境オンプレミス、AWS、Red Hat OpenShift、および Google Cloud 内で実行されます。これは、移行操作に必要なワークロードのほとんどが Kubernetes クラスターによって管理および最適化されることを意味します。

HM を実行するための一般的な要件を満たす Kubernetes 環境は、スムーズな移行に役立つため、一般にリソースを大幅に微調整する必要はありません。ただし、大規模なデータセットの場合、特定のコンポーネントを最適化することによりプロセスを大幅に高速化できます。

リソースの割り当てについて考慮する必要がある主なコンポーネントは次のとおりです。

- **リーダーモードの EDB DMS エージェント**このコンポーネントは、ソースデータベースと同じマシンで、またはソースデータベースと HM に接続できる補助マシンで実行できます。
- **宛先データベースクラスター**これは、HM によって管理されるデータベースまたは別の外部の自己管理データベースです。ただし、HM 管理データベースに焦点を当てます。ここにデータが存在します。
- **ライタモードの EDB DMS エージェント**このコンポーネントは、移行先が外部自己管理データベースクラスターであり、そのデータベースのホストマシンで実行されている場合にのみ必要です。

スナップショットまたはスナップショット+ストリーミング移行を実行するかどうかも、パフォーマンスに異なる影響を与えます。

- **スナップショット移行**これらは主に、大量の既存のデータをソースデータベースから宛先に効率的に転送することに焦点を当てています。これらのパフォーマンスは、合計データサイズと持続的なディスク I/O に大きく影響されます。
- **スナップショット+ストリーミング移行**最初に既存のデータのスナップショットを転送し、次に進行中の変更トランザクションをソースから宛先まで継続的にストリーミングします。ここでのパフォーマンスは、持続トランザクションレート TPS とソースの書き込みアクティビティに追いつく機能によって左右されます。

大規模なデータセットの移行パフォーマンスの最適化

HM および DMS は、リソースの使用量を処理するように設計されています。 100 GB より小さいデータセットの場合、それ以上のチューニングなしで高速な移行を実行するように構成されます。ただし、100 GB を超えるデータセットの場合、いくつかの対象を絞った変更で大幅にパフォーマンスが向上し、移行が迅速に行われる可能性があります。

宛先データベースのパフォーマンスの調整

宛先データベースクラスター HM 管理データベースまたは自己管理データベースを最適化すると、リソース調整から最も重要なパフォーマンスの利点が得られます。

!!!note

これらの推奨事項は、現在のテストにもとづいていますが、変更される場合があります。私たちは、移行パフォーマンスを向上させ、移行時間の削減に継続的に取り組んでいます。

ストレージの推奨事項

移行中、ストレージ IOPS は 2000 でピークに達し、平均スループットは約 300 MB/秒、ピークは最大 350 MB/秒です。これらの需要に対応するには

- WAL ボリュームとデータボリュームを分離します。データボリュームとログ先行書き込み WAL ボリュームの両方で、最小 2500〜3000 IOPS と少なくとも 200〜250 MB/秒のスループットを処理できるストレージボリュームを目指します。最適なパフォーマンスを得るには、WAL とデータに別個のボリュームを使用することを強くお勧めします。
- ユニファイドストレージと WAL ボリューム。ユニファイドストレージと WAL ボリュームは、パフォーマンスに悪影響を与える可能性があります。たとえば、3000 IOPS と 125 MB/秒のスループットで単一のボリュームを使用すると、移行時間が 40%以上増加する可能性があります。ユニファイドストレージと WAL ボリュームを使用する必要がある場合、300〜400 MB/秒のスループットをお勧めします。WAL がいっぱいになると、移行は失敗することに注意してください。

これを調整する方法

HM 管理データベースクラスターの場合

- クラスターのストレージ *Editing a cluster* によって WAL とデータボリュームの割り当てを調整します。
- デフォルトでは、HM 管理クラスターはデータと WAL ボリュームに別個のボリュームを使用しますが、クラスターの作成中に**先行書き込みログに別のストレージボリュームを使用する**オプションをクリアすると、後でこの設定を有効にすることはできません。
- ボリューム速度 IOPS またはスループットは、クラスターの基になるインフラストラクチャによって決まります。HM 管理のデータベースクラスターに移行する場合、クラスターで使用可能な特定のストレージクラス、およびそれに関連する IOPS およびスループット特性は、HM 管理者および基になるクラウドまたはオンプレミス環境によって決定されます。したがって、クラスターで使用可能なストレージオプションを確認し、HM 管理者と協力して、選択したストレージが推奨パフォーマンス基準を満たしていることを確認します。クラウドプロバイダーインフラストラクチャのプロビジョニングの詳細については、Request Amazon EBS volume modifications または Google Cloud - About Persistent Disk performance を参照してください。

メモリの推奨事項

5 GB のメモリで十分ですが、量が多いほど、移行中の最適なパフォーマンスが促進されます。宛先データベースには少なくとも 10 GB のメモリを使用します。

これを調整する方法

HM 管理データベースクラスターの場合

- *Editing a cluster* クラスターのインスタンスサイズ **クラスター設定**の下で、メモリ割り当てを調整します。
- **DB Configuration** で `shared_buffers` 、`effective_cache_size` 、および `wal_buffers` パラメーターを調整することもできます。

CPU の推奨事項

宛先データベースクラスターには、少なくとも 4 つの CPU コアが必要です。コア数が少ないと、移行パフォーマンスに悪影響を与えます。

これを調整する方法

HM 管理データベースクラスターの場合、クラスターのインスタンスサイズに応じて CPU 割り当てを調整します *Editing a cluster* 。

!!!note

移行プロセス中にこれらの最適化を適用します。移行が完了した後、進行中のワークロード要件に基づいて、必要に応じてこれらの値を再調整できます。

広範なテストの結果、移行のパフォーマンスに影響を与える最大の要素はボリューム速度 IOPS およびスループットであることが示されています。CPU を推奨値より減らすとパフォーマンスが低下する可能性がありますが、ストレージの IOPS とスループットが低下するほどではありません。ストレージのスループットを半分にすると、移行時間が 40%以上増加する可能性がありますが、CPU 数を 2 コアに半分にすると、10〜15%増加する可能性があります。これらの推奨事項は、最高の移行パフォーマンスを生み出すように設計されています。より低い値を使用しても、移行は成功する場合がありますが、速度は遅くなります。

WAL 構成

WAL チェックポイントは、データの移行中に最もリソースを消費するタスクの 1 つです。デフォルトでは、`max_wal_size` は 1GB に設定されています。これは、WAL ファイルに書き込まれるすべての GB に対してチェックポイントが作成されることを意味します。移行中に、これにより、チェックポイントが 30〜60 秒ごとに発生し、宛先データベースにかなりの負荷がかかる場合があります。

移行中に `max_wal_size` を増やすことにより、宛先データベースの負荷を軽減できます。テストでは、`max_wal_size` が大きくても移行速度に影響を与えないことがわかりました。たとえば、20 GB `max_wal_size` を使用した移行は、1 GB 設定の移行と同じ時間かかります。ただし、20 GB 設定を使用した移行では、消費するリソースが約 50%少なくなります。

WAL ロギングにより、移行されたすべてのデータがチェックポイントを介して適切に保存されることが保証されることを考えると、いくつかの重要な機能がそれに依存しているため、`max_wal_size` を無効にすることは、ほとんどの EDB データベースで実行可能なオプションではありません。

これを調整する方法

HM 管理データベースクラスターの場合

クラスターの DB 構成 *Editing a cluster* によって WAL 構成を調整します。`max_wal_size` パラメーターを検索し、新しいサイズを MB 単位で指定し、保存して構成を適用します。

EBD DMS エージェントのパフォーマンスの調整

EDB DMS エージェントの最適なパフォーマンスは、これらのコンポーネントがインストールされるホストマシンがこれらの仕様に従っている場合に達成されます。

リーダーモードの EDB DMS エージェント

CPU 3 CPU コア

メモリ 4 GB

ライタモードの EDB DMS エージェント

CPU: 2 CPU コア

メモリ 2.5 GB

共有ホストの考慮事項

最適な移行パフォーマンスを得るには、EDB DMS コンポーネントとソースデータベースが同じマシン/ホストで実行されている場合、リソースの割り当てを注意深く検討してください。

- **DMS エージェントはホストを共有します**両方のエージェントが同じマシン上にあり、1 つがリーダーモードで、2 つ目がライタモードである場合、組み合わせたリソース少なくとも 5 つの CPU コアと 6.5 GB メモリをプロビジョニングします。
- **DMS コンポーネントとソースデータベースはホストを共有します** DMS エージェントがソースデータベースと同じホスト上にある場合、ホストのリソースが大幅に増加して、データベースの操作ワークロードと移行の要求の両方に対応することを確認します。

Migrating

Choose your migration path

各種類のデータベースの移行を実行する方法に関するエンドツーエンドの手順については、以下を参照してください。

- *AWS RDS PostgreSQL or AWS RDS Aurora (PostgreSQL compatible) database to an HM-managed database*
- *Migrate your Oracle database to an HM-managed database*
- *Migrate your self-managed Postgres database to a different self-managed Postgres database*
- *Migrate your self-managed Postgres database to an HM-managed database*

Migrate your AWS RDS/Aurora PostgreSQL database to an HM-managed database

これらは、AWS RDS for PostgreSQL または Aurora PostgreSQL データベースから Hybrid ManagerHM によって管理されるデータベースへの移行を実行するために必要な手順です。

ソースデータベースの準備

データ移行のソースデータベースの準備には、移行専用のユーザーを作成し、データベース構成を変更してレプリケーション機能を有効にすることが含まれます。

1. *Preparing your AWS RDS/Aurora PostgreSQL source database* 。

データベースの評価オプショナル

データベースに HM エージェントをインストールおよび構成して、Postgres データベースの可観測性と監視データを取得します。このオプションの手順は、スキーマ数と名前、テーブル数、オブジェクト数などの高レベルの移行メトリックも提供します。

1. *Configure the HM agent to assess your database* 。

宛先データベースを作成する

HM インスタンスにクラスターを作成して、宛先データベースをホストします。

1. *Creating your HM-managed destination database* 。

スキーマを移行する

非互換性を処理し、スキーマを転送します。

1. *Understanding schema handling* 。
2. *Migrating Postgres schemas with Migration Toolkit* 。

データを移行する

データ転送を実行し、最終的なクリーンアップ手順を完了します。

1. *Creating Hybrid Manager project credentials for the migration* 。
2. *Configuring and running the EDB DMS agent in reader mode* 。
3. **詳細については** 。
4. *Creating a data migration with Hybrid Manager* 。
5. *Finalizing a streaming data migration* ストリーミング移行の場合のみ。
6. *Applying constraints after the migration is finalized* 。

Creating your HM-managed destination database

title:HM 管理の宛先データベースクラスターの作成 navTitle:HM 管理データベース

新しいデータベースを作成する

ハイブリッドマネージャー HM に宛先データベースクラスターがまだない場合は、HM が移行の宛先として使用できるクラスターを作成します。

1. HM コンソールで、データベースクラスターを作成する project を選択します。
2. プロジェクトで、移行の宛先として機能するように create a database cluster への **Create New** と **Database Cluster** を選択します。
3. **Clusters** ページで、クラスターを選択し、 **Quick Connect** オプションを使用してターミナルからデータベースにアクセスします。

4. 移行の宛先として使用する空のデータベースを作成します。または、デフォルトのデータベース `edb_admin` を使用できます。

既存のデータベースを使用する

または、既存の HM 管理データベースを移行の宛先として使用できます。既存のデータベースクラスターを移行の宛先として使用する前に、移行するテーブルと宛先で生成された負荷が既存のワークロードを干渉しないことを確認してください。

既存の HM 管理データベースを宛先として使用する場合は、次の手順に進みます。

Preparing your AWS RDS/Aurora PostgreSQL source database

title:“AWS RDS / Aurora PostgreSQL ソースデータベースの準備” navTitle: “AWS RDS/Aurora PostgreSQL ソースの準備”EDB Data Migration Service を使用して AWS RDS for PostgreSQL または Aurora PostgreSQL データベースからデータを移行する前に、ソースデータベースを以下に準備する必要があります EDB DMS エージェントがソースに接続し、レプリケーションを実行できるようにします。

論理レプリケーションを有効にする

論理レプリケーションを実行するには、リーダーモードの EDB DMS エージェントに以下が必要です。

- `rds.logical_replication` を 1 に設定
- `wal_level` を `logical` に設定
- オプションで、`max_wal_size` を

improved migration performance の 8
GB 以上の推奨設定に設定します

AWS RDS データベースはクラウドサービスプロバイダーによって管理されているため、psql でデフォルト構成を変更することはできません。代わりに、レプリケーションを有効にしてパラメーターグループを作成し、RDS インスタンスに関連付けます。

1. AWS マネジメントコンソール、AWS CLI、または RDS API のいずれかを使用した Create a parameter group 。パラメーターグループファミリーが、RDS データベースが使用している Postgres バージョンと一致することを確認します。

追加の構成情報については、 Parameter groups for Amazon RDS を参照してください。

2. AWS マネジメントコンソール、AWS CLI、または RDS API のいずれかを使用した Associate the created parameter group with your source Amazon RDS database 。
3. AWS マネジメントコンソール、AWS CLI、または RDS API を使用して Modify parameters in a parameter group 。
    - このパラメーターグループの場合、 `rds.logical_replication` から `1` を設定する必要があります。これを構成すると、データベースは `wal_level` を `logical` .
    - に設定します。
    - また、`max_wal_size` が適切な WAL LSN ライフタイムように構成されていることを確認します。

実稼働トラフィックが主に時間指定されたチェックポイントを生成し、WAL サイズに基づいて要求されたチェックポイントを生成しないように、max_wal_size 値を十分に大きく設定します。

ストリーミング移行プロセスでは、EDB DMS のクラウドインフラストラクチャの耐久性のあるメッセージストレージにストリーミングできるまで、WAL で変更を利用できる必要があります。max_wal_size の設定が小さすぎると、パフォーマンスに影響を与える可能性があります。また、Postgres LSN をストリーミングする前に WAL からドロップできることにより、移行プロセスを妨げる可能性があります。max_wal_size を少なくとも 8 GB に設定することをお勧めします。DBA に相談して、適切なサイズを決定します。

4. 変更を適用するには、 reboot your AWS RDS database 。

5. データベースに接続し、wal_level が logical に設定されているかどうか、および max_wal_size が構成どおりに設定されているかどうかを確認することにより、設定が正しく設定されたことを確認します。

```
SHOW WAL_LEVEL;
SHOW MAX_WAL_SIZE;
```

出力は論理的です。

CDC 移行のロールを作成する

1. 移行のロールを作成するには、postgres またはスーパーユーザーアクセスを持つユーザーとしてソースデータベースに接続します。

```
psql -h <PG_HOST> -p <PG_PORT> -U <PG_USERNAME> -d <PG_DB_NAME>
```

そこで

- <PG_DB_NAME> は、接続する Postgres データベースソースの名前です。
- <PG_HOST> は、Postgres データベースホストです。
- <PG_PORT> は、Postgres データベースポートです。
- <PG_USERNAME> は、ロールを作成および付与、移行するテーブルの所有権を変更し、レプリケーションスロットを作成できる管理ユーザーまたはスーパーユーザーです。

このコマンドは、 <PG_USERNAME> に関連付けられたパスワードの入力を求めます。

2. LOGIN を使用して移行用のロールを作成します。

```
CREATE ROLE <MIGRATION_ROLE> WITH LOGIN PASSWORD <MIGRATION_ROLE_PASSWORD>;
```

3. ロールに RDS_REPLICATION 能力を付与します。

```
GRANT RDS_REPLICATION to <MIGRATION_ROLE>;
```

4. <MIGRATION_ROLE> は、Postgres パブリケーションを自動作成できるように、移行するソーステーブルの所有者である必要があります。

これを行う方法は、新しいテーブルの所有者となる追加ロールここでは<REPLICATION_GROUP> を作成し、このロールをテーブルの元の所有者と<MIGRATION_ROLE> の両方に付与することです。

```
CREATE ROLE <REPLICATION_GROUP>;
GRANT <REPLICATION_GROUP> TO <MIGRATION_ROLE>;
GRANT <REPLICATION_GROUP> TO <ORIGINAL_OWNER>;
ALTER TABLE <TABLE_NAME> OWNER TO <REPLICATION_GROUP>;
```

そこで

- <MIGRATION_ROLE> は、移行データベースアクセスに使用する Postgres ロールまたはユーザーの名前です。
- <ORIGINAL_OWNER> は、テーブルの元のプロダクション所有者です。
- <REPLICATION_GROUP> は、パブリケーション自動作成のために移行するソーステーブルを所有するために使用されるロールの名前です。
- <TABLE_NAME> は、移行するテーブルの名前であり、大文字と小文字が区別されます。

5. 移行するテーブルを含むソーススキーマの USAGE を移行ロールに付与します。

```
GRANT USAGE ON SCHEMA <DB_SCHEMA> TO <MIGRATION_ROLE>;
```

新しい<MIGRATION_ROLE> にはスキーマアクセスがありますが、これらのスキーマのソーステーブルへの SELECT アクセスが必要です。

6. ソーステーブルの SELECT を移行ロールに付与します。スキーマ全体またはテーブルごとにアクセスを付与できます。

スキーマ内のすべてのテーブルに、次のコマンドを使用します。

```
GRANT SELECT ON ALL TABLES IN SCHEMA <DB_SCHEMA> TO <MIGRATION_ROLE>;
```

テーブルごとに、次を使用します。

```
GRANT SELECT ON <TABLE_NAME> TO <MIGRATION_ROLE>
```

そこで

- <DB_SCHEMA> は、移行するテーブルのデータベーススキーマ名です。
- <MIGRATION_ROLE> は、移行データベースアクセスに使用する Postgres ロールまたはユーザーの名前です。
- <TABLE_NAME> は、移行するテーブルの名前であり、大文字と小文字が区別されます。

7. ソースデータベースに対する CREATE 特権を移行ロールに付与します。これは pgoutput プラグインで必要であるため、ユーザーはパブリケーションを作成できます。

```
GRANT CREATE ON DATABASE <PG_DB_NAME> to <MIGRATION_ROLE>
```

構成を検証する

ソースデータベースを構成した後、Postgres 構成を検証します。このためには、DMS エージェントにパッケージされた構成検証スクリプトが必要です。

前提条件

1. DMS エージェントを実行しているマシンに Java/OpenJDK 17 以降がインストールされていることを確認します。

2. Install the EDB DMS Agent 。

3. /opt/cdcagent/ の EDB DMS Agent フォルダーに移動します。

構成検証スクリプトを実行する

この例は、検証スクリプトを実行する方法を示しています。

1. 移行する必要があるすべてのテーブルの名前を<schema_name>.<table_name>形式で含む配列を作成します。例

```
arr=(test1.table1 test1.table2 test1.table3 test1.table4)
```

1. 必要なパラメーターを使用してスクリプトを実行します。例

```
PG_USERNAME=postgres PG_PASSWORD=password PG_HOST=localhost PG_PORT=5432 DB_
↪NAME=postgres DBZ_USERNAME=debezium DBZ_PASSWORD=dbz ./postgresConfigValidation.sh "
↪${arr[@]}"
```

次のような応答が得られます。

```
  ***  [Transporter] - Get PostgreSQL Major Version
PostgreSQL major version: 16

  ***  [Transporter] - Validate WAL Level
wal_level:  logical
[Pass] wal_level is logical.

  ***  [Transporter] - Validate max WAL senders
max_wal_senders: 10
[Pass] max_wal_senders is at least 1.

  ***  [Transporter] - Validate max replication slots
max_replication_slots: 10
[Pass] max_replication_slots is at least 1.

  ***  [Transporter] - Validate max WAL size
max_wal_size: 8 GB
[Pass] max_wal_size is at least 8 GB.
```

(次のページに続く)

(前のページからの続き)

```
  ***  [Transporter] - Validate checkpoints
checkpoints_timed: 424
checkpoints_req: 1
[Pass] Timed checkpoints are more frequent than requested checkpoints.

  ***  [Transporter] - Check debezium user role
[Pass] User debezium is present
[Pass] User debezium has replication permission

  ***  [Transporter] - Check SELECT privilege on the tables to be migrated
[Pass] User debezium has select privilege on all tables to be migrated.

  ***  [Transporter] - Check CREATE privilege on database
[Pass] Role debezium has CREATE privilege on database postgres.
```

データベースは CDC 移行の準備が整いました。

!!!note ここで推奨されているようにソースデータベース設定を変更することにより、[Failed] または [Suggestion] ステータスに対処します。 [Failed] チェックは、DMS がデータ移行を実行する前に解決が必要な問題をブロックしています。[Suggestion] チェックでは移行が許可されますが、対処しないでおくとパフォーマンスに悪影響を与える可能性があります。

Configuring the HM agent to assess your database

Hybrid ManagerHM エージェントをインストールおよび構成して、Postgres データベースの可観測性と監視データを取得できます。このオプションの手順は、スキーマ数と名前、テーブル数、オブジェクト数などの高レベルのメトリックを提供し、移行計画に役立ちます。

!!!note 移行を実行するために HM エージェントの構成は必要なく、現在実際のスキーマオブジェクト DDL を取り込みません。

HM エージェントにデータベースを登録するには、以下を参照してください。

- Monitoring self-managed deployments
-

Monitoring Cloud Service Provider managed Postgres deployments

Understanding schema handling

title: スキーマの準備とインポート navTitle: スキーマの準備 deepToC:true EDB Data Migration Service EDB DMS を使用して cdcagent を使用するデータ移行を構成する前に、スキーマを準備して宛先データベースにインポートする必要があります。

EDB DMS はデータのみを移行するため、移行が発生する前に宛先データベースにスキーマをインポートすることが重要です。移行先データベースには、移行のためにスキーマを配置してデータを入力する必要があります。

データの移行が発生する前に、スキーマの制約の一部を含める必要があります。その他は、データの移行が完了した後に適用する必要があります。このアプローチにより、パフォーマンスを低下させることなく移行できます。

スキーマの整合性とパフォーマンスの考慮事項

宛先データベースの制約、トリガー、および WAL ロギングの存在は、データ移行のパフォーマンスに影響を与える可能性があります。このアプローチが可能な場合、スキーマ制約の 2 段階インポートをお勧めします。

主キーと一意性制約

EDB DMS は、主キー/一意制約を活用して、データを宛先データベースに移行するときに 1 回だけ配信を提供します。したがって、データの移行を開始する前に実行するスキーマインポートに主キー/一意制約を含める必要があります。他の種類の制約をスキーマのインポートから除外します。

主キー/一意制約のないテーブルの行の場合、少なくとも 1 回の配信のみを実現できます。データ移行の検証中に重複排除を実行できます。

!!!note

Not null 制約は、宛先サーバーのパフォーマンスに重大な影響を与えず、スキーマのインポートに含めることもできます。

外部キー、チェック、および除外コンストレイン

EDB DMS は、宛先データベースクラスターに対して変更イベントを並列適用できます。ただし、一部の制約タイプを移行すると、移行のパフォーマンスに悪影響を与える可能性があります。この種の制約は、一貫した参照統合ソースデータベースからの飛行中のデータ移行のコンテキストで、不必要な CPU とメモリの使用率をもたらします。

したがって、移行が完了した後にのみ、宛先データベースに外部キー、チェック、および除外制約を適用することをお勧めします。

Migrating Postgres schemas with Migration Toolkit

- – title: Postgres スキーマの準備とインポート navTitle: Postgres スキーマの準備 deepToC: true

Postgres データベースのスキーマ処理

いくつかの異なる方法を使用して、スキーマを準備し、サポートされていない制約を除外し、宛先データベースにインポートできます。以下は、Migration Toolkit を使用して Postgres 移行のスキーマをエクスポートおよびインポートする方法に関するガイドラインです。

DDL を移行して Postgres スキーマをインポートするための他の有効なルートは次のとおりです。

- 宛先データベースにスキーマを手動で作成する
- pg_dump を実行してスキーマを取得し、

pg_restore を実行してそれらを復元する

-

pgAdmin または psql を使用する

エクスポート

Postgres からデータを移行する場合、EDB Migration Toolkit を使用してスキーマを管理することをお勧めします。MTK の offline migration 機能は、データベースのスキーマを抽出し、制約を分離する簡単な方法を提供します。以下にいくつかのガイドラインを示します。

- `-offlineMigration` オプションを使用して、Migration Toolkit は、接続を構成せずに宛先に適用できる SQL スクリプトを生成できます。
- `- schemaOnly <schema_scope>` オプションを使用して、Migration Toolkit がスキーマのエクスポートのみに焦点を当てたスクリプトを生成できます。
- Migration Toolkit を実行した後、宛先データベースに適用する前に、生成された SQL スクリプトから次の制約を除外コメントアウトします。
  - 外部キー制約 s
  - チェック制約
  - 除外制約
- Migration Toolkit の実行後、宛先の別の名前にオブジェクトをマップする予定がある場合は、スキーマ SQL スクリプトを変更して新しい名前を反映します。SQL スクリプトが生成された後、宛先に適用する前に、スクリプト DDL でオブジェクト名を直接編集します。また、スクリプト内の名前を変更したオブジェクトの構造とデータ型がソースと正確に一致することを確認します。DDL スクリプトのオブジェクト名を手動で更新する場合は、更新されたオブジェクトを参照する他のオブジェクトの名前も更新します。たとえば、テーブルの名前を変更する場合は、元のテーブル名を参照するビューまたはコンストレインも更新します。

後のデータの移行中に、新しく割り当てられたオブジェクト名を選択します。詳細については、

Mapping schema, tables and columns during a migration を参照してください。

インポート

DDL を準備し、外部キー、チェック、および除外制約を除外した後、宛先データベースに接続し、SQL 形式の DDL ファイルをインポートします。

インポートを開始するには、生成された offline migration script を実行します。

!!!note

移行プロセスの後半、移行が正常に実行された後、除外された外部キー、チェック、および除外制約を宛先データベースに適用する必要があります。したがって、除外されたコンストレインを追跡して、必要なときに適用できます。

Creating Hybrid Manager project credentials for the migration

title: ハイブリッドマネージャー移行プロジェクトの資格情報の作成 navTitle: プロジェクトの資格情報の作成 deepToC:true データの移行を実行するには、資格情報が必要です。移行のソースまたは宛先をハイブリッドマネージャー HM に追加する前に、移行の資格情報を作成する必要があります。次に、それらを使用して、EDB DMS エージェントが HM に接続できるようにします。

HM project ごとに 1 セットの資格情報が必要です。たとえば、すべての移行に単一の HM プロジェクトを使用している場合、1 セットの資格情報のみが必要です。

1. HM コンソールで、移行アクティビティに使用しているプロジェクトを選択します。
2. プロジェクトで、 **Migrate > Credentials** を選択します。

Credentials ビューには、プロジェクト内のすべての資格情報が表形式で表示されます。各資格情報のエントリには、その名前、ステータス、作成日、および有効期限が表示されます。

3. **Create Migration Credential** を選択し、資格情報の名前を入力して、**Create Migration Credential** を選択します。
4. 資格情報が生成されたら、 **Download Credential** を選択します。

!!!note

資格情報をダウンロードできるのはこのときのみです。今すぐダウンロードせず、資格情報が必要な場合、このセットを無効にして、新しいセットを作成する必要があります。

5. ダウンロードした zip ファイルを解凍し、生成されたフォルダーを EDB DMS エージェントがインストールされているホストにコピーします。

解凍された資格情報フォルダーには、次のファイルが含まれています。

- `int.crt` - `int.truststore.p12` - `client.keystore.p12` - `client-key.pem` - `client-cert.pem`

Configuring and running the EDB DMS agent in reader mode

title: “AWS RDS / Aurora PostgreSQL ソースのリーダーモードでの EDB DMS エージェントの構成と実行” navTitle: “AWS RDS / Aurora PostgreSQL の DMS エージェントの構成” deepToC: true リーダーモードで DMS エージェントを構成および実行して、データを実行します移行。

前提条件

1. DMS エージェントを実行しているマシンに Java/OpenJDK 17 以降がインストールされていることを確認します。

2. Install the EDB DMS agent 。

3. `/opt/cdcagent/run-cdcagent.sh` にある EDB DMS エージェントを書き込み権限で開きます。

AWS RDS for PostgreSQL または Aurora PostgreSQL からの移行のための EDB DMS エージェントの構成

環境に応じてスクリプトの変数を設定し、編集した行のコメントを外します。詳細については、parameters を参照してください。以下は、編集されていないスクリプトを示しています。

```
# !/bin/bash -e

#  run_cdcagent.sh

#
```

(次のページに続く)

(前のページからの続き)

```
#  This script provides a convenient place to specify

#  environment variables used to configure the

#  EDB Data Migration Service Agent.

#

##########################################
#  DMS Agent General Configuration       #

##########################################

#  This ID is used to identify DMS Agent

#  and is specified by the user.

export DBCONFIG_ID=<source_name_of_choice>

#  Determine the run mode of the DMS Agent

#  The run mode can be set to either reader or writer

#  reader mode is used to read changes from a source database

#  writer mode is used to write changes to a destination database

export RUN_MODE=reader

#  This is the DMS backend service used by the DMS Agent

#  Consult your system administrators

export RW_SERVICE_HOST=https://${TRANSPORTER_RW_SERVICE_DOMAIN_NAME}/transporter

##########################################
#  DMS Agent TLS Configuration           #

##########################################

#  You need to create migration credentials in EDB Postgres AI platform

#  and set the path of the credentials directory
```

(次のページに続く)

(前のページからの続き)

```
export CREDENTIAL_DIRECTORY_PATH=$HOME/credentials

##########################################
#  DMS Agent Source DB Configuration     #

##########################################

#  A sample configuration to create a single postgres database connection:

export DBCONFIG_DATABASES_0__TYPE=POSTGRES
export DBCONFIG_DATABASES_0__HOSTNAME=<RDS_endpoint>
export DBCONFIG_DATABASES_0__PORT=<5432>

#  The CATALOG is the database name

export DBCONFIG_DATABASES_0__CATALOG=<source>
export DBCONFIG_DATABASES_0__USERNAME=<postgres>

#  The password env can be set without specifying it here

#  but the env structure looks like this

export DBCONFIG_DATABASES_0__PASSWORD=<password>

#  The resource_id is configured by the customer in the configuration file of the␣
↪beacon agent

#  need to make sure that the resource_id is set to the same value as in the beacon␣
↪agent

# export DBCONFIG_DATABASES_0__RESOURCEID=1234567890abcdef

#  You can increase the index to configure more than

#  one database for the DMS Agent

# export DBCONFIG_DATABASES_1__TYPE=ORACLE
# export DBCONFIG_DATABASES_1__HOSTNAME=localhost
# export DBCONFIG_DATABASES_1__PORT=1521

#  The CATALOG is the database name

# export DBCONFIG_DATABASES_1__CATALOG=ORCLCDB/ORCLPDB1
# export DBCONFIG_DATABASES_1__USERNAME=oracle
```

(次のページに続く)

(前のページからの続き)

```
#  The password env can be set without specifying it here

#  but the env structure looks like this

# export DBCONFIG_DATABASES_1__PASSWORD=password

#  The resource_id is configured by the customer in the configuration file of the␣
↪beacon agent

#  need to make sure that the resource_id is set to the same value as in the beacon␣
↪agent

# export DBCONFIG_DATABASES_1__RESOURCEID=1234567890abcdef

##########################################
#  Optional Parameters Below             #

##########################################

#  FIPS mode can be enabled by setting the FIPS_MODE env variable to true

# export FIPS_MODE=true

#  Configure logging

#  Loglevel for the entire Quarkus application and underlying components

#  WARNING: Enabling this loglevel is extremely verbose

# export QUARKUS_LOG_LEVEL=DEBUG

#  Loglevel for the EDB package

#  Preferred mode for debugging issues related to the logic of the agent

# export QUARKUS_LOG_CATEGORY__COM_ENTERPRISEDB__LEVEL=DEBUG

cd "$(dirname "$0")"

echo "Starting in mode: $RUN_MODE"

case "$RUN_MODE" in
  reader)
    java -jar ./reader/quarkus-run.jar
    ;;
```

(次のページに続く)

(前のページからの続き)

```
  writer)
    java -jar ./writer/quarkus-run.jar
    ;;
  *)
    echo "Unknown RUN_MODE: $RUN_MODE"
    exit 1
    ;;
esac
```

!!!note

DBCONFIG_DATABASES_0__HOSTNAME 値を設定して、AWS コンソールの**接続とセキュリティ**の下にある RDS エンドポイントをポイントします。編集された例は foo_database.cb7asw1xxxxx.eu-west-0.rds.amazonaws.com です。

EDB DMS エージェントの実行

DMS エージェントを起動します。

```
cdcagent
```

Verifying your source appears in the HM console

ソース は、リーダーモードで EDB DMS エージェントを使用してハイブリッドマネージャー HM に接続するデータベースであるため、移行に使用できます。

ソースが EDB DMS エージェントによって認識され、移行に使用できることを確認するには

1. HM コンソールで、現在の移行に使用しているプロジェクトを選択します。
2. 左側のナビゲーションメニューから、 **Migrate > Sources** を選択します。

各ソースのビューには、データベース名、データベースタイプ Postgres または Oracle、cdcagent ID、現在のステータス、ソースタイプなどが表示されます。

詳細情報

HM コンソールのこのページで使用可能なデータの説明については、Sources を参照してください。

Creating a data migration with Hybrid Manager

title HM 管理の宛先への移行の作成 navTitle 移行の作成 EDB DMS エージェントリーダーモードで構成してハイブリッドマネージャー HM への接続を確立した後、HM コンソールで移行を作成します。移行は、EDB DMS の助けを借りて、ソースデータベースと宛先データベース間の同期を確立します。

1. HM コンソールで、移行を管理するプロジェクトを選択します。
2. プロジェクトで、 **移行>移行** を選択します。
3. [**移行**] ページで、[**Create New Migration**] を選択します。

4. **Create OLTP Migration** ページで、名前を移行に割り当てます。

5. 移行のソースを選択します。 EDB DMS エージェントの ID は、 **Source** メニューにリストされています。

6. 移行タイプを選択します**Snapshot** または**Snapshot + Streaming** 。

7. **Destination Type** で、移行用の HM 管理の宛先データベースクラスターとデータベースを選択します。次に、**Next** を選択します。

8. 移行するテーブルと列を選択します。必要に応じて、テーブルと列の名前を変更します。

9. **Create Migration** を選択します。

HM コンソールには、新しい移行が表示されます。

スナップショット+ストリーミング オプションを選択した場合、移行で**実行中** 状態が表示されるときに EDB DMS エージェントは常にデータをストリーミングしています。データの変更は、移行が実行されている限りソースから宛先データベースに複製されます。

!!!note

リーダーモードの EDB DMS エージェントは、データ変更をストリーミングします。DDL オブジェクトの変更はストリーミングされません。

Finalizing a streaming data migration

title: 移行のファイナライズストリーミング移行の場合 navTitle: 移行のファイナライズ移行を作成するときに**Snapshot + Streaming** オプションを選択した場合、EDB DMS エージェントは、移行を次のようにマークするまで、ソースデータベースで実行されたデータ更新を宛先データベースにストリーミングし続けます。完了しましたたとえば、カットオーバーを実行します。

Snapshot オプションを選択した場合、DMS は宛先へのスナップショット全体のコピーが終了した後、移行を完了としてマークするため、このページをスキップできます。

ストリーミングプロセスを停止します

このプロセスは、**Snapshot + Streaming** 移行にのみ適用されます。

データ移行ストリームを停止する最適な時期を注意深く選択することが重要です。ソースで新しい挿入が実行されていないことを確認し、最後の既知の挿入が宛先で同期されたことを確認します。

データが宛先データベースに存在することを確認した後、移行を完了としてマークすることにより、ストリームを停止できます。

1. Hybrid ManagerHM コンソールのプロジェクトで、 **移行>移行**を選択します。

2. 移行の横にあるチェックマークを選択して、移行が完了したことを示します。

または、リストから移行を選択して Migration details page を開きます。次に、 **Mark as Completed** を選択します。

これにより、ストリーミング手順が停止します。

Applying constraints after the migration is finalized

title: 制約の適用 EDB Data Migration Service EDB DMS を使用したデータ移行の開始時に、ソースデータベースのスキーマを準備およびインポートしました。次に、宛先データベースに接続し、スキーマとデータの移行から除外された制約を再適用します。

主キーと一意制約

主キーと一意制約の場合、宛先の Postgres データベースにテーブルと制約が既に作成されています。これにより、EDB DMS はそれらをソースオブジェクトにマッピングし、データを正常に移行できます。他に何もする必要はありません。

not null 制約をスキーマインポートに含めた場合、同じことが適用されます。

外部キー、チェック、および除外コンストレイン

スキーマの準備とインポート中に除外した外部キー、チェック、または除外制約を再適用できるようになりました。

データの完全性の確保

主キーまたは一意制約を持たないテーブルの行は、少なくとも 1 回配信で移行されたため、これらの行は重複である可能性があります。

重複排除は、検証の一部として実行できます。

検証

すべてのデータが移行されたことを確認するには、ソースデータベースと宛先データベースを比較します。

LiveCompare を使用できます。

Migrate your Oracle database to an HM-managed database

これらは、Oracle データベースからハイブリッドマネージャー HM によって管理されるデータベースクラスターへのエンドツーエンドの移行に含まれるタスクです。

ソースデータベースの準備

スキーマの取り込み、移行評価、および後続のデータ移行のためのソースデータベースの準備には、HM エージェントと DMS エージェントの両方の専用ユーザーを作成し、データベース構成を変更してレプリケーション機能を有効にすることが含まれます。

1. *Preparing your Oracle source database for schema ingestion* 。
2. *Preparing your Oracle source for data migration* 。

ソーススキーマを取り込み、データベースを評価する

Oracle データベースを使用して HM エージェントをインストールおよび構成します。このプロセスでは、移行評価を実行して、移行に必要な労力、複雑さ、および時間を検討するのに役立ちます。また、HM エージェントは、ビルトイン移行ポータルでスキーマの移行を実行するために使用できるスキーマ DDL を抽出します。

1. *Creating a machine user to register your Oracle database with the HM Agent* 。
2. *Installing the HM agent to register your Oracle database* 。
3. *Configuring the HM agent to register your Oracle database* 。
4. *Running the HM agent* 。
5. *Running the HM agent as a service* 。
6. *Verifying your Oracle database was assessed for migration* 。

宛先データベースを作成する

HM インスタンスにクラスターを作成して、宛先データベースをホストします。

1. *Creating your HM-managed destination database* 。

スキーマを移行する

移行ポータル統合を使用して、非互換性を処理し、スキーマを転送します。

1. *Understanding schema handling* 。
2. *Migrating Postgres schemas with Migration Toolkit* 。

データを移行する

データ転送を実行し、最終的なクリーンアップ手順を完了します。

1. *Creating Hybrid Manager project credentials for the migration* 。
2. *Configuring and running the EDB DMS agent in reader mode* 。
3. **詳細については** 。
4. *Creating a data migration with Hybrid Manager* 。
5. *Finalizing a streaming data migration* ストリーミング移行の場合のみ。
6. *Applying constraints after the migration is finalized* 。

Preparing your Oracle source database for schema ingestion

!!!note この構成は、データの移行には必要ありません。ただし、役立つ移行評価を提供し、スキーマを取り込むことによりスキーマ変換が促進されます。

Oracle データベースを Hybrid ManagerHM エージェントに登録すると、移行評価およびスキーマ抽出機能が提供されます。

Oracle データベースを評価するには、HM エージェントにデータベースへのアクセスと、評価データを取得するために必要な権限を持つ Oracle ユーザーが必要です。これらの権限は、自動スキーマ抽出とデータベース移行の評価に必要です。これらのロールが割り当てられていないが、機能容量が削減されている場合、HM エージェントは引き続き機能します。

前提条件

HM エージェントの実行に使用している補助マシンに Oracle の SQLPlus ユーティリティをインストールします。

Oracle ユーザーを作成および構成する

HM エージェントでのスキーマ抽出と評価用の Oracle ユーザー CDB の場合は_common user_、非 CDB の場合は通常のユーザーを作成します。後で DMS エージェントでのデータ移行にこの同じ Oracle ユーザーを使用できます。組織の通常のプロビジョニングプロセスに従ってユーザーを作成し、ユーザーが推奨される特権を確実に受け取ります。

または、現在の権限が次の要件を満たし、広すぎない場合は、既存のユーザーを使用できます。

ユーザーを作成するには、sysdba として Oracle CDB にログインします。または、非 CDB アーキテクチャを使用している場合は、sysdba として非 CDB データベースにログインします。

作成する Oracle ユーザーは、接続、スキーマメタデータの読み取り、および評価の実行を許可する次のリソースへのアクセスが必要です。

```
V$VERSION
V$INSTANCE
V$LICENSE
sys.all_tables
sys.all_tab_privs
```

ユーザーには次の特権が必要です。

- `CONNECT` データベース接続の場合。
- `CREATE TABLE` 評価プロセス中に一時的な作業テーブルを作成します。
- `SELECT_CATALOG_ROLE` または `SELECT ANY DICTIONARY` データベースのメタデータとスキーマ情報を読み取るため。

!!!note これらの権限は、スキーマの抽出と評価に必要な最小限です。特定の Oracle データベース構成とセキュリティポリシーによっては、追加の権限の付与が必要になる場合があります。

`CREATE TABLE` 特権は、評価スクリプトが実行中に一時テーブルを作成するために使用されます。データベースには永続的な変更は行われません。

`SELECT_CATALOG_ROLE` は、データディクショナリビューへの読み取りアクセスを提供します最小の権限で推奨。あるいは、`SELECT ANY DICTIONARY` は、すべての辞書オブジェクトへのアクセスを保証するより包括的なシステム権限です。いずれも評価には役立ちます。

セキュリティリスクを引き起こす可能性があるため、絶対に必要な場合を除き、`SYS AS SYSDBA`、`SYSTEM`、または `DBA` ロールを持つ他のユーザーとしてログインすることは避けてください。

Creating a machine user to register your Oracle database with the HM Agent

import MachineUser from '../../../monitoring/mon_ext_dbs/mon_with_agent/create_machine_user.mdx'

Installing the HM agent to register your Oracle database

'unversioned/tools/agent/installing/index.mdx' から InstallAgent をインポートします

Configuring the HM agent to register your Oracle database

Hybrid ManagerHM エージェント `beacon-agent` として展開を構成して、HM との安全な接続を確立し、Oracle ソースデータベースから必要なスキーマと移行データを取得します。

前提条件

*Create a machine user and access key with the `estate ingester* role <https://www.enterprisedb.com/docs/edb-postgres-ai/latest/hybrid-manager/using_hybrid_manager/migration/migration_guides/oracle_to_hm_managed/agent_creating_machine_user/>`_ が割り当てられます。

アクセスキーを`$BEACON_AGENT_ACCESS_KEY` という名前の環境変数に保存して、HM エージェントが後でアクセスできるようにします。

HM 接続パラメーターを取得する

HM エージェントには、HM インストール中に設定される特定の構成値が必要です。管理者またはインストーラーのみがこれらにアクセスできます。HM インスタンスの管理者またはインストーラーに連絡して、次の値を要求します。

- **ビーコンサーバーホスト名** HM インストールを構成するときに、管理者/インストーラーは `beacon_server` サービスの内部エンドポイントを指定する必要があります。HM インスタンスの管理者またはインストーラーに、インストールに使用される Helm チャート構成ファイルで `beacon_server` サービスに設定したホスト名 URL を提供するように依頼します。このために、`values.yaml` ファイルの `parameters.upm-beacon.server_host` または `BEACON_SERVICE_DOMAIN_NAME` に設定した値を検索します。この値は、後で HM エージェントを構成するために必要になります。
- **ルート証明書パス必要な場合** HM エージェントには、HM インスタンスの `beacon_server` サービスへの信頼できる接続が必要です。HM エージェントを実行しているサーバーが HM の TLS 証明書組織のセキュリティインフラストラクチャによって管理されているを本質的に信頼していない場合、HM インスタンスの管理者またはインストーラーに証明書を含むファイルを提供するように依頼します。この証明書は、HM エージェントを実行しているマシンに保存します。後で HM エージェントを構成するために、この証明書へのディレクトリパスを提供する必要があります。

`beacon_server` およびルート証明書の詳細については、Administrative tasks を参照してください。

ソースデータベース接続パラメーターを定義する

DSN は、HM エージェントが Oracle ソースデータベースに接続するために使用する接続文字列です。環境変数`$DSN` またはダイレクト URL 形式を使用して `databases.dsn` パラメーターを設定します。

Oracle データベースへの接続の DSN 文字列は、次の形式を使用する必要があります。

```
DSN="oracle://<username>:<password>@<host>:<port>/<oracle_db_name>"
```

そこで

- <username> および<password> は、データベースの準備時に権限を付与したユーザーに対応します。
- <host> は、Oracle データベースが実行されているインスタンスのホスト名です。
- <port> は、データを取得するデータベースのポート番号です。
- <oracle_db_name> は、データを取得するデータベースのカタログまたは名前です。たとえば、パスワード password で認証されたユーザー oracle として、ポート 1521 でデータベース ORCLPDB1 に接続するには、エージェントが Oracle データベースと同じインスタンスで実行されていると仮定します。

```
DSN="oracle://oracle:password@localhost:1521/ORCLPDB1"
```

!!!注 パスワードの特殊文字 Postgres または Oracle ユーザーのパスワードに特殊な非 ASCII 文字が含まれる場合、DSN 文字列またはパスワード環境変数でパスワードの URL エンコードバージョンを使用します。たとえば、パスワードが pa$$word の場合、$$ 文字を%24%24 としてエンコードします。

```
DSN="oracle://oracle:pa%24%24word@localhost:1521/ORCLPDB1"
```

構成ファイルの準備

エージェントの構成ファイルを準備します。

1. ホームディレクトリに HM エージェント構成ディレクトリを作成します。

```
mkdir ${HOME}/.beacon
```

!!!注 この場所が便利でない場合は、次のいずれかを使用することもできます。

- ***/etc/beacon* - *beacon-agent* コマンドを実行するディレクトリ HM エージェントは、*/etc/beacon* で始まり、*${HOME}/.beacon* で構成ファイルを検索します。最終的なフォールバックとして、実行されたディレクトリを検索します。 .. ::**
    1. このディレクトリ内に、ファイル *beacon_agent.yaml* を作成します。

```
touch .beacon/beacon_agent.yaml
```

1. 次のテンプレートをコピーして、新しいファイルに貼り付けます。このテンプレートは、HM コンソールが移行評価を生成し、評価とインポートのためのすべてのスキーマを取り込みできるように設計されています。

このテンプレートでは、 $BEACON_AGENT_ACCESS_KEY および$DSN は、前に構成した環境変数です。テンプレート内のすべての< >プレースホルダーを置き換えます。詳細は、Parameter reference を参照してください。

```
agent:
 access_key: $BEACON_AGENT_ACCESS_KEY
```

(次のページに続く)

(前のページからの続き)

```
  beacon_server: <beacon_server>:9443
  project_id: <project_id>
  providers:
    - "onprem"
  root_ca_path: ""
  schema_providers:
    - "onprem-schema"
provider:
  onprem:
    clusters:
      - resource_id: "<cluster_resource_id>"
        name: "<cluster_name>"
        nodes:
          - resource_id: "<node_resource_id>"
            dsn: $DSN
            schema:
              enabled: true
            poll_interval: 15s
            tags:
              - "<tag_names>"
```

1. ファイルを保存します。

HM エージェントを構成したら、実行するか、サービスとして実行するように構成できます。

環境変数を設定するための推奨事項

構成ファイルですべてのパラメーターを直接設定できますが、`$BEACON_AGENT_ACCESS_KEY` や`$DSN` などの機密データは、シークレットマネージャーで管理される環境変数として扱うことをお勧めします。この戦略により、機密の資格情報がプレーンテキストの構成ファイルに保存されるのを防ぎます。環境変数を使用する場合、HM エージェントの実行中にそれらが利用できることを確認してください。

実稼働環境のための最も安全なアプローチについて、組織の IT 部門に相談してください。

短期間の移行またはテストとデモの場合、アクティブなターミナルセッションで `export` コマンドを使用できます。

関連トピック

- スキーマ取り込みのより詳細な制御たとえば、スキーマを除外または含める場合、Controlling schema ingestion を参照してください。
- 管理者がルート証明書と `beacon_server` の値を取得する方法に関するガイダンスが必要な場合は、Administrative tasks を参照してください。

Running the HM agent

’../../../monitoring/mon_ext_dbs/mon_with_agent/run_agent.mdx’から StartAgent をインポートします

Running the HM agent as a service

継続的なメトリックストリーミングのために、データベースがハイブリッドマネージャー HM エステートで可視されたままであるように保証には、サービスとして実行するようにエージェントを構成します。サービスとして実行すると、システムの起動時に起動し、エラーの後に再起動できます。

エージェントをサービスとして設定しない場合、手動で実行した場合にのみメトリックがストリーミングされます。また、データベースは、エージェントがストリーミングを停止してから 72 時間後に HM 不動産から消えます。

エージェントをサービスとして設定する前に、インストールと構成を行います。

!!!note

エージェントパッケージの将来のバージョンでは、これが自動的に設定される可能性があります。

この例では、RHEL 9 マシンで実行するように HM エージェントを構成します。オペレーティングシステム支援は、値を置き換えるタイミングを知るのに役立ちます。

1. 選択したテキストエディタを使用してファイルを作成します。この例のコマンドでは、vi エディターを使用します。

```
sudo vi /usr/lib/systemd/system/beacon-agent.service
```

AWS EC2 インスタンスの他のデフォルトのシステムディレクトリは次のとおりです。

Operating system	Default system directory
RHEL	/usr/lib/systemd/system/beacon-agent.service
Rocky Linux	/usr/lib/systemd/system/beacon-agent.service
AlmaLinux	/usr/lib/systemd/system/beacon-agent.service
Ubuntu	/etc/systemd/system/beacon-agent.service
Debian	/lib/systemd/system/beacon-agent.service

1. 次のテンプレートを新しい `beacon-agent.service` ファイルに貼り付けて、プレースホルダー< >の情報を置き換えます。

```
# TEMPLATE

[Unit]
Description=Agent

After=network.target

[Service]
Type=simple
```

(次のページに続く)

(前のページからの続き)

```
User=<user>

WorkingDirectory=</home/user/>
Environment="PATH=</path/to/Oracle/db/binary>:</Oracle/user/binaries>:</system/
↪binaries>" # e.g. /u01/app/oracle/product/19.0.0/dbhome_1/bin:/home/oracle/.local/
↪bin:/home/oracle/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin
Environment="ORACLE_HOME=</path/to/Oracle/db/binary>" # e.g. /u01/app/oracle/product/
↪19.0.0/dbhome_1

ExecStart=</usr/local/bin/beacon-agent>

Restart=on-failure
RestartSec=60

[Install]
WantedBy=multi-user.target
```

Service settings	Placeholder to replace	Guidelines
User=	*<user>*	Provide the name of the system user which will run the Agent.
WorkingDirectory=	*</home/user/>*	Set the working directory to where you stored the *.beacon* directory. The path depends on the operating system.
Environment=	*"PATH=</path/to/Ora*	Set to all user and system directories that contain Oracle-related executables to ensure the Agent can use *sqlplus* for schema extraction during runtime. >"
Environment=	*"ORACLE_HOME=</path/*	Set to the directory of the Oracle database executables to enable *sqlplus* to extract schemas.
ExecStart=	*</usr/local/bin/beacon agent>*	Provide the path to the directory where the Agent was installed. The path depends on the operating system.

1. 次のコマンドを実行して、新しいサービスをリロード、有効化、および開始します。これらのコマンドは、前述のすべてのオペレーティングシステムに適用されます。

```
sudo systemctl daemon-reload
sudo systemctl enable beacon-agent.service
sudo systemctl start beacon-agent.service
```

1. エージェントはサービスとして実行されています。ログを確認します。

```
journalctl -u beacon-agent.service
```

Verifying your Oracle database was assessed for migration

Oracle データベースがハイブリッドマネージャー HM エージェントによって認識され、評価に利用できることを確認するには、HM コンソールで **Estate** を選択し、 **Migrations** を選択します。

データベースがテーブルに表示される場合、次のリソースにアクセスできます。

- 移行評価。データベーススキーマ数、コア数、移行の複雑さ、および作業レベルの概要を提供します。**Estate** ページの

Migrations tab から入手できます。

- スキーマの拡張分析、グローバルでより詳細な評価を提供する移行データベース detailed view 、および移行の実行に役立つ HM コンソールの他のツールへの直接接続を提供する**移行** ハブ。これらのリソースは、 **Estates** ページの **Migrations** タブから特定の Oracle データベースを選択すると利用できます。
- ビルトインの移行ポータルでスキーマの移行を実行するために活用できる自動スキーマ DDL 抽出。

Creating your HM-managed destination database

title:HM 管理の宛先データベースクラスターの作成 navTitle:HM 管理データベース

新しいデータベースを作成する

ハイブリッドマネージャー HM に宛先データベースクラスターがまだない場合は、HM が移行の宛先として使用できるクラスターを作成します。

1. HM コンソールで、データベースクラスターを作成する project を選択します。
2. プロジェクトで、移行の宛先として機能するように create a database cluster への **Create New** と **Database Cluster** を選択します。
3. **Clusters** ページで、クラスターを選択し、 **Quick Connect** オプションを使用してターミナルからデータベースにアクセスします。
4. 移行の宛先として使用する空のデータベースを作成します。または、デフォルトのデータベース `edb_admin` を使用できます。

既存のデータベースを使用する

または、既存の HM 管理データベースを移行の宛先として使用できます。既存のデータベースクラスターを移行の宛先として使用する前に、移行するテーブルと宛先で生成された負荷が既存のワークロードを干渉しないことを確認してください。

既存の HM 管理データベースを宛先として使用する場合は、次の手順に進みます。

Preparing your Oracle source for data migration

!!!note この構成は、ハイブリッドマネージャー HM 機能がデータをレプリケートできるため、データの移行に必要です。

- – title:Oracle ソースデータベースの準備 navTitle:Oracle ソースの準備

移行のために Oracle ソースを準備するには、 sysdba 権限が必要です。

Oracle ソースデータベースを次のように構成します。

- ステップ *1* アーカイブログモードを有効にする 。
- Enable supplemental logging for the database and table of interest 。
- ステップ *3* 移行ユーザーのテーブルスペースを準備する 。
- Create migration user and grant necessary privileges to carry out the data migration 。
- Grant SELECT on source tables 。
- Optionally, for better performance, ensure adequate redo log space is available 。
- Validate the configuration 。

前提条件

データベースの構成を変更するには、sysdba 特権で Oracle ソースデータベースに接続する必要があります。次の例では、SQL*Plus を使用してこの接続を確立します。

ステップ 1 アーカイブログモードを有効にする

Debezium が Oracle のすべてのデータベースの変更をキャプチャするには、データベースをアーカイブログモードで構成する必要があります。Debezium はデータベースの REDO ログを読んで変更を追跡するため、これは重要です。

アーカイブログモードでは、Oracle は上書きされる前に各 REDO ログファイルのコピーを保存します。これにより、すべてのトランザクションの継続的で完全な履歴が保証され、Debezium はギャップのない変更データの信頼できるストリームを維持する必要があります。

sysdba として Oracle コンテナデータベース CDB にログインします。または、非 CDB アーキテクチャを使用している場合は、sysdba として非 CDB データベースにログインします。

```
sqlplus <DBA_USER>/<PASSWD>@<HOST>:<PORT>/<DB_IDENTIFIER> as sysdba
```

そこで

- <DBA_USER> は、sysdba 特権を持つ Oracle ユーザーです。
- <PASSWD> は、 DBA_USER のパスワードです。
- <HOST> は、Oracle データベースのホスト名です。
- <PORT> は、Oracle データベースのポートです。

- <DB_IDENTIFIER> は、CDB の場合、コンテナデータベース名たとえば CDB1 です。非 CDB の場合、これはサービス名推奨または SID たとえば ORCL のいずれかです。

次の SQL コマンドを実行して、アーカイブログモードが有効になっているかどうかを確認します。

```
archive log list;
```

返されたコンテンツは、データベースモードを示します。

```
- - below output indicates archive log mode is enabled
Database log mode   Archive Mode -- (or "Archive Log Mode" in new versions of oracle)
- - or below output indicates archive log mode is not enabled
Database log mode   No Archive Mode
```

アーカイブログモードを既に有効にしている場合、この手順をスキップできます。そうでない場合は、現在の SQL セッションからログアウトし、次のコマンドを実行して有効にします。

SQL セッションに接続します。

```
ORACLE_SID=<DB_IDENTIFIER> sqlplus /nolog
```

そこで

- <DB_IDENTIFIER> は、CDB の場合、コンテナデータベース名 CDB1 などです。非 CDB の場合、これはサービス名推奨または SID ORCL などです。

次の SQL コマンドを実行します。

```
CONNECT <DBA_USER>/<PASSWORD> AS sysdba;
- - Set the location and size for Fast Recovery Area (FRA)
ALTER SYSTEM SET db_recovery_file_dest_size = <RECOVERY_FILE_DEST_SIZE>;
ALTER SYSTEM SET db_recovery_file_dest = <RECOVERY_FILE_DEST> scope=spfile;
- - Restart the database in the mount phase to enable archiving
SHUTDOWN IMMEDIATE;
STARTUP MOUNT
- - Enable archive log mode and open the database for use
ALTER DATABASE archivelog;
ALTER DATABASE open;
- - Verify the change was successful
archive log list;
exit;
```

そこで

- <DBA_USER> は、sysdba 特権を持つ Oracle ユーザーです。
- <PASSWD> は、 DBA_USER のパスワードです。
- <RECOVERY_FILE_DEST> は、Oracle REDO ログが保存される Oracle 高速リカバリ領域 FRA のディレクトリパスです。

- <RECOVERY_FILE_DEST_SIZE> は、FRA に割り当てられた領域の量です。たとえば、100 ギガバイトの場合は 100G です。

> **注釈**
>
> *db_recovery_file_dest* および `db_recovery_file_dest_size` の設定は、Debezium が読み取るアーカイブログに必要な保存場所である Oracle の FRA を構成します。

これらのパラメーターは、データベースの安定性にとって重要です。運用環境に設定する前に、必ず DBA に相談してください。

ステップ 2 データベースと対象のテーブルのサプリメンタルログを有効にする

サプリメンタルロギングとは、 *before* 状態などの Oracle REDO ログの追加情報のキャプチャを指します。この追加の REDO ログ情報は、EDB DMS のようなログベースのアプリケーションが変更イベントをキャプチャするために必要です。詳細については、Oracle の Supplemental Logging documentation を参照してください。

データの移行に Debezium を使用するには、データベースレベルとテーブルレベルの両方でサプリメンタルログを有効にする必要があります。これは、Debezium が変更データを正しくキャプチャするために重要です。

- **データベースレベルのサプリメンタルログ**は、変更データキャプチャ CDC の基礎です。これにより、すべての変更メッセージに追跡番号が含まれます。通常、これは変更された行の主キーです。これにより、どのレコードが変更されたかを一意に識別できます。これがないと、Debezium は変更イベントをソース行にリンクできません。
- **テーブルレベルのサプリメンタルログ**は、詳細情報を提供します。これにより、変更メッセージには行全体の変更前イメージ、つまり、変更が発生する前のすべての列の完全なコピーが含まれます。Debezium は変更前と後の状態を比較できるため、これは何が変更されたかを正確に理解するのに特に役立ちます。これがないと、Debezium は更新された列のみをキャプチャし、変更の完全なコンテキストをキャプチャしません。

sysdba として Oracle CDB にログインします。または、非 CDB アーキテクチャを使用している場合は、sysdba として非 CDB データベースにログインします。

```
sqlplus <DBA_USER>/<PASSWD>@<HOST>:<PORT>/<DB_IDENTIFIER> as sysdba
```

そこで

- <DBA_USER> は、sysdba 特権を持つ Oracle ユーザーです。
- <PASSWD> は、 DBA_USER のパスワードです。
- <HOST> は、Oracle データベースのホスト名です。
- <PORT> は、Oracle データベースのポートです。
- <DB_IDENTIFIER> は、CDB の場合、コンテナデータベース名 CDB1 などです。非 CDB の場合、これはサービス名推奨または SID ORCL などです。

データベースレベルでサプリメンタルログを有効にします。

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;
```

CDB アーキテクチャを使用している場合、移行する PDB に切り替えます。非 CDB アーキテクチャを使用している場合、この手順をスキップできます。

```
ALTER SESSION SET CONTAINER = <PDB_NAME>;
```

<PDB_NAME> は、移行する Oracle PDB の名前です。

移行するすべてのテーブルのサプリメンタルログを有効にします。

```
ALTER TABLE <SCHEMA_NAME>.<TABLE_NAME> ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
```

そこで

- <SCHEMA_NAME> は、テーブルが属しているスキーマです。
- <TABLE_NAME> は、移行するテーブルの名前です。

注釈

- 移行するテーブルでサプリメンタルログが有効になっていることを確認します。 Debezium の要件であるため、それがないと同期は機能しません。

- Oracle の場合、 **スキーマ** と **ユーザー**という用語は同じ意味で使用されます。データ移行の SQL コマンドを実行するときは、エラーを回避するために、必ずスキーマ名を正しいユーザーアカウントに設定してください。
- Oracle XE の互換性制限については、 limitations を参照してください。

ステップ 3 移行ユーザーのテーブルスペースを準備する

データの移行を実行するには、DMS とその基になる Debezium ライブラリの両方が専用のユーザーが必要です。このユーザーを作成する前に、必要なテーブルスペースを設定します。

CDB アーキテクチャの場合、2 つのテーブルスペースが必要です。

- **CDB レベルのテーブルスペース**このテーブルスペースは、Debezium の内部プロセス用です。これは、ログマイニングおよびコンテナデータベースレベルで実行する必要があるその他の管理操作に使用されます。
- **PDB レベルのテーブルスペース**これは、プラガブルデータベースに固有の Debezium 操作用です。その PDB 内でキャプチャしている個々のテーブルに関連するプロセスを処理します。

非 CDB アーキテクチャの場合、必要な表領域は 1 つだけです。コンテナ/PDB の分離がないため、この単一のテーブルスペースは、ログマイニングとテーブル操作の両方を組み合わせます。

sysdba として Oracle CDB にログインします。または、非 CDB アーキテクチャを使用している場合は、sysdba として非 CDB データベースにログインします。

```
sqlplus <DBA_USER>/<PASSWD>@<HOST>:<PORT>/<DB_IDENTIFIER> as sysdba
```

そこで

- <DBA_USER> は、sysdba 特権を持つ Oracle ユーザーです。
- <PASSWD> は、 DBA_USER のパスワードです。
- <HOST> は、Oracle データベースのホスト名です。
- <PORT> は、Oracle データベースのポートです。
- <DB_IDENTIFIER> は、CDB の場合、コンテナデータベース名 CDB1 などです。非 CDB の場合、これはサービス名推奨または SID ORCL などです。

SQL コマンドを使用してテーブルスペースを作成します。CDB アーキテクチャの場合、この表領域は CDB 用ですが、非 CDB アーキテクチャの場合、これは通常のデータベース用です。

```
CREATE TABLESPACE <TABLESPACE_NAME> DATAFILE <TABLESPACE_PATH>.dbf SIZE 25M REUSE␣
↪AUTOEXTEND ON MAXSIZE UNLIMITED;
```

そこで

- <TABLESPACE_NAME> は、テーブルスペースの名前です。ここでは、有効な名前を使用できます。
- <TABLESPACE_PATH>.dbf は、このテーブルスペースのファイルパスです。常に DBA に相談して、環境に合わせて正しく適切なファイルパスを決定します。

次に、CDB アーキテクチャを使用している場合は、移行するプラガブルデータベース PDB に切り替えます。非 CDB Oracle データベースを使用している場合、この手順をスキップできます。

```
ALTER SESSION SET CONTAINER = <PDB_NAME>;
```

<PDB_NAME> は、移行する Oracle の PDB の名前です。PDB のテーブルスペースを作成します。

```
CREATE TABLESPACE <TABLESPACE_NAME> DATAFILE <PDB_TABLESPACE_PATH>.dbf SIZE 25M REUSE␣
↪AUTOEXTEND ON MAXSIZE UNLIMITED;
  -- 25M is a minimal example size. Production systems typically require larger␣
↪initial allocations (e.g., 512M+).
  -- Consult your DBA for organization-specific sizing policies.
- - continue adding more datafiles if necessary
- - ALTER TABLESPACE <TABLESPACE_NAME> ADD DATAFILE <PDB_TABLESPACE_PATH2>.dbf SIZE␣
↪25M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
```

そこで

- <TABLESPACE_NAME> は、PDB テーブルスペースの名前です。一貫性を保つために、CDB 表領域に使用したのと同じ名前を使用できます。
- <PDB_TABLESPACE_PATH>.dbf は、テーブルスペースのファイルパスです。常に DBA に相談して、ご使用の環境の正しいパスを決定します。

ステップ 4 移行ユーザーを作成し、特権を付与する

DMS には、専用の移行ユーザー CDB の **共通ユーザー**、非 CDB の通常のユーザーが必要です。

sysdba として Oracle CDB にログインします。または、非 CDB アーキテクチャを使用している場合は、sysdba として非 CDB データベースにログインします。

```
sqlplus <DBA_USER>/<PASSWD>@<HOST>:<PORT>/<DB_IDENTIFIER> as sysdba
```

どこで

- <DBA_USER> は、sysdba 特権を持つ Oracle ユーザーです。
- <PASSWD> は、 DBA_USER のパスワードです。
- <HOST> は、Oracle データベースのホスト名です。
- <PORT> は、Oracle データベースのポートです。
- <DB_IDENTIFIER> は、CDB の場合、コンテナデータベース名 CDB1 などです。非 CDB の場合、これはサービス名推奨または SID たとえば、ORCL のいずれかです

次に、移行専用の Oracle ユーザーを作成します。作成方法とその権限は、CDB または非 CDB データベースを使用しているかどうかによって異なります。

!!!note HM エージェントでのスキーマ抽出と評価のための専用の Oracle ユーザーを以前に作成した場合、ここで同じユーザーを再利用できます。その場合、同じ方法で構成され、次の権限があることを確認します。

CDB データベース

CDB アーキテクチャを使用している場合、次の SQL を 1 つずつ実行して移行ユーザー共通のユーザーである必要があります。必要な権限を付与します。

```
CREATE USER <MIGRATION_USER> IDENTIFIED BY <MIGRATION_USER_PASSWORD>
    DEFAULT TABLESPACE <TABLESPACE_NAME>
    QUOTA UNLIMITED ON <TABLESPACE_NAME>
    CONTAINER=ALL;
GRANT CREATE SESSION TO <MIGRATION_USER> CONTAINER=ALL;
GRANT SET CONTAINER TO <MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ON V_$DATABASE to <MIGRATION_USER> CONTAINER=ALL;
GRANT FLASHBACK ANY TABLE TO <MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ANY TABLE TO <MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT_CATALOG_ROLE TO <MIGRATION_USER> CONTAINER=ALL;
GRANT EXECUTE_CATALOG_ROLE TO <MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ANY TRANSACTION TO <MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ANY DICTIONARY TO <MIGRATION_USER> CONTAINER=ALL;
GRANT LOGMINING TO <MIGRATION_USER> CONTAINER=ALL;
GRANT CREATE TABLE TO <MIGRATION_USER> CONTAINER=ALL;
GRANT LOCK ANY TABLE TO <MIGRATION_USER> CONTAINER=ALL;
GRANT CREATE SEQUENCE TO <MIGRATION_USER> CONTAINER=ALL;
GRANT EXECUTE ON DBMS_LOGMNR TO <MIGRATION_USER> CONTAINER=ALL;
```

(次のページに続く)

(前のページからの続き)

```
GRANT EXECUTE ON DBMS_LOGMNR_D TO <MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ON V_$LOGMNR_LOGS TO <MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ON V_$LOGMNR_CONTENTS TO <MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ON V_$LOGFILE TO <MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ON V_$ARCHIVED_LOG TO <MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ON V_$ARCHIVE_DEST_STATUS TO <MIGRATION_USER> CONTAINER=ALL;
GRANT SELECT ON V_$TRANSACTION TO <MIGRATION_USER> CONTAINER=ALL;
```

そこで

- <MIGRATION_USER> は、DMS 用に作成する共通ユーザーの名前です。
- <MIGRATION_USER_PASSWORD> は、新しい移行ユーザーのパスワードです。
- <TABLESPACE_NAME> は、前に作成した CDB 表領域の名前です。簡単にするために、CDB と PDB 表領域の両方に同じ名前を使用することをお勧めします。

前の手順で CDB および PDB テーブルスペースに別の名前を使用した場合、PDB にログインし、次のコマンドを実行して、ユーザーのデフォルトのテーブルスペースを設定する必要があります。

```
ALTER USER <MIGRATION_USER> DEFAULT TABLESPACE <PDB_TABLESPACE_NAME>
```

非 CDB データベース

非 CDB データベースを使用している場合は、代わりに次の SQL コマンドを使用します。

```
CREATE USER <MIGRATION_USER> IDENTIFIED BY <MIGRATION_USER_PASSWORD>
    DEFAULT TABLESPACE <TABLESPACE_NAME>
    QUOTA UNLIMITED ON <TABLESPACE_NAME>;
GRANT CREATE SESSION TO <MIGRATION_USER>;
GRANT SELECT ON V_$DATABASE to <MIGRATION_USER>;
GRANT FLASHBACK ANY TABLE TO <MIGRATION_USER>;
GRANT SELECT ANY TABLE TO <MIGRATION_USER>;
GRANT SELECT_CATALOG_ROLE TO <MIGRATION_USER>;
GRANT EXECUTE_CATALOG_ROLE TO <MIGRATION_USER>;
GRANT SELECT ANY TRANSACTION TO <MIGRATION_USER>;
GRANT SELECT ANY DICTIONARY TO <MIGRATION_USER>;
GRANT LOGMINING TO <MIGRATION_USER>;
GRANT CREATE TABLE TO <MIGRATION_USER>;
GRANT LOCK ANY TABLE TO <MIGRATION_USER>;
GRANT CREATE SEQUENCE TO <MIGRATION_USER>;
GRANT EXECUTE ON DBMS_LOGMNR TO <MIGRATION_USER>;
GRANT EXECUTE ON DBMS_LOGMNR_D TO <MIGRATION_USER>;
GRANT SELECT ON V_$LOGMNR_LOGS TO <MIGRATION_USER>;
GRANT SELECT ON V_$LOGMNR_CONTENTS TO <MIGRATION_USER>;
GRANT SELECT ON V_$LOGFILE TO <MIGRATION_USER>;
GRANT SELECT ON V_$ARCHIVED_LOG TO <MIGRATION_USER>;
```

(次のページに続く)

(前のページからの続き)

```
GRANT SELECT ON V_$ARCHIVE_DEST_STATUS TO <MIGRATION_USER>;
GRANT SELECT ON V_$TRANSACTION TO <MIGRATION_USER>;
```

ここで、 - <MIGRATION_USER> は、CDC 移行テーブルアクセス用に作成するユーザーの名前です。 - <MIGRATION_USER_PASSWORD> は、移行ユーザーのパスワードです。 - <TABLESPACE_NAME> は、<MIGRATION_USER> のテーブルスペースです。

ステップ 5 ソーステーブルに SELECT を付与する

sysdba として PDB にログインします。または、非 CDB アーキテクチャを使用している場合は、sysdba として非 CDB データベースにログインします。

```
sqlplus <DBA_USER>/<PASSWD>@<HOST>:<PORT>/<DB_IDENTIFIER> as sysdba
```

そこで

- <DBA_USER> は、sysdba としてログインできる Oracle ユーザーです。
- <PASSWD> は、<DBA_USER> のパスワードです。
- <HOST> は、Oracle データベースのホスト名です。
- <PORT> は、Oracle データベースポートです。
- <DB_IDENTIFIER> は、CDB の場合、プラガブルデータベース名 PDB1 などです。非 CDB の場合、これはサービス名推奨または SID ORCL などです。

移行するテーブルごとに、作成した移行ユーザーに SELECT 権限を付与する必要があります。

```
GRANT SELECT ON <SCHEMA_NAME>.<TABLE_NAME> TO <MIGRATION_USER>
```

そこで

- <SCHEMA_NAME> は、移行するテーブルを含むスキーマの名前です。
- <TABLE_NAME> は、移行する個々のテーブルの名前であり、大文字と小文字が区別されます。
- <MIGRATION_USER> は、CDC 移行の目的でテーブルへの SELECT アクセスを付与されているユーザーの名前です。

> **注釈**
>
> Oracle の場合、*スキーマ*と*ユーザー*という用語は互換性があります。移行エラーを防ぐには、必ず正しいスキーマ名を使用してください。

オプショナル ステップ 6 パフォーマンスを向上させるために十分な REDO ログ領域を利用できることを確認する

データ移行プロセスは、2 つの主なフェーズで構成されています。データの一貫したスナップショットと、それに続く継続的な変更ストリーム。この変更ストリームは、LogMiner および Oracle データベースの REDO ログを利用しています。

REDO ログの変更の有効期間は制限されています。これは、ログのサイズと数、およびデータベースの変更のスループットによって決まります。REDO ログが小さすぎると、頻繁なログスイッチが発生する可能性があり、移行パフォーマンスに悪影響を与える可能性があります。

データベースの REDO ログの状態を確認するには、sysdba として Oracle CDB にログインします。または、非 CDB アーキテクチャを使用している場合は、sysdba として非 CDB データベースにログインします。

```
sqlplus <DBA_USER>/<PASSWD>@<HOST>:<PORT>/<DB_IDENTIFIER> as sysdba
```

そこで

- <DBA_USER> は、sysdba 特権を持つ Oracle ユーザーです。
- <PASSWD> は、 DBA_USER のパスワードです。
- <HOST> は、Oracle データベースのホスト名です。
- <PORT> は、Oracle データベースのポートです。
- <DB_IDENTIFIER> は、CDB の場合、コンテナデータベース名 CDB1 などです。非 CDB の場合、これはサービス名推奨または SID ORCL などです。

現在のログファイル構成を表示するには、次の SQL クエリーを実行します。

```
SELECT GROUP#, TYPE, MEMBER FROM V_$LOGFILE;
__OUTPUT__
    GROUP# TYPE    MEMBER
- --------- ------- --------------------------------------------------
         1 ONLINE  /opt/oracle/oradata/ORCLCDB/redo03.log
         2 ONLINE  /opt/oracle/oradata/ORCLCDB/redo01.log
         3 ONLINE  /opt/oracle/oradata/ORCLCDB/redo04.log
```

```
SELECT GROUP#, ARCHIVED, BYTES/1024/1024 MB, STATUS FROM V_$LOG;
__OUTPUT__
    GROUP# ARC  MB STATUS
- --------- --- --- ----------------
         1 YES 2000 INACTIVE
         3 YES 2000 INACTIVE
         3 NO  2000 CURRENT
```

たとえば、出力には、それぞれのサイズが 2000MB の 3 つのロググループが表示される場合がありますが、これは多くの運用環境にとって小さすぎる場合があります。

必要に応じて、次のようなコマンドを使用して REDO ログを調整できます。

```
ALTER DATABASE ADD LOGFILE GROUP 4 (/opt/oracle/oradata/ORCLCDB/redo04.log) SIZE 8G;
ALTER DATABASE ADD LOGFILE GROUP 5 (/opt/oracle/oradata/ORCLCDB/redo05.log) SIZE 8G;
ALTER DATABASE ADD LOGFILE GROUP 6 (/opt/oracle/oradata/ORCLCDB/redo06.log) SIZE 8G;
ALTER DATABASE ADD LOGFILE GROUP 7 (/opt/oracle/oradata/ORCLCDB/redo07.log) SIZE 8G;
ALTER SYSTEM ARCHIVE LOG CURRENT;
```

(次のページに続く)

(前のページからの続き)

```
ALTER SYSTEM CHECKPOINT;
ALTER SYSTEM ARCHIVE LOG CURRENT;
ALTER SYSTEM CHECKPOINT;
ALTER SYSTEM ARCHIVE LOG CURRENT;
ALTER SYSTEM CHECKPOINT;
ALTER DATABASE DROP LOGFILE GROUP 1;
ALTER DATABASE DROP LOGFILE GROUP 2;
ALTER DATABASE DROP LOGFILE GROUP 3;
```

これらのコマンドは、それぞれが単一の REDO ログファイルを含む 4 つの 8GB REDO ロググループを作成します。

> **注釈**
>
> 常に DBA に相談して、本番環境に適切な REDO ログサイズを決定します。

ステップ 7 構成を検証する

ソースデータベースを構成した後、Oracle 構成を検証します。このために、DMS エージェントにパッケージされている構成検証スクリプトを使用できます。

検証の前提条件

1. DMS エージェントを実行しているマシンに Java/OpenJDK 17 以降がインストールされていることを確認します。

2. Install the EDB DMS Agent 。

3. `/opt/cdcagent/` の EDB DMS Agent フォルダーに移動します。

構成検証スクリプトを実行する

EDB DMS Reader インストール `cdcreader` としてパッケージ化には、Oracle 構成を検証し、問題の特定に役立つヘルパースクリプトが付属しています。データベースを構成した後、スクリプトを実行して、すべてのチェックに合格することを確認することをお勧めします。

1. 移行する必要があるすべてのテーブルの名前を<schema_name>.<table_name>形式で含む配列を作成します。例

```
arr=(test1.table1 test1.table2 test1.table3 test1.table4)
```

1. 必要なパラメーターを使用してスクリプトを実行します。例

```
ODB_SYSDBA_USERNAME=sys ODB_SYSDBA_PASSWORD=password ODB_HOST=localhost ODB_PORT=5432␣
↪ORACLE_SID=postgres ODB_DBZ_USERNAME=debezium ODB_DBZ_PASSWORD=dbz LOG_MINER_TBS_
↪NAME=LOGMINER_TBS ./oracleConfigValidation.sh "${arr[@]}"
```

次のような応答が得られます。

```
 ***  [Transporter] - Validate system user connectivity
[Pass] Provide username and password of system user can connect to database.

  ***  [Transporter] - Validate debezium user connectivity
[Pass] Provide username and password of debezium user can connect to database.

  ***  [Transporter] - Validate Archive Log mode
[Pass] Database log mode is Archive Mode.

  ***  [Transporter] - Validate fast recovery area parameters
[Pass] Recovery area path was set.
[Pass] Recovery area size limit meets requirements.
[Pass] Recovery area remaining size meets requirements.

  ***  [Transporter] - Validate if database supplemental logging is enabled
[Pass] Supplemental logging check for database is passed.

  ***  [Transporter] - Validate if table supplemental logging is enabled
[Pass] All source tables enable supplemental logging.

  ***  [Transporter] - Validate redo log count&size
[Pass] Redo log count meets the requirements.
[Pass] Redo log size meets the requirements.
[Pass] Redo logs are all active.

  ***  [Transporter] - Validate if tablespace for migration with Limited Privileges␣
↪were created
[Pass] LOGMINER_TBS tablespace has enough datafiles.
[Pass] LogMiner DataFile Size meet the requirements.
[Pass] LogMiner DataFile Max Size meet the requirements.
[Pass] LogMiner DataFile are auto-extendable.

  ***  [Transporter] - Validate if user for migration with Limited Privileges were␣
↪created
Oracle version is 19, which is 12c or newer.
[Pass] This user has all sys privileges that migration needs.
[Pass] This user has all role privileges that migration needs.
[Pass] This user has all select privilege on specific tables that migration needs.
[Pass] This user has all execute privilege on specific tables that migration needs.

  ***  [Transporter] - Validate if user for migration with LOGMINER_TBS tablespace
[Pass] User for migration use LOGMINER_TBS tablespace.
```

データベースは CDC 移行の準備が整いました。

> **注釈**
>
> ここで推奨されているようにソースデータベース設定を変更することにより、*[Failed]* または`[Suggestion]`ステータスに対処します。 *[Failed]* チェックは、DMS がデータ移行を実行する前に解決する必要がある問題をブロックしています。 *[Suggestion]* チェックでは移行が許可されますが、対処しないでおくとパフォーマンスに悪影響を与える可能性があります。

SSL 構成

SSL 接続を受け入れるようにソースデータベースサーバーが構成されていることを確認して、EDB DMS Reader が接続できるようにします。この構成を有効にするには、サーバー証明書とサーバープライベートキーを作成する必要があります。OpenSSL のようなツールを使用して、キーを作成できます。

データベースは CDC 移行の準備が整いました。

詳細情報

-

Debezium Oracle Connector ドキュメントを参照してください。

- Oracle の Enabling the Fast Recovery Area を参照してください。

Understanding schema handling

title: スキーマの準備とインポート navTitle: スキーマの準備 deepToC:true EDB Data Migration Service EDB DMS を使用して `cdcagent` を使用するデータ移行を構成する前に、スキーマを準備して宛先データベースにインポートする必要があります。

EDB DMS はデータのみを移行するため、移行が発生する前に宛先データベースにスキーマをインポートすることが重要です。移行先データベースには、移行のためにスキーマを配置してデータを入力する必要があります。

データの移行が発生する前に、スキーマの制約の一部を含める必要があります。その他は、データの移行が完了した後に適用する必要があります。このアプローチにより、パフォーマンスを低下させることなく移行できます。

スキーマの整合性とパフォーマンスの考慮事項

宛先データベースの制約、トリガー、および WAL ロギングの存在は、データ移行のパフォーマンスに影響を与える可能性があります。このアプローチが可能な場合、スキーマ制約の 2 段階インポートをお勧めします。

主キーと一意性制約

EDB DMS は、主キー/一意制約を活用して、データを宛先データベースに移行するときに 1 回だけ配信を提供します。したがって、データの移行を開始する前に実行するスキーマインポートに主キー/一意制約を含める必要があります。他の種類の制約をスキーマのインポートから除外します。

主キー/一意制約のないテーブルの行の場合、少なくとも 1 回の配信のみを実現できます。データ移行の検証中に重複排除を実行できます。

!!!note

Not null 制約は、宛先サーバーのパフォーマンスに重大な影響を与えず、スキーマのインポートに含めることもできます。

外部キー、チェック、および除外コンストレイン

EDB DMS は、宛先データベースクラスターに対して変更イベントを並列適用できます。ただし、一部の制約タイプを移行すると、移行のパフォーマンスに悪影響を与える可能性があります。この種の制約は、一貫した参照統合ソースデータベースからの飛行中のデータ移行のコンテキストで、不必要な CPU とメモリの使用率をもたらします。

したがって、移行が完了した後にのみ、宛先データベースに外部キー、チェック、および除外制約を適用することをお勧めします。

Migrating Oracle schemas with Migration Portal

title:Oracle スキーマの準備とインポート navTitle:Oracle スキーマの準備 deepToC:true

Oracle データベースのスキーマ処理

いくつかの異なる方法を使用して、スキーマを準備し、サポートされていない制約を除外し、宛先データベースにインポートできます。以下は、移行ポータルを使用して Oracle のスキーマをエクスポートおよびインポートする方法に関するガイドラインです。

DDL を移行して Postgres スキーマをインポートするための他の有効なルートは次のとおりです。

- 宛先データベースにスキーマを手動で作成する
- pg_dump を実行してスキーマを取得し、

pg_restore を実行してそれらを復元する

-

pgAdmin または psql を使用する

エクスポート

移行ポータルの協調的な project-scoped workspace を使用して、スキーマを変換および移行します。Hybrid ManagerHM は、HM コンソールを使用して移行ポータルプロジェクトを作成できる統合を提供します。そのプロジェクトでは、データベースの移行の実行中に、 HM agent によってプリロードされたソースデータベーススキーマを簡単に追加できます。

前提条件

- Hybrid Manager を使用して

registered an Oracle database します。

- オプショナル HM 管理者

enabled and configured the AI Copilot with the AI/LLM of your organization
。この構成は、移行ポータルでのスキーマの非互換性を解決するのに役立ちます。

Migration Portal プロジェクトの作成

ハイブリッドマネージャーから移行ポータルプロジェクトを作成できます。

1. **Estate** ページで、 **Migrations** タブを選択します。
2. データベースの詳細ビューを開くには、移行するデータベースの名前を選択します。
3. データベースの詳細ビューで、 **Schemas** タブを選択します。
4. 移行するスキーマのいずれかの横にある **New MP Project** またはプラス記号を選択します。

!!!tip **Dependencies** 列には、現在のスキーマが依存する他のスキーマが表示されます。スキーマとそのすべての依存関係を単一の移行ポータルプロジェクトにグループ化して、参照されるすべてのオブジェクトが宛先データベースに存在することを確認することをお勧めします。

1. ポップアップウィンドウで、スキーマの評価、変換、および移行をホストする移行ポータルプロジェクトの詳細を入力します。**続行** を選択します。
2. この移行ポータルプロジェクトに追加するスキーマをすべて選択します。HM エージェントが正常に取り込んだスキーマと、別の移行ポータルプロジェクトの一部になっていないスキーマのみを選択できます。

!!!note 既存のプロジェクトにスキーマを後で追加することはできません。代わりに、別のプロジェクトを使用して、他のスキーマを同じ宛先データベースに移行します。

1. **Create New Project** を選択します。

- 移行ポータルプロジェクトは、分析のためにロードされたソーススキーマを使用してバックグラウンドで作成されます。

HM がスキーマのロードと移行ポータルでのプロジェクトの作成が終了すると、プロジェクトの名前のリンクが追加されます。リンクを選択して Migration Portal を開きます。

選択したスキーマは、移行ポータルによってアップロードされ、評価されました。AI Copilot または QuickHelp を使用して、非互換性の解決に役立ちます。

インポート

標準スキーマのインポート手順

スキーマ、テーブル、および列の名前を変更しない予定がある場合は、**オンライン移行**オプションを使用して宛先データベースに直接接続し、スキーマを移行します。

1. すべての互換性問題を解決したら、[**Migrate to**] を選択します。
2. **移行** オプションを選択します推奨。
3. 移行するスキーマを選択しますが、データベースの下のオブジェクトのリストから**コンストレイン**を無効にします。 **Next** を選択します。

!!!note

スキーマを宛先データベースにインポートする前に、SQL 形式の DDL から外部キー、チェック、および除外制約を除外していることを確認します。

1. **Connect to your cluster** ページで、宛先として使用するデータベースクラスターのホスト名を入力します。

!!!ヒント HM 管理のデータベースクラスターに移行する場合は、Clusters、cluser 名、および Connect**を選択することにより、読み取り/書き込みホストおよびその他の接続データを見つけることができます。 .. ::**

!!!warning HM の分散高可用性および高度な高可用性 PGD 宛先 この警告は、移行ポータルのオンライン移行オプションを使用して、 EDB を実行している HM 管理の分散高可用性または高度な高可用性データベースクラスターにスキーマを移行する場合に適用されます。 EDB Postgres Advanced Server。 **Target Database**と**Maintenance Database**フィールドは両方とも、デフォルトの `bdrdb` データベースを指している必要があります。これにより、移行されたスキーマがクラスター全体でレプリケートされるプライマリ PGD データベースに作成されます。 .. :: 1. ** Next**を選択し、スキーマが正常にインポートされるまで待ちます。

1. ターゲットデータベースクラスターに接続し、スキーマが移行されたことを確認します。

!!!note

移行プロセスの後半で、移行が正常に実行された後、宛先データベースに除外された外部キー、チェック、および除外制約を再適用する必要があります。したがって、除外されたコンストレインを追跡します。ドキュメントには、それらを再適用する時期と方法についてのガイドがあります。

オブジェクトの名前を変更するスキーマインポートプロシージャオプショナル

スキーマ、テーブル、または列を宛先の別の名前にマップする予定がある場合は、**オフライン移行**オプションを使用します。これにより、編集する DDL ファイルをダウンロードできます。

1. すべての互換性問題を解決したら、[**Migrate to**] を選択します。
2. **オフライン移行** オプションを選択します。
3. 移行するスキーマを選択しますが、データベースの下のオブジェクトのリストから**コンストレイン**を無効にします。 **Next** を選択します。

!!!note

スキーマを宛先データベースにインポートする前に、SQL 形式の DDL から外部キー、チェック、および除外制約を除外していることを確認します。

1. **Download SQL file** を選択します。
2. SQL スクリプトが生成された後、宛先に適用する前に、スクリプト DDL でオブジェクト名を直接編集します。スクリプト内の名前が変更されたオブジェクトの構造とデータ型がソースと完全に一致することを確認します。詳細については、

Mapping schema, tables and columns during a migration を参照してください。

DDL スクリプトのオブジェクト名を手動で更新する場合は、更新されたオブジェクトを参照する他のオブジェクトの名前も更新してください。たとえば、テーブルの名前を変更する場合は、元のテーブル名を参照するビューまたはコンストレインも更新します。

1. 宛先データベースに接続し、変更した DDL ファイルを適用します。

!!!note

移行プロセスの後半で、移行が正常に実行された後、宛先データベースに除外された外部キー、チェック、および除外制約を再適用する必要があります。したがって、除外されたコンストレインを追跡します。ドキュメントには、それらを再適用する時期と方法についてのガイドがあります。

Creating Hybrid Manager project credentials for the migration

title: ハイブリッドマネージャー移行プロジェクトの資格情報の作成 navTitle: プロジェクトの資格情報の作成 deepToC:true データの移行を実行するには、資格情報が必要です。移行のソースまたは宛先をハイブリッドマネージャー HM に追加する前に、移行の資格情報を作成する必要があります。次に、それらを使用して、EDB DMS エージェントが HM に接続できるようにします。

HM project ごとに 1 セットの資格情報が必要です。たとえば、すべての移行に単一の HM プロジェクトを使用している場合、1 セットの資格情報のみが必要です。

1. HM コンソールで、移行アクティビティに使用しているプロジェクトを選択します。
2. プロジェクトで、 **Migrate > Credentials** を選択します。

Credentials ビューには、プロジェクト内のすべての資格情報が表形式で表示されます。各資格情報のエントリには、その名前、ステータス、作成日、および有効期限が表示されます。

3. **Create Migration Credential** を選択し、資格情報の名前を入力して、**Create Migration Credential** を選択します。
4. 資格情報が生成されたら、 **Download Credential** を選択します。

!!!note

資格情報をダウンロードできるのはこのときのみです。今すぐダウンロードせず、資格情報が必要な場合、このセットを無効にして、新しいセットを作成する必要があります。

5. ダウンロードした zip ファイルを解凍し、生成されたフォルダーを EDB DMS エージェントがインストールされているホストにコピーします。

解凍された資格情報フォルダーには、次のファイルが含まれています。

- `int.crt` - `int.truststore.p12` - `client.keystore.p12` - `client-key.pem` - `client-cert.pem`

Configuring and running the EDB DMS agent in reader mode

title:Oracle のリーダーモードでの EDB DMS エージェントの構成と実行 ソース navTitle:Oracle の DMS エージェントの構成 deepToC:true データ移行を実行するようにリーダーモードで DMS エージェントを構成および実行します。

前提条件

1. DMS エージェントを実行しているマシンに Java/OpenJDK 17 以降がインストールされていることを確認します。

2. Install the EDB DMS agent 。

3. `/opt/cdcagent/run-cdcagent.sh` にある EDB DMS エージェントを書き込み権限で開きます。

Oracle からの移行のための EDB DMS エージェントの構成

環境に応じてスクリプトの変数を設定し、編集した行のコメントを外します。詳細については、parameters を参照してください。以下は、編集されていないスクリプトを示しています。

```
# !/bin/bash -e

#  run_cdcagent.sh

#

#  This script provides a convenient place to specify

#  environment variables used to configure the

#  EDB Data Migration Service Agent.

#

##########################################
#  DMS Agent General Configuration       #

##########################################

#  This ID is used to identify DMS Agent

#  and is specified by the user.

export DBCONFIG_ID=<source_name_of_choice>

#  Determine the run mode of the DMS Agent

#  The run mode can be set to either reader or writer

#  reader mode is used to read changes from a source database

#  writer mode is used to write changes to a target database

export RUN_MODE=reader

#  This is the DMS backend service used by the Agent

#  Consult your system administrators

export RW_SERVICE_HOST=https://${TRANSPORTER_RW_SERVICE_DOMAIN_NAME}/transporter
```

(次のページに続く)

(前のページからの続き)

```
########################################
#  DMS Agent TLS Configuration         #

########################################

#  You need to create migration credentials in EDB Postgres AI platform

#  and set the path of the credentials directory

export CREDENTIAL_DIRECTORY_PATH=$HOME/credentials

########################################
#  DMS Agent DB Configuration     #

########################################

#  A sample configuration to create a single postgres database connection:

export DBCONFIG_DATABASES_0__TYPE=ORACLE
export DBCONFIG_DATABASES_0__HOSTNAME=<host>
export DBCONFIG_DATABASES_0__PORT=<1521>

#  The CATALOG is the database name

export DBCONFIG_DATABASES_0__CATALOG=<source>
export DBCONFIG_DATABASES_0__USERNAME=<SYS>

#  The password env can be set without specifying it here

#  but the env structure looks like this

export DBCONFIG_DATABASES_0__PASSWORD=<password>

#  The resource_id is configured by the customer in the configuration file of the␣
↪beacon agent

#  need to make sure that the resource_id is set to the same value as in the beacon␣
↪agent

# export DBCONFIG_DATABASES_0__RESOURCEID=1234567890abcdef

#  You can increase the index to configure more than

#  one database for the DMS Agent
```

(次のページに続く)

(前のページからの続き)

```
# export DBCONFIG_DATABASES_1__TYPE=ORACLE
# export DBCONFIG_DATABASES_1__HOSTNAME=localhost
# export DBCONFIG_DATABASES_1__PORT=1521

#  The CATALOG is the database name

# export DBCONFIG_DATABASES_1__CATALOG=ORCLCDB/ORCLPDB1
# export DBCONFIG_DATABASES_1__USERNAME=oracle

#  The password env can be set without specifying it here

#  but the env structure looks like this

# export DBCONFIG_DATABASES_1__PASSWORD=password

#  The resource_id is configured by the customer in the configuration file of the␣
↪beacon agent

#  need to make sure that the resource_id is set to the same value as in the beacon␣
↪agent

# export DBCONFIG_DATABASES_1__RESOURCEID=1234567890abcdef

##########################################
#  Optional Parameters Below             #

##########################################

#  FIPS mode can be enabled by setting the FIPS_MODE env variable to true

# export FIPS_MODE=true

#  Configure logging

#  Loglevel for the entire Quarkus application and underlying components

#  WARNING: Enabling this loglevel is extremely verbose

# export QUARKUS_LOG_LEVEL=DEBUG

#  Loglevel for the EDB package

#  Preferred mode for debugging issues related to the logic of the agent

# export QUARKUS_LOG_CATEGORY__COM_ENTERPRISEDB__LEVEL=DEBUG
```

(次のページに続く)

(前のページからの続き)

```
cd "$(dirname "$0")"

echo "Starting in mode: $RUN_MODE"

case "$RUN_MODE" in
  reader)
    java -jar ./reader/quarkus-run.jar
    ;;
  writer)
    java -jar ./writer/quarkus-run.jar
    ;;
  *)
    echo "Unknown RUN_MODE: $RUN_MODE"
    exit 1
    ;;
esac
```

EDB DMS エージェントの実行

移行を開始します

```
cdcagent
```

Verifying your source appears in the HM console

ソース は、リーダーモードで EDB DMS エージェントを使用してハイブリッドマネージャー HM に接続するデータベースであるため、移行に使用できます。

ソースが EDB DMS エージェントによって認識され、移行に使用できることを確認するには

1. HM コンソールで、現在の移行に使用しているプロジェクトを選択します。
2. 左側のナビゲーションメニューから、 **Migrate > Sources** を選択します。

各ソースのビューには、データベース名、データベースタイプ Postgres または Oracle、cdcagent ID、現在のステータス、ソースタイプなどが表示されます。

詳細情報

HM コンソールのこのページで使用可能なデータの説明については、Sources を参照してください。

Creating a data migration with Hybrid Manager

title HM 管理の宛先への移行の作成 navTitle 移行の作成 EDB DMS エージェントリーダーモードで構成してハイブリッドマネージャー HM への接続を確立した後、HM コンソールで移行を作成します。移行は、EDB DMS の助けを借りて、ソースデータベースと宛先データベース間の同期を確立します。

1. HM コンソールで、移行を管理するプロジェクトを選択します。

2. プロジェクトで、 **移行>移行** を選択します。
3. [**移行**] ページで、[**Create New Migration**] を選択します。
4. **Create OLTP Migration** ページで、名前を移行に割り当てます。
5. 移行のソースを選択します。 EDB DMS エージェントの ID は、 **Source** メニューにリストされています。

6. 移行タイプを選択します**Snapshot** または**Snapshot + Streaming** 。

7. **Destination Type** で、移行用の HM 管理の宛先データベースクラスターとデータベースを選択します。次に、**Next** を選択します。
8. 移行するテーブルと列を選択します。必要に応じて、テーブルと列の名前を変更します。
9. **Create Migration** を選択します。

HM コンソールには、新しい移行が表示されます。

スナップショット+ストリーミング オプションを選択した場合、移行で**実行中** 状態が表示されるときに EDB DMS エージェントは常にデータをストリーミングしています。データの変更は、移行が実行されている限りソースから宛先データベースに複製されます。

!!!note

リーダーモードの EDB DMS エージェントは、データ変更をストリーミングします。DDL オブジェクトの変更はストリーミングされません。

Finalizing a streaming data migration

title: 移行のファイナライズストリーミング移行の場合 navTitle: 移行のファイナライズ移行を作成するときに**Snapshot + Streaming** オプションを選択した場合、EDB DMS エージェントは、移行を次のようにマークするまで、ソースデータベースで実行されたデータ更新を宛先データベースにストリーミングし続けます。完了しましたたとえば、カットオーバーを実行します。

Snapshot オプションを選択した場合、DMS は宛先へのスナップショット全体のコピーが終了した後、移行を完了としてマークするため、このページをスキップできます。

ストリーミングプロセスを停止します

このプロセスは、**Snapshot + Streaming** 移行にのみ適用されます。

データ移行ストリームを停止する最適な時期を注意深く選択することが重要です。ソースで新しい挿入が実行されていないことを確認し、最後の既知の挿入が宛先で同期されたことを確認します。

データが宛先データベースに存在することを確認した後、移行を完了としてマークすることにより、ストリームを停止できます。

1. Hybrid ManagerHM コンソールのプロジェクトで、 **移行>移行**を選択します。
2. 移行の横にあるチェックマークを選択して、移行が完了したことを示します。

または、リストから移行を選択して Migration details page を開きます。次に、 **Mark as Completed** を選択します。

これにより、ストリーミング手順が停止します。

Applying constraints after the migration is finalized

title: 制約の適用 EDB Data Migration Service EDB DMS を使用したデータ移行の開始時に、ソースデータベースのスキーマを準備およびインポートしました。次に、宛先データベースに接続し、スキーマとデータの移行から除外された制約を再適用します。

主キーと一意制約

主キーと一意制約の場合、宛先の Postgres データベースにテーブルと制約が既に作成されています。これにより、EDB DMS はそれらをソースオブジェクトにマッピングし、データを正常に移行できます。他に何もする必要はありません。

not null 制約をスキーマインポートに含めた場合、同じことが適用されます。

外部キー、チェック、および除外コンストレイン

スキーマの準備とインポート中に除外した外部キー、チェック、または除外制約を再適用できるようになりました。

データの完全性の確保

主キーまたは一意制約を持たないテーブルの行は、少なくとも 1 回配信で移行されたため、これらの行は重複である可能性があります。

重複排除は、検証の一部として実行できます。

検証

すべてのデータが移行されたことを確認するには、ソースデータベースと宛先データベースを比較します。

LiveCompare を使用できます。

Migrate your self-managed Postgres database to an HM-managed database

これらは、自己管理の Postgres データベースから Hybrid ManagerHM によって管理されるデータベースクラスターへの移行を実行するために必要なすべての手順です。

ソースデータベースの準備

データ移行のソースデータベースの準備には、移行専用のユーザーを作成し、データベース構成を変更してレプリケーション機能を有効にすることが含まれます。

1. *Preparing your AWS RDS/Aurora PostgreSQL source database* 。

データベースの評価オプショナル

データベースに HM エージェントをインストールおよび構成して、Postgres データベースの可観測性と監視データを取得します。このオプションの手順は、スキーマ数と名前、テーブル数、オブジェクト数などの高レベルの移行メトリックも提供します。

1. *Configure the HM agent to assess your database* 。

宛先データベースを作成する

HM インスタンスにクラスターを作成して、宛先データベースをホストします。

1. *Creating your HM-managed destination database* 。

スキーマを移行する

非互換性を処理し、スキーマを転送します。

1. *Understanding schema handling* 。
2. *Migrating Postgres schemas with Migration Toolkit* 。

データを移行する

データ転送を実行し、最終的なクリーンアップ手順を完了します。

1. *Creating Hybrid Manager project credentials for the migration* 。
2. *Configuring and running the EDB DMS agent in reader mode* 。
3. **詳細については** 。
4. *Creating a data migration with Hybrid Manager* 。
5. *Finalizing a streaming data migration* ストリーミング移行の場合のみ。
6. *Applying constraints after the migration is finalized* 。

Applying constraints after the migration is finalized

title: 制約の適用 EDB Data Migration Service EDB DMS を使用したデータ移行の開始時に、ソースデータベースのスキーマを準備およびインポートしました。次に、宛先データベースに接続し、スキーマとデータの移行から除外された制約を再適用します。

主キーと一意制約

主キーと一意制約の場合、宛先の Postgres データベースにテーブルと制約が既に作成されています。これにより、EDB DMS はそれらをソースオブジェクトにマッピングし、データを正常に移行できます。他に何もする必要はありません。

not null 制約をスキーマインポートに含めた場合、同じことが適用されます。

外部キー、チェック、および除外コンストレイン

スキーマの準備とインポート中に除外した外部キー、チェック、または除外制約を再適用できるようになりました。

データの完全性の確保

主キーまたは一意制約を持たないテーブルの行は、少なくとも 1 回配信で移行されたため、これらの行は重複である可能性があります。

重複排除は、検証の一部として実行できます。

検証

すべてのデータが移行されたことを確認するには、ソースデータベースと宛先データベースを比較します。

LiveCompare を使用できます。

Configuring the HM agent to assess your database

Hybrid ManagerHM エージェントをインストールおよび構成して、Postgres データベースの可観測性と監視データを取得できます。このオプションの手順は、スキーマ数と名前、テーブル数、オブジェクト数などの高レベルのメトリックを提供し、移行計画に役立ちます。

!!!note 移行を実行するために HM エージェントの構成は必要なく、現在実際のスキーマオブジェクト DDL を取り込みません。

HM エージェントにデータベースを登録するには、以下を参照してください。

- Monitoring self-managed deployments
-

Monitoring Cloud Service Provider managed Postgres deployments

Configuring and running the EDB DMS agent in reader mode

title: Postgres のリーダーモードでの EDB DMS エージェントの構成と実行ソース navTitle: Postgres の DMS エージェントの構成 deepToC:true データ移行を実行するようにリーダーモードで DMS エージェントを構成および実行します。

前提条件

1. DMS エージェントを実行しているマシンに Java/OpenJDK 17 以降がインストールされていることを確認します。

2. Install the EDB DMS agent 。

3. `/opt/cdcagent/run-cdcagent.sh` にある EDB DMS エージェントを書き込み権限で開きます。

Postgres からの移行のための EDB DMS エージェントの構成

環境に応じてスクリプトの変数を設定し、編集した行のコメントを外します。詳細については、parameters を参照してください。以下は、編集されていないスクリプトを示しています。

```
# !/bin/bash -e

#  run_cdcagent.sh
```

(次のページに続く)

(前のページからの続き)

```
#

#  This script provides a convenient place to specify

#  environment variables used to configure the

#  EDB Data Migration Service Agent.

#

##########################################
#  DMS Agent General Configuration       #

##########################################

#  This ID is used to identify DMS Agent

#  and is specified by the user.

export DBCONFIG_ID=<source_name_of_choice>

#  Determine the run mode of the DMS Agent

#  The run mode can be set to either reader or writer

#  reader mode is used to read changes from a source database

#  writer mode is used to write changes to a target database

export RUN_MODE=reader

#  This is the DMS backend service used by the Agent

#  Consult your system administrators

export RW_SERVICE_HOST=https://${TRANSPORTER_RW_SERVICE_DOMAIN_NAME}/transporter

##########################################
#  DMS Agent TLS Configuration           #

##########################################

#  You need to create migration credentials in EDB Postgres AI platform
```

(次のページに続く)

(前のページからの続き)

```
#  and set the path of the credentials directory

export CREDENTIAL_DIRECTORY_PATH=$HOME/credentials

##########################################
#  DMS Agent DB Configuration     #

########################################

#  A sample configuration to create a single postgres database connection:

export DBCONFIG_DATABASES_0__TYPE=POSTGRES
export DBCONFIG_DATABASES_0__HOSTNAME=<localhost/host>
export DBCONFIG_DATABASES_0__PORT=<5432>

#  The CATALOG is the database name

export DBCONFIG_DATABASES_0__CATALOG=<source>
export DBCONFIG_DATABASES_0__USERNAME=<postgres>

#  The password env can be set without specifying it here

#  but the env structure looks like this

export DBCONFIG_DATABASES_0__PASSWORD=<password>

#  The resource_id is configured by the customer in the configuration file of the␣
↪beacon agent

#  need to make sure that the resource_id is set to the same value as in the beacon␣
↪agent

# export DBCONFIG_DATABASES_0__RESOURCEID=1234567890abcdef

#  You can increase the index to configure more than

#  one database for the DMS Agent

# export DBCONFIG_DATABASES_1__TYPE=ORACLE
# export DBCONFIG_DATABASES_1__HOSTNAME=localhost
# export DBCONFIG_DATABASES_1__PORT=1521

#  The CATALOG is the database name

# export DBCONFIG_DATABASES_1__CATALOG=ORCLCDB/ORCLPDB1
```

(次のページに続く)

(前のページからの続き)

```
# export DBCONFIG_DATABASES_1__USERNAME=oracle

#  The password env can be set without specifying it here

#  but the env structure looks like this

# export DBCONFIG_DATABASES_1__PASSWORD=password

#  The resource_id is configured by the customer in the configuration file of the␣
↪beacon agent

#  need to make sure that the resource_id is set to the same value as in the beacon␣
↪agent

# export DBCONFIG_DATABASES_1__RESOURCEID=1234567890abcdef

########################################
#  Optional Parameters Below           #

########################################

#  FIPS mode can be enabled by setting the FIPS_MODE env variable to true

# export FIPS_MODE=true

#  Configure logging

#  Loglevel for the entire Quarkus application and underlying components

#  WARNING: Enabling this loglevel is extremely verbose

# export QUARKUS_LOG_LEVEL=DEBUG

#  Loglevel for the EDB package

#  Preferred mode for debugging issues related to the logic of the agent

# export QUARKUS_LOG_CATEGORY__COM_ENTERPRISEDB__LEVEL=DEBUG

cd "$(dirname "$0")"

echo "Starting in mode: $RUN_MODE"

case "$RUN_MODE" in
  reader)
```

(次のページに続く)

(前のページからの続き)

```
    java -jar ./reader/quarkus-run.jar
    ;;
  writer)
    java -jar ./writer/quarkus-run.jar
    ;;
  *)
    echo "Unknown RUN_MODE: $RUN_MODE"
    exit 1
    ;;
esac
```

EDB DMS エージェントの実行

移行を開始します

```
cdcagent
```

Creating your HM-managed destination database

title:HM 管理の宛先データベースクラスターの作成 navTitle:HM 管理データベース

新しいデータベースを作成する

ハイブリッドマネージャー HM に宛先データベースクラスターがまだない場合は、HM が移行の宛先として使用できるクラスターを作成します。

1. HM コンソールで、データベースクラスターを作成する project を選択します。
2. プロジェクトで、移行の宛先として機能するように create a database cluster への **Create New** と **Database Cluster** を選択します。
3. **Clusters** ページで、クラスターを選択し、 **Quick Connect** オプションを使用してターミナルからデータベースにアクセスします。
4. 移行の宛先として使用する空のデータベースを作成します。または、デフォルトのデータベース `edb_admin` を使用できます。

既存のデータベースを使用する

または、既存の HM 管理データベースを移行の宛先として使用できます。既存のデータベースクラスターを移行の宛先として使用する前に、移行するテーブルと宛先で生成された負荷が既存のワークロードを干渉しないことを確認してください。

既存の HM 管理データベースを宛先として使用する場合は、次の手順に進みます。

Creating a data migration with Hybrid Manager

title HM 管理の宛先への移行の作成 navTitle 移行の作成 EDB DMS エージェントリーダーモードで構成してハイブリッドマネージャー HM への接続を確立した後、HM コンソールで移行を作成します。移行は、EDB DMS の助けを借りて、ソースデータベースと宛先データベース間の同期を確立します。

1. HM コンソールで、移行を管理するプロジェクトを選択します。
2. プロジェクトで、 **移行>移行** を選択します。
3. [**移行**] ページで、[**Create New Migration**] を選択します。
4. **Create OLTP Migration** ページで、名前を移行に割り当てます。
5. 移行のソースを選択します。 EDB DMS エージェントの ID は、 **Source** メニューにリストされています。

6. 移行タイプを選択します**Snapshot** または**Snapshot + Streaming** 。

7. **Destination Type** で、移行用の HM 管理の宛先データベースクラスターとデータベースを選択します。次に、**Next** を選択します。
8. 移行するテーブルと列を選択します。必要に応じて、テーブルと列の名前を変更します。
9. **Create Migration** を選択します。

HM コンソールには、新しい移行が表示されます。

スナップショット+ストリーミング オプションを選択した場合、移行で**実行中** 状態が表示されるときに EDB DMS エージェントは常にデータをストリーミングしています。データの変更は、移行が実行されている限りソースから宛先データベースに複製されます。

!!!note

リーダーモードの EDB DMS エージェントは、データ変更をストリーミングします。DDL オブジェクトの変更はストリーミングされません。

Creating Hybrid Manager project credentials for the migration

title: ハイブリッドマネージャー移行プロジェクトの資格情報の作成 navTitle: プロジェクトの資格情報の作成 deepToC:true データの移行を実行するには、資格情報が必要です。移行のソースまたは宛先をハイブリッドマネージャー HM に追加する前に、移行の資格情報を作成する必要があります。次に、それらを使用して、EDB DMS エージェントが HM に接続できるようにします。

HM project ごとに 1 セットの資格情報が必要です。たとえば、すべての移行に単一の HM プロジェクトを使用している場合、1 セットの資格情報のみが必要です。

1. HM コンソールで、移行アクティビティに使用しているプロジェクトを選択します。
2. プロジェクトで、 **Migrate > Credentials** を選択します。

Credentials ビューには、プロジェクト内のすべての資格情報が表形式で表示されます。各資格情報のエントリには、その名前、ステータス、作成日、および有効期限が表示されます。

3. **Create Migration Credential** を選択し、資格情報の名前を入力して、**Create Migration Credential** を選択します。

4. 資格情報が生成されたら、 **Download Credential** を選択します。

!!!note

資格情報をダウンロードできるのはこのときのみです。今すぐダウンロードせず、資格情報が必要な場合、このセットを無効にして、新しいセットを作成する必要があります。

5. ダウンロードした zip ファイルを解凍し、生成されたフォルダーを EDB DMS エージェントがインストールされているホストにコピーします。

解凍された資格情報フォルダーには、次のファイルが含まれています。

- `int.crt` - `int.truststore.p12` - `client.keystore.p12` - `client-key.pem` - `client-cert.pem`

Finalizing a streaming data migration

title: 移行のファイナライズストリーミング移行の場合 navTitle: 移行のファイナライズ移行を作成するときに**Snapshot + Streaming** オプションを選択した場合、EDB DMS エージェントは、移行を次のようにマークするまで、ソースデータベースで実行されたデータ更新を宛先データベースにストリーミングし続けます。完了しましたたとえば、カットオーバーを実行します。

Snapshot オプションを選択した場合、DMS は宛先へのスナップショット全体のコピーが終了した後、移行を完了としてマークするため、このページをスキップできます。

ストリーミングプロセスを停止します

このプロセスは、**Snapshot + Streaming** 移行にのみ適用されます。

データ移行ストリームを停止する最適な時期を注意深く選択することが重要です。ソースで新しい挿入が実行されていないことを確認し、最後の既知の挿入が宛先で同期されたことを確認します。

データが宛先データベースに存在することを確認した後、移行を完了としてマークすることにより、ストリームを停止できます。

1. Hybrid ManagerHM コンソールのプロジェクトで、 **移行>移行**を選択します。
2. 移行の横にあるチェックマークを選択して、移行が完了したことを示します。

または、リストから移行を選択して Migration details page を開きます。次に、 **Mark as Completed** を選択します。

これにより、ストリーミング手順が停止します。

Understanding schema handling

title: スキーマの準備とインポート navTitle: スキーマの準備 deepToC:true EDB Data Migration Service EDB DMS を使用して `cdcagent` を使用するデータ移行を構成する前に、スキーマを準備して宛先データベースにインポートする必要があります。

EDB DMS はデータのみを移行するため、移行が発生する前に宛先データベースにスキーマをインポートすることが重要です。移行先データベースには、移行のためにスキーマを配置してデータを入力する必要があります。

データの移行が発生する前に、スキーマの制約の一部を含める必要があります。その他は、データの移行が完了した後に適用する必要があります。このアプローチにより、パフォーマンスを低下させることなく移行できます。

スキーマの整合性とパフォーマンスの考慮事項

宛先データベースの制約、トリガー、および WAL ロギングの存在は、データ移行のパフォーマンスに影響を与える可能性があります。このアプローチが可能な場合、スキーマ制約の 2 段階インポートをお勧めします。

主キーと一意性制約

EDB DMS は、主キー/一意制約を活用して、データを宛先データベースに移行するときに 1 回だけ配信を提供します。したがって、データの移行を開始する前に実行するスキーマインポートに主キー/一意制約を含める必要があります。他の種類の制約をスキーマのインポートから除外します。

主キー/一意制約のないテーブルの行の場合、少なくとも 1 回の配信のみを実現できます。データ移行の検証中に重複排除を実行できます。

!!!note

Not null 制約は、宛先サーバーのパフォーマンスに重大な影響を与えず、スキーマのインポートに含めることもできます。

外部キー、チェック、および除外コンストレイン

EDB DMS は、宛先データベースクラスターに対して変更イベントを並列適用できます。ただし、一部の制約タイプを移行すると、移行のパフォーマンスに悪影響を与える可能性があります。この種の制約は、一貫した参照統合ソースデータベースからの飛行中のデータ移行のコンテキストで、不必要な CPU とメモリの使用率をもたらします。

したがって、移行が完了した後にのみ、宛先データベースに外部キー、チェック、および除外制約を適用することをお勧めします。

Migrating Postgres schemas with Migration Toolkit

title:Postgres スキーマの準備とインポート navTitle:Postgres スキーマの準備 deepToC:true

Postgres データベースのスキーマ処理

いくつかの異なる方法を使用して、スキーマを準備し、サポートされていない制約を除外し、宛先データベースにインポートできます。以下は、Migration Toolkit を使用して Postgres 移行のスキーマをエクスポートおよびインポートする方法に関するガイドラインです。

DDL を移行して Postgres スキーマをインポートするための他の有効なルートは次のとおりです。

- 宛先データベースにスキーマを手動で作成する
- pg_dump を実行してスキーマを取得し、

pg_restore を実行してそれらを復元する

-

pgAdmin または psql を使用する

エクスポート

Postgres からデータを移行する場合、EDB Migration Toolkit を使用してスキーマを管理することをお勧めします。MTK の offline migration 機能は、データベースのスキーマを抽出し、制約を分離する簡単な方法を提供します。以下にいくつかのガイドラインを示します。

- `-offlineMigration` オプションを使用して、Migration Toolkit は、接続を構成せずに宛先に適用できる SQL スクリプトを生成できます。
- - `schemaOnly <schema_scope>` オプションを使用して、Migration Toolkit がスキーマのエクスポートのみに焦点を当てたスクリプトを生成できます。
- Migration Toolkit を実行した後、宛先データベースに適用する前に、生成された SQL スクリプトから次の制約を除外コメントアウトします。
  - 外部キー制約 s
  - チェック制約
  - 除外制約
- Migration Toolkit の実行後、宛先の別の名前にオブジェクトをマップする予定がある場合は、スキーマ SQL スクリプトを変更して新しい名前を反映します。SQL スクリプトが生成された後、宛先に適用する前に、スクリプト DDL でオブジェクト名を直接編集します。また、スクリプト内の名前を変更したオブジェクトの構造とデータ型がソースと正確に一致することを確認します。DDL スクリプトのオブジェクト名を手動で更新する場合は、更新されたオブジェクトを参照する他のオブジェクトの名前も更新します。たとえば、テーブルの名前を変更する場合は、元のテーブル名を参照するビューまたはコンストレインも更新します。

後のデータの移行中に、新しく割り当てられたオブジェクト名を選択します。詳細については、

Mapping schema, tables and columns during a migration を参照してください。

インポート

DDL を準備し、外部キー、チェック、および除外制約を除外した後、宛先データベースに接続し、SQL 形式の DDL ファイルをインポートします。

インポートを開始するには、生成された offline migration script を実行します。

!!!note

移行プロセスの後半、移行が正常に実行された後、除外された外部キー、チェック、および除外制約を宛先データベースに適用する必要があります。したがって、除外されたコンストレインを追跡して、必要なときに適用できます。

Preparing your Postgres source database

title:Postgres ソースデータベースの準備 navTitle:Postgres ソースの準備 deepToC:true

ソースデータベースに接続します

移行のために Postgres ソースを準備するには、管理者権限が必要です。データ移行の特権が制限された変更データキャプチャ CDC 移行ロールを作成します。

psql または同様のクライアントを使用して SQL ステートメントを実行します。

`psql` を使用してソースデータベースに接続するには

```
psql -h <PG_HOST> -p <PG_PORT> -U <PG_USERNAME> -d <PG_DB_NAME>
```

そこで

- <PG_DB_NAME> は、接続する Postgres データベースソースの名前です。
- <PG_HOST> は、Postgres データベースホストです。
- <PG_PORT> は、Postgres データベースポートです。
- <PG_USERNAME> は、ロールの作成と付与、移行するテーブルの所有権を変更、およびレプリケーションスロットを作成できる管理ユーザーです。

このコマンドは、 <PG_USERNAME> に関連付けられたパスワードの入力を求めます。

Postgres 構成を設定する

Postgres の次の構成エントリが次のように `postgresql.conf` ファイルで設定されていることを確認します。

1. `wal_level` を `logical` として構成します。

CDC 移行プロセスは、Postgres 論理デコーディングを活用します。`wal_level` を `logical` に設定すると、Postgres ログ先行書き込み WAL の論理デコードが有効になります。

1. `max_wal_senders` を適切に構成します。

EDB Data Migration Service の移行がデータベースの最初のストリーミングクライアントである場合、`max_wal_senders` を少なくとも 1 に設定します。他のストリーミングクライアントが存在する場合があります。ストリーミングクライアントの接続の適切な値については、DBA に相談してください。

1. `max_replication_slots` を適切に構成します。

CDC 移行プロセスの場合、`max_replication_slots` は少なくとも 1 である必要があります。組織が Postgres レプリケーションを使用している場合、この値は高くなる場合があります。

1. 適切な WAL LSN のライフタイムのために `max_wal_size` を構成します。

実稼働トラフィックが主に時間指定されたチェックポイントを生成し、WAL サイズに基づいて要求されたチェックポイントを生成しないように、`max_wal_size` 値を十分に大きく設定します。

ストリーミング移行プロセスでは、EDB DMS のクラウドインフラストラクチャの耐久性のあるメッセージストレージにストリーミングできるまで、WAL で変更を利用できる必要があります。`max_wal_size` の設定が小さすぎると、パフォーマンスに影響を与える可能性があります。また、Postgres LSN をストリーミングする前に WAL からドロップできることにより、移行プロセスを妨げる可能性があります。

improved migration performance 場合は `max_wal_size` を少なくとも 8 GB に設定することをお勧めします。DBA に相談して、適切なサイズを決定します。

1. データベースサーバーを再起動して、これらの変更を適用します。

新しいロールを作成し、CDC 移行のためのアクセスを付与する

LOGIN および REPLICATION 機能を付与して CDC 移行の新しいロールを作成します。

```
CREATE ROLE <MIGRATION_ROLE> WITH REPLICATION LOGIN PASSWORD <MIGRATION_ROLE_PASSWORD>
↪;
```

<MIGRATION_ROLE> Postgres パブリケーションを自動作成するには、ソーステーブルを所有する必要があります。ソーステーブルは既に別のロールが所有しているため、新しい所有者として機能できるロール/ユーザーを作成し、指定されたレプリケーショングループロールを現在のテーブル所有者と<MIGRATION_ROLE>の両方に付与します。

```
CREATE ROLE <REPLICATION_GROUP>;
GRANT <REPLICATION_GROUP> TO <MIGRATION_ROLE>;
GRANT <REPLICATION_GROUP> TO <ORIGINAL_OWNER>;
ALTER TABLE <TABLE_NAME> OWNER TO <REPLICATION_GROUP>
```

そこで

- <MIGRATION_ROLE> は、CDC 移行データベースアクセスに使用する Postgres ロールまたはユーザーの名前です。
- <ORIGINAL_OWNER> は、テーブルの元のプロダクション所有者です。
- <REPLICATION_GROUP> は、パブリケーション自動作成のために移行するソーステーブルを所有するために使用されるロールの名前です。
- <TABLE_NAME> は、移行するテーブルの名前であり、大文字と小文字が区別されます。

ソーススキーマの使用を CDC 移行ロールに付与する

スキーマ内のテーブルを選択するには、スキーマのテーブルに SELECT を付与することに加えて、スキーマで使用を付与する必要があります。この使用法は、移行するテーブルを含むすべてのソーススキーマに必要です。

次のコマンドを使用して、スキーマの使用を付与できます。

```
GRANT USAGE ON SCHEMA <DB_SCHEMA> TO <MIGRATION_ROLE>;
```

ソーステーブルの SELECT を CDC 移行ロールに付与

新しい<MIGRATION_ROLE> はスキーマアクセスを持つようになりましたが、これらのスキーマのソーステーブルへの SELECT アクセスがまだ必要です。スキーマ全体またはテーブルごとにアクセスを付与できます。

スキーマのテーブル全体の場合、次を使用します。

```
GRANT SELECT ON ALL TABLES IN SCHEMA <DB_SCHEMA> TO <MIGRATION_ROLE>;
```

テーブルごとに、次を使用します。

```
GRANT SELECT ON <TABLE_NAME> TO <MIGRATION_ROLE>
```

そこで

- <DB_SCHEMA> は、移行するテーブルのデータベーススキーマ名です。
- <MIGRATION_ROLE> は、CDC 移行データベースアクセスに使用する Postgres ロールまたはユーザーの名前です。
- <TABLE_NAME> は、移行するテーブルの名前であり、大文字と小文字が区別されます。

ソースデータベースの CREATE 特権を CDC 移行ロールに付与します

pgoutput プラグインが使用されているため、Debezium がパブリケーションを作成できるようにするには、移行ユーザーにソースデータベースの CREATE 特権を付与する必要があります。 CREATE 特権を付与するには

```
GRANT CREATE ON DATABASE <PG_DB_NAME> to <MIGRATION_ROLE>
```

SSL 構成

ソースデータベースサーバーを accept SSL connections に構成して、リーダーモードの EDB DMS エージェントが接続できるようにします。この構成を有効にするには、サーバー証明書とサーバープライベートキーを作成する必要があります。プライベートキーは、OpenSSL のようなツールで作成できます。

構成を検証する

ソースデータベースを構成した後、Postgres 構成を検証します。このためには、DMS エージェントにパッケージされた構成検証スクリプトが必要です。

前提条件

1. DMS エージェントを実行しているマシンに Java/OpenJDK 17 以降がインストールされていることを確認します。

2. Install the EDB DMS Agent 。

3. /opt/cdcagent/ の EDB DMS Agent フォルダーに移動します。

構成検証スクリプトを実行する

この例は、検証スクリプトを実行する方法を示しています。

1. 移行する必要があるすべてのテーブルの名前を<schema_name>.<table_name>形式で含む配列を作成します。例

```
arr=(test1.table1 test1.table2 test1.table3 test1.table4)
```

1. 必要なパラメーターを使用してスクリプトを実行します。ここでの DBZ_USER は、cdc のユーザー上記の MIGRATION_ROLE の値を参照します。ここでの DBZ_PASSWORD は、cdc のユーザー上記の MIGRATION_ROLE_PASSWORD の値に設定したパスワードを指します、例

```
PG_USERNAME=postgres PG_PASSWORD=password PG_HOST=localhost PG_PORT=5432 DB_
↪NAME=postgres DBZ_USERNAME=debezium DBZ_PASSWORD=dbz ./postgresConfigValidation.sh "
↪${arr[@]}"
```

次のような応答が得られます。

```
  ***  [Transporter] - Get PostgreSQL Major Version
PostgreSQL major version: 16

  ***  [Transporter] - Validate WAL Level
wal_level:  logical
[Pass] wal_level is logical.

  ***  [Transporter] - Validate max WAL senders
max_wal_senders: 10
[Pass] max_wal_senders is at least 1.

  ***  [Transporter] - Validate max replication slots
max_replication_slots: 10
[Pass] max_replication_slots is at least 1.

  ***  [Transporter] - Validate max WAL size
max_wal_size: 1 GB
[Pass] max_wal_size is at least 8 GB.

  ***  [Transporter] - Validate checkpoints
checkpoints_timed: 424
checkpoints_req: 1
[Pass] Timed checkpoints are more frequent than requested checkpoints.

  ***  [Transporter] - Check debezium user role
[Pass] User debezium is present
[Pass] User debezium has replication permission

  ***  [Transporter] - Check SELECT privilege on the tables to be migrated
[Pass] User debezium has select privilege on all tables to be migrated.

  ***  [Transporter] - Check CREATE privilege on database
[Pass] Role debezium has CREATE privilege on database postgres.
```

データベースは CDC 移行の準備が整いました。

!!!note ここで推奨されているようにソースデータベース設定を変更することにより、[Failed] または [Suggestion] ステータスに対処します。 [Failed] チェックは、DMS がデータ移行を実行する前に解決が必要な問題をブロックしています。[Suggestion] チェックでは移行が許可されますが、対処しないでおくとパフォーマンスに悪影響を与える可能性があります。

詳細情報

- CDC 構成については、

Debezium Postgres Connector を参照してください。

- `max_replication_slots` 構成の詳細については、

Postgres replication documentation を参照してください。

-

*EDB blog post on tuning `max_wal_size* <https://www.enterprisedb.com/blog/tuning-maxwalsize-postgresql>`_ および Postgres WAL documentation を参照してください。

Verifying your source appears in the HM console

ソース は、リーダーモードで EDB DMS エージェントを使用してハイブリッドマネージャー HM に接続するデータベースであるため、移行に使用できます。

ソースが EDB DMS エージェントによって認識され、移行に使用できることを確認するには

1. HM コンソールで、現在の移行に使用しているプロジェクトを選択します。
2. 左側のナビゲーションメニューから、 **Migrate > Sources** を選択します。

各ソースのビューには、データベース名、データベースタイプ Postgres または Oracle、`cdcagent` ID、現在のステータス、ソースタイプなどが表示されます。

詳細情報

HM コンソールのこのページで使用可能なデータの説明については、Sources を参照してください。

Migrate your self-managed Postgres database to a different self-managed Postgres database

これらは、ハイブリッドマネージャー HM のデータ移行サービス DMS を使用して、自己管理の Postgres データベースから別の自己管理の Postgres データベースへの移行を実行するために必要なすべての手順です。

ソースデータベースの準備

データ移行のためにソースデータベースを準備します。これには、移行の専用ユーザーを作成し、データベース構成を変更してレプリケーション機能を有効にすることが含まれます。

1. *Preparing your AWS RDS/Aurora PostgreSQL source database* 。

データベースの評価オプショナル

データベースに HM エージェントをインストールおよび構成して、Postgres データベースの可観測性と監視データを取得します。このオプションの手順は、スキーマ数と名前、テーブル数、オブジェクト数などの高レベルの移行メトリックも提供します。

1. *Configure the HM agent to assess your database* 。

宛先データベースの準備

宛先データベースをホストするサーバーを作成します。次に、宛先データベースの構成を変更して、レプリケーション機能を有効にします。

1. *Creating your HM-managed destination database* 。
2. *Preparing your Postgres self-managed destination database* 。

スキーマを移行する

非互換性を処理し、スキーマを転送します。

1. *Understanding schema handling* 。
2. *Migrating Postgres schemas with Migration Toolkit* 。

データを移行する

データ転送を実行し、最終的なクリーンアップ手順を完了します。

1. *Creating Hybrid Manager project credentials for the migration* 。
2. *Configuring and running the EDB DMS agent in reader mode* 。
3. **詳細については** 。
4. *Configuring and running the EDB DMS agent in writer mode* 。
5. **詳細については** 。
6. *Creating a data migration with Hybrid Manager* 。
7. *Finalizing a streaming data migration* ストリーミング移行の場合のみ。
8. *Applying constraints after the migration is finalized* 。

Creating your self-managed destination database

title: 宛先データベースクラスターの作成 navTitle: データベースの作成既存のデータベースを移行の宛先として使用できます。既存のインスタンスを移行の宛先として使用するには、移行するテーブルと宛先で生成される負荷が既存のワークロードを干渉しないことを確認します。

自己管理の宛先データベースがまだない場合は、EDB DMS エージェントがハイブリッドマネージャー HM 移行のために接続できるデータベースを作成します。この例では、選択した Postgres データベースディストリビューションをインストールし、サーバーを初期化していることを前提としています。

1. 宛先の Postgres サーバーに接続します。

```
psql -p <PG_PORT> -U <PG_USERNAME>
```

1. 移行の宛先として使用する空の宛先データベースを作成します。

この例では、宛先データベースは `tr_target` です。

```
CREATE DATABASE tr_target;
```

Configuring the HM agent to assess your database

Hybrid ManagerHM エージェントをインストールおよび構成して、Postgres データベースの可観測性と監視データを取得できます。このオプションの手順は、スキーマ数と名前、テーブル数、オブジェクト数などの高レベルのメトリックを提供し、移行計画に役立ちます。

!!!note 移行を実行するために HM エージェントの構成は必要なく、現在実際のスキーマオブジェクト DDL を取り込みません。

HM エージェントにデータベースを登録するには、以下を参照してください。

- Monitoring self-managed deployments
-

Monitoring Cloud Service Provider managed Postgres deployments

Preparing your Postgres source database

title:Postgres ソースデータベースの準備 navTitle:Postgres ソースの準備 deepToC:true

ソースデータベースに接続します

移行のために Postgres ソースを準備するには、管理者権限が必要です。データ移行の特権が制限された変更データキャプチャ CDC 移行ロールを作成します。

psql または同様のクライアントを使用して SQL ステートメントを実行します。

`psql` を使用してソースデータベースに接続するには

```
psql -h <PG_HOST> -p <PG_PORT> -U <PG_USERNAME> -d <PG_DB_NAME>
```

そこで

- <PG_DB_NAME> は、接続する Postgres データベースソースの名前です。
- <PG_HOST> は、Postgres データベースホストです。
- <PG_PORT> は、Postgres データベースポートです。
- <PG_USERNAME> は、ロールの作成と付与、移行するテーブルの所有権を変更、およびレプリケーションスロットを作成できる管理ユーザーです。

このコマンドは、 <PG_USERNAME> に関連付けられたパスワードの入力を求めます。

Postgres 構成を設定する

Postgres の次の構成エントリが次のように `postgresql.conf` ファイルで設定されていることを確認します。

1. `wal_level` を `logical` として構成します。

CDC 移行プロセスは、Postgres 論理デコーディングを活用します。wal_level を logical に設定すると、Postgres ログ先行書き込み WAL の論理デコードが有効になります。

1. max_wal_senders を適切に構成します。

EDB Data Migration Service の移行がデータベースの最初のストリーミングクライアントである場合、max_wal_senders を少なくとも 1 に設定します。他のストリーミングクライアントが存在する場合があります。ストリーミングクライアントの接続の適切な値については、DBA に相談してください。

1. max_replication_slots を適切に構成します。

CDC 移行プロセスの場合、max_replication_slots は少なくとも 1 である必要があります。組織が Postgres レプリケーションを使用している場合、この値は高くなる場合があります。

1. 適切な WAL LSN のライフタイムのために max_wal_size を構成します。

実稼働トラフィックが主に時間指定されたチェックポイントを生成し、WAL サイズに基づいて要求されたチェックポイントを生成しないように、max_wal_size 値を十分に大きく設定します。

ストリーミング移行プロセスでは、EDB DMS のクラウドインフラストラクチャの耐久性のあるメッセージストレージにストリーミングできるまで、WAL で変更を利用できる必要があります。max_wal_size の設定が小さすぎると、パフォーマンスに影響を与える可能性があります。また、Postgres LSN をストリーミングする前に WAL からドロップできることにより、移行プロセスを妨げる可能性があります。

improved migration performance 場合は max_wal_size を少なくとも 8 GB に設定することをお勧めします。DBA に相談して、適切なサイズを決定します。

1. データベースサーバーを再起動して、これらの変更を適用します。

新しいロールを作成し、CDC 移行のためのアクセスを付与する

LOGIN および REPLICATION 機能を付与して CDC 移行の新しいロールを作成します。

```
CREATE ROLE <MIGRATION_ROLE> WITH REPLICATION LOGIN PASSWORD <MIGRATION_ROLE_PASSWORD>
↪;
```

<MIGRATION_ROLE> Postgres パブリケーションを自動作成するには、ソーステーブルを所有する必要があります。ソーステーブルは既に別のロールが所有しているため、新しい所有者として機能できるロール/ユーザーを作成し、指定されたレプリケーショングループロールを現在のテーブル所有者と<MIGRATION_ROLE>の両方に付与します。

```
CREATE ROLE <REPLICATION_GROUP>;
GRANT <REPLICATION_GROUP> TO <MIGRATION_ROLE>;
GRANT <REPLICATION_GROUP> TO <ORIGINAL_OWNER>;
ALTER TABLE <TABLE_NAME> OWNER TO <REPLICATION_GROUP>
```

そこで

- <MIGRATION_ROLE> は、CDC 移行データベースアクセスに使用する Postgres ロールまたはユーザーの名前です。
- <ORIGINAL_OWNER> は、テーブルの元のプロダクション所有者です。

- <REPLICATION_GROUP> は、パブリケーション自動作成のために移行するソーステーブルを所有するために使用されるロールの名前です。
- <TABLE_NAME> は、移行するテーブルの名前であり、大文字と小文字が区別されます。

ソーススキーマの使用を CDC 移行ロールに付与する

スキーマ内のテーブルを選択するには、スキーマのテーブルに SELECT を付与することに加えて、スキーマで使用を付与する必要があります。この使用法は、移行するテーブルを含むすべてのソーススキーマに必要です。

次のコマンドを使用して、スキーマの使用を付与できます。

```
GRANT USAGE ON SCHEMA <DB_SCHEMA> TO <MIGRATION_ROLE>;
```

ソーステーブルの SELECT を CDC 移行ロールに付与

新しい<MIGRATION_ROLE> はスキーマアクセスを持つようになりましたが、これらのスキーマのソーステーブルへの SELECT アクセスがまだ必要です。スキーマ全体またはテーブルごとにアクセスを付与できます。

スキーマのテーブル全体の場合、次を使用します。

```
GRANT SELECT ON ALL TABLES IN SCHEMA <DB_SCHEMA> TO <MIGRATION_ROLE>;
```

テーブルごとに、次を使用します。

```
GRANT SELECT ON <TABLE_NAME> TO <MIGRATION_ROLE>
```

そこで

- <DB_SCHEMA> は、移行するテーブルのデータベーススキーマ名です。
- <MIGRATION_ROLE> は、CDC 移行データベースアクセスに使用する Postgres ロールまたはユーザーの名前です。
- <TABLE_NAME> は、移行するテーブルの名前であり、大文字と小文字が区別されます。

ソースデータベースの CREATE 特権を CDC 移行ロールに付与します

pgoutput プラグインが使用されているため、Debezium がパブリケーションを作成できるようにするには、移行ユーザーにソースデータベースの CREATE 特権を付与する必要があります。 CREATE 特権を付与するには

```
GRANT CREATE ON DATABASE <PG_DB_NAME> to <MIGRATION_ROLE>
```

SSL 構成

ソースデータベースサーバーを accept SSL connections に構成して、リーダーモードの EDB DMS エージェントが接続できるようにします。この構成を有効にするには、サーバー証明書とサーバープライベートキーを作成する必要があります。プライベートキーは、OpenSSL のようなツールで作成できます。

構成を検証する

ソースデータベースを構成した後、Postgres 構成を検証します。このためには、DMS エージェントにパッケージされた構成検証スクリプトが必要です。

前提条件

1. DMS エージェントを実行しているマシンに Java/OpenJDK 17 以降がインストールされていることを確認します。

2. Install the EDB DMS Agent 。

3. /opt/cdcagent/ の EDB DMS Agent フォルダーに移動します。

構成検証スクリプトを実行する

この例は、検証スクリプトを実行する方法を示しています。

1. 移行する必要があるすべてのテーブルの名前を<schema_name>.<table_name>形式で含む配列を作成します。例

```
arr=(test1.table1 test1.table2 test1.table3 test1.table4)
```

1. 必要なパラメーターを使用してスクリプトを実行します。ここでの DBZ_USER は、cdc のユーザー上記の MIGRATION_ROLE の値を参照します。ここでの DBZ_PASSWORD は、cdc のユーザー上記の MIGRATION_ROLE_PASSWORD の値に設定したパスワードを指します、例

```
PG_USERNAME=postgres PG_PASSWORD=password PG_HOST=localhost PG_PORT=5432 DB_
↪NAME=postgres DBZ_USERNAME=debezium DBZ_PASSWORD=dbz ./postgresConfigValidation.sh "
↪${arr[@]}"
```

次のような応答が得られます。

```
  ***  [Transporter] - Get PostgreSQL Major Version
PostgreSQL major version: 16

  ***  [Transporter] - Validate WAL Level
wal_level:  logical
[Pass] wal_level is logical.

  ***  [Transporter] - Validate max WAL senders
max_wal_senders: 10
[Pass] max_wal_senders is at least 1.

  ***  [Transporter] - Validate max replication slots
max_replication_slots: 10
[Pass] max_replication_slots is at least 1.

  ***  [Transporter] - Validate max WAL size
```

(次のページに続く)

(前のページからの続き)

```
max_wal_size: 1 GB
[Pass] max_wal_size is at least 8 GB.

  ***  [Transporter] - Validate checkpoints
checkpoints_timed: 424
checkpoints_req: 1
[Pass] Timed checkpoints are more frequent than requested checkpoints.

  ***  [Transporter] - Check debezium user role
[Pass] User debezium is present
[Pass] User debezium has replication permission

  ***  [Transporter] - Check SELECT privilege on the tables to be migrated
[Pass] User debezium has select privilege on all tables to be migrated.

  ***  [Transporter] - Check CREATE privilege on database
[Pass] Role debezium has CREATE privilege on database postgres.
```

データベースは CDC 移行の準備が整いました。

!!!note ここで推奨されているようにソースデータベース設定を変更することにより、`[Failed]` または `[Suggestion]` ステータスに対処します。`[Failed]` チェックは、DMS がデータ移行を実行する前に解決が必要な問題をブロックしています。`[Suggestion]` チェックでは移行が許可されますが、対処しないでおくとパフォーマンスに悪影響を与える可能性があります。

詳細情報

- CDC 構成については、

Debezium Postgres Connector を参照してください。

- `max_replication_slots` 構成の詳細については、

Postgres replication documentation を参照してください。

-

*EDB blog post on tuning `max_wal_size* <https://www.enterprisedb.com/blog/tuning-maxwalsize-postgresql>`_ および Postgres WAL documentation を参照してください。

Understanding schema handling

title: スキーマの準備とインポート navTitle: スキーマの準備 deepToC:true EDB Data Migration Service EDB DMS を使用して `cdcagent` を使用するデータ移行を構成する前に、スキーマを準備して宛先データベースにインポートする必要があります。

EDB DMS はデータのみを移行するため、移行が発生する前に宛先データベースにスキーマをインポートすることが重要です。移行先データベースには、移行のためにスキーマを配置してデータを入力する必要があります。

データの移行が発生する前に、スキーマの制約の一部を含める必要があります。その他は、データの移行が完了した後に適用する必要があります。このアプローチにより、パフォーマンスを低下させることなく移行できます。

スキーマの整合性とパフォーマンスの考慮事項

宛先データベースの制約、トリガー、および WAL ロギングの存在は、データ移行のパフォーマンスに影響を与える可能性があります。このアプローチが可能な場合、スキーマ制約の 2 段階インポートをお勧めします。

主キーと一意性制約

EDB DMS は、主キー/一意制約を活用して、データを宛先データベースに移行するときに 1 回だけ配信を提供します。したがって、データの移行を開始する前に実行するスキーマインポートに主キー/一意制約を含める必要があります。他の種類の制約をスキーマのインポートから除外します。

主キー/一意制約のないテーブルの行の場合、少なくとも 1 回の配信のみを実現できます。データ移行の検証中に重複排除を実行できます。

!!!note

Not null 制約は、宛先サーバーのパフォーマンスに重大な影響を与えず、スキーマのインポートに含めることもできます。

外部キー、チェック、および除外コンストレイン

EDB DMS は、宛先データベースクラスターに対して変更イベントを並列適用できます。ただし、一部の制約タイプを移行すると、移行のパフォーマンスに悪影響を与える可能性があります。この種の制約は、一貫した参照統合ソースデータベースからの飛行中のデータ移行のコンテキストで、不必要な CPU とメモリの使用率をもたらします。

したがって、移行が完了した後にのみ、宛先データベースに外部キー、チェック、および除外制約を適用することをお勧めします。

Migrating Postgres schemas with Migration Toolkit

title:Postgres スキーマの準備とインポート navTitle:Postgres スキーマの準備 deepToC:true

Postgres データベースのスキーマ処理

いくつかの異なる方法を使用して、スキーマを準備し、サポートされていない制約を除外し、宛先データベースにインポートできます。以下は、Migration Toolkit を使用して Postgres 移行のスキーマをエクスポートおよびインポートする方法に関するガイドラインです。

DDL を移行して Postgres スキーマをインポートするための他の有効なルートは次のとおりです。

- 宛先データベースにスキーマを手動で作成する
- pg_dump を実行してスキーマを取得し、

pg_restore を実行してそれらを復元する

-

pgAdmin または psql を使用する

エクスポート

Postgres からデータを移行する場合、EDB Migration Toolkit を使用してスキーマを管理することをお勧めします。MTK の offline migration 機能は、データベースのスキーマを抽出し、制約を分離する簡単な方法を提供します。以下にいくつかのガイドラインを示します。

- `-offlineMigration` オプションを使用して、Migration Toolkit は、接続を構成せずに宛先に適用できる SQL スクリプトを生成できます。
- `- schemaOnly <schema_scope>` オプションを使用して、Migration Toolkit がスキーマのエクスポートのみに焦点を当てたスクリプトを生成できます。
- Migration Toolkit を実行した後、宛先データベースに適用する前に、生成された SQL スクリプトから次の制約を除外コメントアウトします。
  - 外部キー制約 s
  - チェック制約
  - 除外制約
- Migration Toolkit の実行後、宛先の別の名前にオブジェクトをマップする予定がある場合は、スキーマ SQL スクリプトを変更して新しい名前を反映します。SQL スクリプトが生成された後、宛先に適用する前に、スクリプト DDL でオブジェクト名を直接編集します。また、スクリプト内の名前を変更したオブジェクトの構造とデータ型がソースと正確に一致することを確認します。DDL スクリプトのオブジェクト名を手動で更新する場合は、更新されたオブジェクトを参照する他のオブジェクトの名前も更新します。たとえば、テーブルの名前を変更する場合は、元のテーブル名を参照するビューまたはコンストレインも更新します。

後のデータの移行中に、新しく割り当てられたオブジェクト名を選択します。詳細については、

Mapping schema, tables and columns during a migration を参照してください。

インポート

DDL を準備し、外部キー、チェック、および除外制約を除外した後、宛先データベースに接続し、SQL 形式の DDL ファイルをインポートします。

インポートを開始するには、生成された offline migration script を実行します。

!!!note

移行プロセスの後半、移行が正常に実行された後、除外された外部キー、チェック、および除外制約を宛先データベースに適用する必要があります。したがって、除外されたコンストレインを追跡して、必要なときに適用できます。

Creating Hybrid Manager project credentials for the migration

title: ハイブリッドマネージャー移行プロジェクトの資格情報の作成 navTitle: プロジェクトの資格情報の作成 deepToC:true データの移行を実行するには、資格情報が必要です。移行のソースまたは宛先をハイブリッドマネージャー HM に追加する前に、移行の資格情報を作成する必要があります。次に、それらを使用して、EDB DMS エージェントが HM に接続できるようにします。

HM project ごとに 1 セットの資格情報が必要です。たとえば、すべての移行に単一の HM プロジェクトを使用している場合、1 セットの資格情報のみが必要です。

1. HM コンソールで、移行アクティビティに使用しているプロジェクトを選択します。

2. プロジェクトで、 **Migrate > Credentials** を選択します。

Credentials ビューには、プロジェクト内のすべての資格情報が表形式で表示されます。各資格情報のエントリには、その名前、ステータス、作成日、および有効期限が表示されます。

3. **Create Migration Credential** を選択し、資格情報の名前を入力して、**Create Migration Credential** を選択します。

4. 資格情報が生成されたら、 **Download Credential** を選択します。

!!!note

資格情報をダウンロードできるのはこのときのみです。今すぐダウンロードせず、資格情報が必要な場合、このセットを無効にして、新しいセットを作成する必要があります。

5. ダウンロードした zip ファイルを解凍し、生成されたフォルダーを EDB DMS エージェントがインストールされているホストにコピーします。

解凍された資格情報フォルダーには、次のファイルが含まれています。

- `int.crt` - `int.truststore.p12` - `client.keystore.p12` - `client-key.pem` - `client-cert.pem`

Configuring and running the EDB DMS agent in reader mode

title: Postgres のリーダーモードでの EDB DMS エージェントの構成と実行ソース navTitle: Postgres の DMS エージェントの構成 deepToC:true データ移行を実行するようにリーダーモードで DMS エージェントを構成および実行します。

前提条件

1. DMS エージェントを実行しているマシンに Java/OpenJDK 17 以降がインストールされていることを確認します。

2. Install the EDB DMS agent 。

3. `/opt/cdcagent/run-cdcagent.sh` にある EDB DMS エージェントを書き込み権限で開きます。

Postgres からの移行のための EDB DMS エージェントの構成

環境に応じてスクリプトの変数を設定し、編集した行のコメントを外します。詳細については、parameters を参照してください。以下は、編集されていないスクリプトを示しています。

```
# !/bin/bash -e

#  run_cdcagent.sh

#

```

(次のページに続く)

(前のページからの続き)

```
#  This script provides a convenient place to specify

#  environment variables used to configure the

#  EDB Data Migration Service Agent.

#

##########################################
#  DMS Agent General Configuration       #

##########################################

#  This ID is used to identify DMS Agent

#  and is specified by the user.

export DBCONFIG_ID=<source_name_of_choice>

#  Determine the run mode of the DMS Agent

#  The run mode can be set to either reader or writer

#  reader mode is used to read changes from a source database

#  writer mode is used to write changes to a target database

export RUN_MODE=reader

#  This is the DMS backend service used by the Agent

#  Consult your system administrators

export RW_SERVICE_HOST=https://${TRANSPORTER_RW_SERVICE_DOMAIN_NAME}/transporter

##########################################
#  DMS Agent TLS Configuration           #

##########################################

#  You need to create migration credentials in EDB Postgres AI platform

#  and set the path of the credentials directory

export CREDENTIAL_DIRECTORY_PATH=$HOME/credentials
```

(次のページに続く)

(前のページからの続き)

```
########################################
#  DMS Agent DB Configuration     #

########################################

#  A sample configuration to create a single postgres database connection:

export DBCONFIG_DATABASES_0__TYPE=POSTGRES
export DBCONFIG_DATABASES_0__HOSTNAME=<localhost/host>
export DBCONFIG_DATABASES_0__PORT=<5432>

#  The CATALOG is the database name

export DBCONFIG_DATABASES_0__CATALOG=<source>
export DBCONFIG_DATABASES_0__USERNAME=<postgres>

#  The password env can be set without specifying it here

#  but the env structure looks like this

export DBCONFIG_DATABASES_0__PASSWORD=<password>

#  The resource_id is configured by the customer in the configuration file of the␣
↪beacon agent

#  need to make sure that the resource_id is set to the same value as in the beacon␣
↪agent

# export DBCONFIG_DATABASES_0__RESOURCEID=1234567890abcdef

#  You can increase the index to configure more than

#  one database for the DMS Agent

# export DBCONFIG_DATABASES_1__TYPE=ORACLE
# export DBCONFIG_DATABASES_1__HOSTNAME=localhost
# export DBCONFIG_DATABASES_1__PORT=1521

#  The CATALOG is the database name

# export DBCONFIG_DATABASES_1__CATALOG=ORCLCDB/ORCLPDB1
# export DBCONFIG_DATABASES_1__USERNAME=oracle

#  The password env can be set without specifying it here
```

(次のページに続く)

(前のページからの続き)

```
#  but the env structure looks like this

# export DBCONFIG_DATABASES_1__PASSWORD=password

#  The resource_id is configured by the customer in the configuration file of the␣
↪beacon agent

#  need to make sure that the resource_id is set to the same value as in the beacon␣
↪agent

# export DBCONFIG_DATABASES_1__RESOURCEID=1234567890abcdef

########################################
#  Optional Parameters Below           #

########################################

#  FIPS mode can be enabled by setting the FIPS_MODE env variable to true

# export FIPS_MODE=true

#  Configure logging

#  Loglevel for the entire Quarkus application and underlying components

#  WARNING: Enabling this loglevel is extremely verbose

# export QUARKUS_LOG_LEVEL=DEBUG

#  Loglevel for the EDB package

#  Preferred mode for debugging issues related to the logic of the agent

# export QUARKUS_LOG_CATEGORY__COM_ENTERPRISEDB__LEVEL=DEBUG

cd "$(dirname "$0")"

echo "Starting in mode: $RUN_MODE"

case "$RUN_MODE" in
  reader)
    java -jar ./reader/quarkus-run.jar
    ;;
  writer)
```

(次のページに続く)

(前のページからの続き)

```
    java -jar ./writer/quarkus-run.jar
    ;;
  *)
    echo "Unknown RUN_MODE: $RUN_MODE"
    exit 1
    ;;
esac
```

EDB DMS エージェントの実行

移行を開始します

```
cdcagent
```

Verifying your source appears in the HM console

ソース は、リーダーモードで EDB DMS エージェントを使用してハイブリッドマネージャー HM に接続するデータベースであるため、移行に使用できます。

ソースが EDB DMS エージェントによって認識され、移行に使用できることを確認するには

1. HM コンソールで、現在の移行に使用しているプロジェクトを選択します。
2. 左側のナビゲーションメニューから、 **Migrate > Sources** を選択します。

各ソースのビューには、データベース名、データベースタイプ Postgres または Oracle、cdcagent ID、現在のステータス、ソースタイプなどが表示されます。

詳細情報

HM コンソールのこのページで使用可能なデータの説明については、Sources を参照してください。

Preparing your Postgres self-managed destination database

移行を開始する前に、宛先データベースを準備します。

前提条件

- 宛先の Postgres データベースサーバーにデータベースを作成します。

SSL 構成

Hybrid ManagerHM の外部で作成された Postgres インスタンスに移行する場合にのみ SSL を構成する必要があります。HM 管理の宛先では、この構成は必要ありません。

ソースデータベースサーバーを accept SSL connections に構成して、EDB DMS エージェントが接続できるようにします。この構成を有効にするには、サーバー証明書とサーバープライベートキーを作成する必要がありますたとえば、OpenSSL を使用して。

テストのために、次の例では、Ubuntu 上の PostgreSQL 17 サーバーの OpenSSL を使用して自己署名証明書とキーを作成します。bin および data ディレクトリは、OS、 *Supported Postgres distributions* 、およびバージョンによって異なる場合があります。

1. ターミナルで、SSL 証明書とキーを生成します。

```
openssl req -new -x509 -days 365 -nodes -out server.crt -keyout server.key
```

ターミナルのプロンプトに従って、ファイルを作成します。

2. 作成したファイルを Postgres データディレクトリに移動して、サーバーがそれらにアクセスできるようにします。

```
sudo cp server.crt server.key /var/lib/postgresql/17/<data_directory>
```

3. ファイルに必要な権限を設定します。

```
sudo chmod 600 /var/lib/postgresql/17/<data_directory>/server.key
sudo chmod 600 /var/lib/postgresql/17/<data_directory>/server.crt
sudo chown postgres:postgres /var/lib/postgresql/17/<data_directory>/server.crt /var/
↪lib/postgresql/17/<data_directory>/server.key
```

4. `postgresql.conf` ファイルを変更して SSL を有効にします。

```
ssl = on
ssl_cert_file = server.crt
ssl_key_file = server.key
```

5. `pg_hba.conf` ファイルを更新して、ローカル SSL 接続を許可します。次の行を追加します。

```
hostssl all             all             0.0.0.0/0               md5
```

6. サーバーを再起動して変更を適用します。

```
/usr/lib/postgresql/17/bin/pg_ctl -D /var/lib/postgresql/17/<data_directory> restart
```

CDC 移行のロールを作成する

HM の外部で作成された Postgres インスタンスに移行する場合にのみ、移行用のロールを構成する必要があります。HM 管理の宛先では、この構成は必要ありません。

EDB DMS エージェントは、移行中にデータにアクセスし、書き込むことができるように特定の権限を必要とします。このためには、既存のスーパーユーザーを使用するか、必要な権限を持つ新しいユーザーを作成できます。この例では、新しいユーザーを作成します。

1. `LOGIN` 機能を付与して CDC 移行のロールを作成します。

```
CREATE ROLE <MIGRATION_ROLE_WRITER> WITH LOGIN PASSWORD <ROLE_PASSWORD>;
```

<MIGRATION_ROLE_WRITER> は、宛先データベースへのアクセスに使用する Postgres ロールまたはユーザーの名前です。

2. ユーザーがレプリケーションセッションの動作を制御できるように権限を付与します。

```
GRANT SET ON PARAMETER session_replication_role TO <MIGRATION_ROLE_WRITER>
```

!!!note

レプリケーションセッションの制御の付与は、Postgres 15 以降のバージョンでのみサポートされています。宛先データベースが Postgres 14 以前の場合、移行にスーパーユーザーを使用します。

DBCONFIG_DATABASES_0__USERNAME および DBCONFIG_DATABASES_0__PASSWORD 値を介してこのユーザーを EDB DMS エージェント during configuration に提供します。

Configuring and running the EDB DMS agent in writer mode

title: ライタモードでの EDB DMS エージェントの構成と実行 navTitle:DMS エージェントの構成と実行 deepToC:true 移行の宛先が自己管理の Postgres である場合にのみ、ライタモードで EDB DMS エージェントを構成および実行し、データ移行を実行しますデータベース。Hybrid ManagerHM 管理データベースへの移行の場合、ライタモードで追加の DMS エージェントは必要ありません。

前提条件

1. マシンに Java/OpenJDK 17 以降がインストールされていることを確認します。

2. Install the EDB DMS agent 。

3. /opt/cdcagent/run-cdcagent.sh にある EDB DMS エージェントを書き込み権限で開きます。

自己管理の Postgres への移行のための EDB DMS エージェントの構成

1. 環境に従ってスクリプト内の変数を設定し、編集した行のコメントを外します。詳細については、parameters を参照してください。以下は、編集されていないスクリプトを示しています。

```
# !/bin/bash -e

#  run_cdcagent.sh

#

#  This script provides a convenient place to specify

#  environment variables used to configure the

#  EDB Data Migration Service Agent.

#

########################################
#  DMS Agent General Configuration       #
```

(次のページに続く)

(前のページからの続き)

```
##########################################

#  This ID is used to identify DMS Agent

#  and is specified by the user.

export DBCONFIG_ID=<destination_name_of_choice>

#  Determine the run mode of the DMS Agent

#  The run mode can be set to either reader or writer

#  reader mode is used to read changes from a source database

#  writer mode is used to write changes to a target database

export RUN_MODE=writer

#  This is the DMS backend service used by the DMS Agent

#  Consult your system administrators

export RW_SERVICE_HOST=https://${TRANSPORTER_RW_SERVICE_DOMAIN_NAME}/transporter

##########################################
#  DMS Agent TLS Configuration           #

##########################################

#  You need to create migration credentials in EDB Postgres AI platform

#  and set set the path of the credentials directory

export CREDENTIAL_DIRECTORY_PATH=$HOME/credentials

##########################################
#  DMS Agent DB Configuration    #

##########################################

#  A sample configuration to create a single postgres database connection:

export DBCONFIG_DATABASES_0__TYPE=POSTGRES
export DBCONFIG_DATABASES_0__HOSTNAME=<localhost/host>
export DBCONFIG_DATABASES_0__PORT=<5332>
```

(次のページに続く)

(前のページからの続き)

```
#  The CATALOG is the database name

export DBCONFIG_DATABASES_0__CATALOG=<destination_db_name>
export DBCONFIG_DATABASES_0__USERNAME=<postgres>

#  The password env can be set without specifying it here

#  but the env structure looks like this

export DBCONFIG_DATABASES_0__PASSWORD=<password>

#  The resource_id is configured by the customer in the configuration file of the␣
↪beacon agent

#  need to make sure that the resource_id is set to the same value as in the beacon␣
↪agent

# export DBCONFIG_DATABASES_0__RESOURCEID=1234567890abcdef

#  You can increase the index to configure more than

#  one database for the DMS Agent

# export DBCONFIG_DATABASES_1__TYPE=ORACLE
# export DBCONFIG_DATABASES_1__HOSTNAME=localhost
# export DBCONFIG_DATABASES_1__PORT=1521

#  The CATALOG is the database name

# export DBCONFIG_DATABASES_1__CATALOG=ORCLCDB/ORCLPDB1
# export DBCONFIG_DATABASES_1__USERNAME=oracle

#  The password env can be set without specifying it here

#  but the env structure looks like this

# export DBCONFIG_DATABASES_1__PASSWORD=password

#  The resource_id is configured by the customer in the configuration file of the␣
↪beacon agent

#  need to make sure that the resource_id is set to the same value as in the beacon␣
↪agent
```

(次のページに続く)

(前のページからの続き)

```
# export DBCONFIG_DATABASES_1__RESOURCEID=1234567890abcdef

##########################################
#  Optional parameters below             #

##########################################

#  FIPS mode can be enabled by setting the FIPS_MODE env variable to true

# export FIPS_MODE=true

#  Configure logging

#  Generic loglevel

# export QUARKUS_LOG_LEVEL=DEBUG

#  Loglevel for a single package

# export QUARKUS_LOG_CATEGORY__COM_ENTERPRISEDB__LEVEL=DEBUG

cd "$(dirname "$0")"

echo "Starting in mode: $RUN_MODE"

case "$RUN_MODE" in
  reader)
    java -jar ./reader/quarkus-run.jar
    ;;
  writer)
    java -jar ./writer/quarkus-run.jar
    ;;
  *)
    echo "Unknown RUN_MODE: $RUN_MODE"
    exit 1
    ;;
esac
```

EDB DMS エージェントの実行

1. 移行を開始します。

```
cdcagent
```

Verifying your destination appears in the HM console

宛先 は、ライタモードで EDB DMS エージェントを使用してハイブリッドマネージャー HM に接続したデータベースであるため、移行に使用できます。

宛先が EDB DMS エージェントによって認識され、移行に使用できることを確認するには

1. HM コンソールで、現在の移行に使用しているプロジェクトを選択します。
2. 左側のナビゲーションメニューから、 **Migrate > Destinations** を選択します。

各宛先のビューには、データベース名、データベースタイプ、cdcagent ID、現在のステータス、宛先タイプなどが表示されます。

詳細情報

HM コンソールのこのページで使用可能なデータの説明については、Destinations を参照してください。

Creating a data migration with Hybrid Manager

title: 自己管理の宛先への移行の作成 navTitle: 移行の作成リーダーモードの EDB DMS エージェントとライタモードの EDB DMS エージェントを構成してハイブリッドマネージャー HM への接続を確立した後、HM で移行を作成しますコンソール。移行は、EDB DMS の助けを借りて、ソースデータベースと宛先データベース間の同期を確立します。

1. HM コンソールで、移行を管理するプロジェクトを選択します。
2. プロジェクトで、 **Migrate > Migrations** を選択します。
3. [**移行**] ページで、[**Create New Migration**] を選択します。
4. **Create OLTP Migration** ページで、名前を移行に割り当てます。
5. 移行のソースを選択します。 EDB DMS エージェントの ID は、 **Source** メニューにリストされています。

6. 移行タイプを選択します**Snapshot** または**Snapshot + Streaming** 。

7. **Destination Type** で、移行の宛先として自己管理インスタンスを選択します。次に、 **Next** を選択します。
8. 移行するテーブルと列を選択します。必要に応じて、テーブルと列の名前を変更します。
9. **Create Migration** を選択します。

HM コンソールには、新しい移行が表示されます。

スナップショット+ストリーミング オプションを選択した場合、移行で**実行中** 状態が表示されるときに EDB DMS エージェントは常にデータをストリーミングしています。データの変更は、移行が実行されている限りソースから宛先データベースに複製されます。

!!!note

リーダーモードの EDB DMS エージェントは、データ変更をストリーミングします。DDL オブジェクトの変更はストリーミングされません。

Finalizing a streaming data migration

title: 移行のファイナライズストリーミング移行の場合 navTitle: 移行のファイナライズ移行を作成するときに**Snapshot + Streaming** オプションを選択した場合、EDB DMS エージェントは、移行を次のようにマークするまで、ソースデータベースで実行されたデータ更新を宛先データベースにストリーミングし続けます。完了しましたたとえば、カットオーバーを実行します。

Snapshot オプションを選択した場合、DMS は宛先へのスナップショット全体のコピーが終了した後、移行を完了としてマークするため、このページをスキップできます。

ストリーミングプロセスを停止します

このプロセスは、**Snapshot + Streaming** 移行にのみ適用されます。

データ移行ストリームを停止する最適な時期を注意深く選択することが重要です。ソースで新しい挿入が実行されていないことを確認し、最後の既知の挿入が宛先で同期されたことを確認します。

データが宛先データベースに存在することを確認した後、移行を完了としてマークすることにより、ストリームを停止できます。

1. Hybrid ManagerHM コンソールのプロジェクトで、 **移行>移行**を選択します。
2. 移行の横にあるチェックマークを選択して、移行が完了したことを示します。

または、リストから移行を選択して Migration details page を開きます。次に、 **Mark as Completed** を選択します。

これにより、ストリーミング手順が停止します。

Applying constraints after the migration is finalized

title: 制約の適用 EDB Data Migration Service EDB DMS を使用したデータ移行の開始時に、ソースデータベースのスキーマを準備およびインポートしました。次に、宛先データベースに接続し、スキーマとデータの移行から除外された制約を再適用します。

主キーと一意制約

主キーと一意制約の場合、宛先の Postgres データベースにテーブルと制約が既に作成されています。これにより、EDB DMS はそれらをソースオブジェクトにマッピングし、データを正常に移行できます。他に何もする必要はありません。

not null 制約をスキーマインポートに含めた場合、同じことが適用されます。

外部キー、チェック、および除外コンストレイン

スキーマの準備とインポート中に除外した外部キー、チェック、または除外制約を再適用できるようになりました。

データの完全性の確保

主キーまたは一意制約を持たないテーブルの行は、少なくとも 1 回配信で移行されたため、これらの行は重複である可能性があります。

重複排除は、検証の一部として実行できます。

検証

すべてのデータが移行されたことを確認するには、ソースデータベースと宛先データベースを比較します。

LiveCompare を使用できます。

Reference

EDB DMS agent parameter reference

DMS エージェントの一般的な構成

これらのパラメーターは、EDB DMS エージェントが次の目的で必要な情報を定義します。

- 移行に使用している Hybrid ManagerHM インスタンスとプロジェクトにアクセスします
- ソースデータベースに接続してデータを収集するか、宛先データベースに接続してデータを移行します

!!!note DMS エージェントインスタンスは、リーダーまたはライタモードでのみ実行できます。

DBCONFIG_ID

ソース/宛先を識別するために割り当てる名前。この名前は、後で HM コンソールの**ソース** タブの **Migrate** セクションにソースとして、または **Destinations** タブに宛先として表示されます。

ID の場合

- 最大文字長は 255 文字です。
- 小文字、大文字、数字、アンダースコア_、ハイフン-が使用できます。他の特殊文字はサポートされていません。
- ID はプロジェクト内で一意である必要があります。同じプロジェクト内の 2 つ以上のソース/デスティネーションに同じ ID を使用しないでください。

RUN_MODE

EDB DMS エージェントを実行する方法を決定します。DMS エージェントを使用して HM を移行ソースに接続している場合、`reader` に設定します。 HM を外部の自己管理の移行先に接続する場合、`writer` に設定します。

移行の宛先が自己管理の Postgres データベースである場合にのみ、DMS エージェントをライタモードで実行します。

RW_SERVICE_HOST

EDB DMS エージェントが移行を実行するために使用する DMS の URL を指定します。RW_SERVICE_HOST は、移行の DMS トランスポーターサービスを活用するため、移行コンポーネント通信用に HM 管理者またはインストーラーが確立したドメイン名である TRANSPORTER_RW_SERVICE_DOMAIN_NAME から派生できます。

RW_SERVICE_HOST を取得するには

1. HM インストーラーまたは管理者に TRANSPORTER_RW_SERVICE_DOMAIN_NAME の値を提供するように依頼します。
2. https:// プロトコルプレフィックスを追加し、管理者提供の URL に/transporter を追加します。

この変更された URL は、RW_SERVICE_HOST に設定する正しい値です。

詳細情報

RW_SERVICE_HOST 値は、常に TRANSPORTER_RW_SERVICE_DOMAIN_NAME から直接取得する必要があります。次の表は、 RW_SERVICE_HOST が導き出される方法の例を示し、推奨される命名規則とカスタムの命名規則の両方を示しています。

推奨

TRANS-PORTER_RW_SERVIC	RW_SERVICE_HOST
dms.examplecompany.com	https://dms.examplecompany.com/transporter
dms-dev.examplecompany.com	https://dms-dev.examplecompany.com/transporter
dms.hm001.examplecompa	https://dms.hm001.examplecompany.com/transporter
dms-hm-us-east-001.examplecompany.com	https://dms-hm-us-east-001.examplecompany.com/transporter

カスタム

TRANS-PORTER_RW_SERVIC	RW_SERVICE_HOST
987635.examplecom-pany.com	https://987635.examplecompany.com/transporter
migra-tion_service.examplecomp	https://migration_service.examplecompany.com/transporter
loca-tion001.examplecompany.c	https://location001.examplecompany.com/transporter
us_east.examplecompany.c	https://us_east.examplecompany.com/transporter

CREDENTIAL_DIRECTORY_PATH

downloaded from the HM console 資格情報フォルダーのディレクトリパス。

DMS エージェントデータベース構成

これらのパラメーター DBCONFIG_DATABASES セクションは、EDB DMS エージェントが移行のために正しいデータベースに接続するために必要なソース/宛先データベース情報のリストを定義します。

複数のデータベースを移行するように EDB DMS エージェントを構成できます。DBCONFIG_DATABASES_0__TYPE セクションは、最初のデータベースの情報を区切ります。DBCONFIG_DATABASES_1__TYPE を使用して、2 番目のデータベースにデータを提供できます。インデックスを手動で増やして、EDB DMS エージェント DBCONFIG_DATABASES_2__TYPE、DBCONFIG_DATABASES_3__TYPE にセクションを追加します。

DBCONFIG_DATABASES_0__TYPE

ソース/宛先データベースタイプ。

EDB DMS エージェントは、ソースデータベースタイプリーダーモードとして ORACLE および POSTGRES をサポートしますが、宛先タイプライタモードとしては POSTGRES のみをサポートします。

DBCONFIG_DATABASES_0__HOSTNAME

ソース/宛先データベースのホスト名。

!!!note

RDS データベースの場合、 DBCONFIG_DATABASES_0__HOSTNAME 値が、AWS コンソールの **接続とセキュリティ**の下にある RDS エンドポイントを指していることを確認します。編集された例は foo_database.cb7asw1xxxxx.eu-west-0.rds.amazonaws.com です。

DBCONFIG_DATABASES_0__PORT

ソース/宛先データベースのポート。

DBCONFIG_DATABASES_0__CATALOG

ソース/宛先データベースサーバーのデータベース名。

DBCONFIG_DATABASES_0__USERNAME

ソース/宛先データベースのデータベースユーザー名。

DBCONFIG_DATABASES_0__PASSWORD

ソース/宛先データベースのデータベースユーザー名のパスワード。スクリプトで指定せずにパスワードの環境変数を設定できますが、変数は次のフォーマットに従う必要があります。export DBCONFIG_DATABASES_0__PASSWORD=password

DBCONFIG_DATABASES_0__RESOURCEID

リソース管理のためのユーザー定義の識別子。モニタリングとスキーマ取り込みのためにこのデータベースを HM エージェントに登録している場合、ここでは同じ `resource_id` を使用します。この一貫性は、DMS が HM エージェントと DMS エージェントコンポーネントの両方にわたってリソースを正しく相関させるために重要です。

FIPS_MODE

FIPS 標準に準拠する必要がある場合は、環境変数 `export FIPS_MODE=true` を true に設定して、DMS エージェントの FIPS モードを有効にします。

QuarkUS_LOG_LEVEL

基になるすべてのフレームワークとサードパーティのライブラリを含む Quarkus アプリケーション全体のデフォルトのログレベルを設定します。

アプリケーションエコシステム全体内の問題を診断する必要がある場合を除き、これを `DEBUG` または `TRACE` に設定することは避けてください。この方法を使用すると、過度に詳細で分析が困難なログが生成されます。集中的なデバッグには、代わりに `QUARKUS_LOG_CATEGORY__COM_ENTERPRISEDB__LEVEL` を使用します。

QUARKUS_LOG_CATEGORY__COM_ENTERPRISEDB__LEVEL

`com.enterprisedb` パッケージ内のクラスに専用のログレベルを設定します。

これは、デバッグに推奨されるパラメーターです。DMS エージェントのコアロジックスキーマ抽出、移行評価などに関連するコンポーネントの詳細なログを取得するには、このパラメーターを `DEBUG` または `TRACE` に設定します。これにより、有用な出力が分離され、サードパーティライブラリからのメッセージが除外されます。

Using the HM console to view migration assessments

Hybrid ManagerHM の主な使用の 1 つは、データベースの移行の計画と実行に役立つことです。

外部 Oracle または Postgres データベースを HM エージェントたとえば

register an Oracle database with the agent に登録できます。 **Estate** ページからアクセスできる**移行**タブには、登録した外部データベースと、計画に使用できる移行評価がリストされます。

これは、外部データベースを HM に移行する際の HM コンソールの最初の停止です。

Migration テーブル

デフォルト列

テーブルのデフォルトビューには、分析されたデータベースごとに次の情報が含まれています。

データベース名 評価されたデータベースの名前。

データベースタイプデータベースタイプ。現在サポートされているデータベースタイプについては、ステップ *7* 互換性のためにクラスターを更新する を参照してください。

リソース ID: `beacon_agent.yaml` 構成でデータベースに割り当てたリソース ID または `databases.resource_id`。詳細については、HM エージェントの *EDB DMS agent parameter reference* を参照してください。

バージョンデータベースバージョン。現在サポートされているバージョンについては、**ステップ 7 互換性のためにクラスターを更新する** を参照してください。

or Cores: データベースに割り当てられたコアの数。

of Schemas: DSN ユーザーがアクセスできる評価済みデータベース内のスキーマの数。DSN ユーザーが SELECT 特権を持つ少なくとも 1 つのテーブルが含まれています。DSN ユーザーは、HM エージェントの構成中に指定したソースデータベースユーザーです。

!!!note

- Oracle データベースでは、ビルトイン `SYS` および `SYSTEM` スキーマは列数から除外されます。
- DSN ユーザーのスキーマは、テーブルがない場合、通常除外されます。ただし、既知の問題により、DDL エクストラクターが抽出中に一時テーブルを作成するときに含まれる場合があります。
- HM エージェントは、実行中にテーブルと列のスキーマ変更を検出します。ただし、ビューまたはプロシージャへの変更は、HM エージェントを再起動するまで検出されません。

Engine: この列は Postgres データベースにのみ適用され、 PostgreSQL、EDB Postgres Advanced Server、または EDB Postgres Extended Server などのデータベースサーバーディストリビューションを示します。

複雑度この値は Oracle スキーマに対してのみ生成され、データベースを HM に移行するための複雑さのレベルを提供します。可能な値は次のとおりです。

- **非常に簡単** Oracle スキーマには互換性のない機能やオブジェクトが含まれておらず、データモデルは EDB Postgres Advanced Server と完全な互換性があります。
- **簡単**移行は比較的簡単です。データベースの構造と機能には互換性があり、最小限の調整が必要です。
- **中**中程度の移行作業が必要です。一部のデータベース構造または機能は、 EDB Postgres Advanced Server で同様の機能を取得するために変更を必要とする場合があります。
- **ハード** 多大な努力と専門知識が必要です。Oracle のデータベース構造、機能またはカスタムコードには、移行中に対処する必要がある大きな違いまたは複雑さがあります。
- **非常に難しい** 移行は非常に複雑で、要求が高いです。 Oracle と EDB Postgres Advanced Server データベースの構造、機能、カスタムコードの大幅な違いには、データ、オブジェクト、ステートメントを変換できるように専門知識が必要です。コードを書き直すか、アーキテクチャの再設計が必要になる場合があります。

移行 LoE この値は、Oracle スキーマについてのみ生成され、スキーマを変換し、データを HM に移行するために必要な営業日 8 時間単位の労力レベルを表します。この推定は、Oracle から Postgres への移行を専門とする EDB のプロフェッショナルサービスのメンバーによって適用される労力に対応しています。この値は、データベース機能、スキーマの複雑さ、データボリューム、および日数を見積もるその他の要素を含むデータベースおよびスキーマレベルの評価を考慮します。

ステータス移行の現在の状態。データベースの *Detailed view* からステータスを手動で更新して、進行状況を反映できます。

その他の列

歯車アイコンを使用して、追加の列を表示できます。

最終評価データベースが最後に評価されて、分析が最新であるかどうかを判断してから経過した期間。

管理データベースが自己管理、クラウドサービスプロバイダー、または HM によって管理されているかどうかを示します。

プロジェクト名 HM エージェントの構成に使用されたプロジェクト。これは、`agent.project_id` でエージェントに割り当てるプロジェクトです。詳細については、HM エージェント *EDB DMS agent parameter reference* を参照してください。

タグ `databases.tags` の特定のデータベースの移行に割り当てられたラベル。詳細については、HM エージェント *EDB DMS agent parameter reference* を参照してください。

TDS: GB 単位の合計データベースサイズ。これには、データベースデータのみが含まれます。メタオブジェクト、一時ファイル、ログファイルなど用に、移行先データベースに追加の領域を計画します。

移行テーブルビューの表示とフィルタリング

Estate ページの **Migration** タブには、HM エージェントに登録した後に HM で移行できるすべてのデータベースのテーブルベースの分析が表示されます。評価結果の表示またはデータベースのフィルタリング方法をカスタマイズするには、次のことができます。

- キーワードを入力して特定のデータベースを検索します。 **Search** ファンクションは、キーワードを使用して、 **リソース ID** 、**プロジェクト名** 、および **データベースタイプ**値と一致するデータベースを表示します。
- **Filter** セレクターを使用して、Postgres バージョンとプロジェクト名でデータベースを表示します。
- 結果を新しいものから古いもの、古いものから新しいもの、およびアルファベット順に並べます。
- 稲妻アイコンを使用して、テーブルを更新する頻度を選択します。アイコンが有効になっていない場合、テーブルは手動で更新しない限り更新されません。**自動更新**が有効になっている場合、テーブルは 30 秒ごとに更新されます。または、事前構成された間隔のいずれかを選択できます。
- 歯車アイコンを使用して、[**移行**] タブに表示される列をカスタマイズします。列をデフォルトの状態にリセットすることもできます。
- テーブルのサイズを画面に合わせて調整します。

関連トピック

- **ステップ 7 互換性のためにクラスターを更新する** を参照して、移行評価がサポートされているデータベースを理解します。
- データが HM エージェントによってどのように保存されるかを理解するには、*Data security and sovereignty* を参照してください。

Detailed view

ハイブリッドマネージャー HM の **エステート** ページの **移行**タブからデータベースを選択すると、詳細ビューが開きます。このビューには、外部データベースから移行を実行するために活用できる機能が含まれています。

!!!note 現在、Postgres データベースは高レベルの評価のみを提供しています。Oracle データベースでは、より包括的な分析が利用できます。

ステータス

データベース名の横にあるステータスセレクターを使用して、移行ワークフローを管理および追跡します。このセレクターを使用すると、データベースの移行の現在の状態を更新でき、自分とチームメンバーが明確にわかります。

次のステータスから選択できます。

- アセスメント デフォルト
- 企画
- 移行しないことにした
- 進行中
- 完了

スキーマ

[**Schemas**] タブは、接続したデータベースで見つかったスキーマの概要を提供します。データベースのサマリーメトリックと、少なくとも 1 つのテーブルで SELECT アクセスを付与したスキーマのリストが表示されます。少なくとも 1 つのテーブルを含むスキーマのみが表示されます。

この情報は、移行作業の範囲を調整し、スキーマの移行を効果的に計画するのに役立ちます。また、Oracle データベースの場合、**Schemas** タブを使用して、抽出したスキーマから直接 *create new Migration Portal projects* を行うことができます。

!!!重要

HM エージェントは、実行中にテーブルと列のスキーマ変更を検出します。ただし、ビューまたはプロシージャへの変更は、HM エージェントを再起動するまで検出されません。

サマリーカード

- 選択したデータベースのスキーマの合計。
- 選択したデータベースのテーブルの合計数。
- 選択したデータベースのオブジェクトの合計数。
- すべてのスキーマの複雑さの概要、複雑さごとに色分けされています非常に簡単、簡単、中程度、難しい、非常に難しい。Oracle データベースでのみ使用できます。
- 8 時間勤務日で定義された日の範囲 0-5、6-15、16-25、26-40、40+日で色分けされた移行レベル。Oracle データベースでのみ使用できます。

スキーマテーブル

- **Name:** スキーマの名前。
- **最新の評価** Oracle データベースでのみ使用できます。最新の評価のステータス。可能な値は、未開始、進行中、Complete、および Error です。
- **最終評価** Oracle データベースでのみ使用できます。最後にスキーマが評価された時刻に対応するタイムスタンプ。新しいスキーマが利用可能なことを HM エージェントが検出した場合、または既存のスキーマが更新された場合、評価は再生成または更新されます。
- **# of Tables:** 評価されたスキーマのテーブルの数。
- **# of Objects:** 評価されたスキーマ内のオブジェクトの数。
- **複雑度** Oracle データベースでのみ使用できます。詳細なオブジェクトレベルの分析に基づいて、特定のスキーマを EDB Postgres Advanced Server に変換することがいかに難しいかを示します。指定できる値は、Very Easy、Easy、Medium、Hard、および Very Hard です。スキーマレベルの分析はより詳細であるため、単一のスキーマの複雑さはデータベース全体の複雑さよりも高くなる場合があります。
- **移行 LoE** Oracle データベースでのみ使用できます。詳細なオブジェクトレベルの分析に基づいて、特定のスキーマの移行にかかる日数を 8 時間営業日で示します。
- **依存関係** Oracle データベースでのみ使用できます。現在のスキーマが依存するスキーマの名前。スキーマによって参照されるすべてのオブジェクトが宛先データベースに存在することを確認するために、依存関係を含むスキーマとその依存関係を同じバッチで移行することをお勧めします。
- **MP プロジェクト** Oracle データベースでのみ使用できます。統合された移行ポータルを使用して、Oracle スキーマを EDB Postgres Advanced Server に変換します。プラス記号を選択してウィンドウが開き、移行ポータル**プロジェクトスコープのワークスペースとユーザーのワークスペース** で新しいスキーマ移行プロジェクトを開始するための詳細を提供できます。詳細を提供すると、HM はバックグラウンドで移行ポータルプロジェクトを作成します。このプロジェクトには、後で ***Migration*** タブからアクセスできます。

アセスメント

[**Assessment**] タブは、接続されたデータベースの移行評価を提供し、作業の範囲を調整し、データベースオブジェクトの移行を計画するのに役立つ洞察を提供します。このセクションには、主要なメトリックと、生成された分析に寄与する要素の詳細な内訳を含む評価の概要が含まれています。

`.json` フォーマットで評価結果をエクスポートするには、 **Download Report** を選択します。

評価の詳細 主要な分析結果と評価メタデータの概要を示します。

互換性の概要 Oracle データベースでのみ使用できます。データベース機能とスキーマオブジェクト、および全体的な移行の複雑さと作業のレベルに寄与するその他の要素を概要します。複雑さは、複数の要因を分析するスコアリングシステムに応じて割り当てられます。要素には、データベースサイズ、ラージオブジェクトの使用量、使用されるパーティションタイプ、Oracle データベース機能、プロシージャルデータベースオブジェクト、EDB Postgres Advanced Server と互換性があるかどうかが含まれます。分析された各オブジェクトには、1〜5 のスコア非常に簡単から非常に難しいまでが与えられます。最大複雑度の値がここに表示されます。

使用される Oracle PL/ SQL 機能 Oracle データベースでのみ使用できます。Oracle ビルトイン機能を含むすべての PL/ SQL オブジェクトをリストします。Oracle PL/ SQL 機能の移行の影響を理解するには、オブジェクト数、出現数、EDB Postgres Advanced Server 互換かどうか、および移行の複雑さ評価の結果を参照します。

特定の機能のすべての呼び出しに、同等の EDB Postgres Advanced Server 実装がある場合、その機能は **EPAS 互換**としてマークされます。同等のものが検出されない場合、機能は非互換としてマークされます。

使用される Oracle データベース機能 Oracle データベースでのみ使用できます。データベースで使用されているすべての Oracle データベース機能をリストします。

オブジェクト内の互換性のない PL/ SQL 機能 Oracle データベースでのみ使用できます。Oracle および Oracle 以外の互換性のない PL/ SQL 機能をすべてリストし、そのスキーマ、オブジェクトタイププロシージャ、ファンクションなど、オブジェクト名、互換性のない機能名、機能の表示頻度、および移行の複雑さ。

移行

移行関連のタスクの実行に役立ちます。

宛先の作成移行の宛先として機能する新しいクラスターを作成します。詳細については、*Creating a database cluster* を参照してください。

スキーマの移行 - Oracle データベースの場合、このオプションでは、ビルトイン移行ポータル**プロジェクトスコープのワークスペースとユーザーのワークスペース** でプロジェクトを作成し、Oracle スキーマを EDB Postgres Advanced Server と互換性のあるものに変換するために使用できます。非互換性を解決すると、移行ポータルサービスを使用して宛先クラスターに接続し、変換されたスキーマをインポートすることもできます。

- Postgres データベースの場合、このオプションは、他のツールの中でも、Migration Toolkit または pg_dump などの外部ツールを使用して Postgres スキーマを移行する方法に関する情報を提供します。

データの移行 HM の Data Migration Service を活用して、CDC テクノロジーでデータ移行を実行します。この機能を使用してデータ移行を実行する前に、データベースの準備、スキーマの移行、および EDB DMS エージェントのインストールと構成を行ってください。完全なデータベースの移行を実行する方法のエンドツーエンドの例については、Choose your migration を参照してください。

Migration assessment support

自己管理データベースを Hybrid ManagerHM エージェントに登録した後、HM は移行評価を生成します。評価は現在、次のデータベースバージョンでサポートされています。

!!!note 現在、Postgres 評価は高レベルの移行の概要を提供します。Oracle データベースでは、より詳細な分析が利用できます。

Oracle

- Oracle 11g
- Oracle 12c
- Oracle 18c
- Oracle 19c

新しいバージョン 21c、23c なども評価される可能性がありますが、まだ完全に認定されていません。

Postgres

- PostgreSQL 13-17
- EDB Postgres Advanced Server 13-17
- EDB Postgres 拡張サーバー 13-17

Migrating schemas

スキーマの移行は、データベースの移行において重要な手順です。プロセスの複雑さはデータベースシステムによって異なるため、使用するツールはソースデータベースによって異なります。

Oracle ソース

Oracle ソースの場合、Oracle と PostgreSQL の違いにより、スキーマの移行はより複雑になります。このプロセスには、直接互換性のない構文、データ型、およびファンクションを処理するための専用の変換ステップが必要です。

このプロセスを合理化するために、Hybrid ManagerHM は統合された移行ポータルを提供します。このサービスは、Oracle スキーマの評価を自動化し、非互換性のフラグを立て、複雑なスキーマの問題の解決に役立つ AI コパイロットが含まれています。

Oracle schema migration

Postgres ソースデータベース

Postgres ソース PostgreSQL、 EDB Postgres Advanced Server、または EDB Postgres Extended Server オンプレミスまたはクラウドディストリビューションからスキーマを移行するのは、より簡単なプロセスです。ソースと HM 管理の宛先は両方とも同じテクノロジーに基づいているため、スキーマに互換性があり、変換の必要がありません。

このタスクには標準の PostgreSQL ツールを使用できます。Migration ToolkitMTK または pg_dump を使用して、ソースデータベースからスキーマを手動でエクスポートし、HM 管理の宛先クラスターにインポートすることをお勧めします。

Postgres クラスター保留中

Using Migration Portal to assess, convert, and migrate Oracle schemas

移行ポータルは、Oracle データベーススキーマを評価、分析、変換、および EDB Postgres Advanced Server に移行するのに役立ちます。web-based service として、または Hybrid ManagerHM のアプリケーションとして利用できます。

ハイブリッドマネージャー移行ポータル vs Web ベースの移行ポータル

移行ポータルの HM 展開の一部の機能は一意であるか、Web ベースのバージョンと機能が異なります。

AI コパイロット

!!!note

HM の移行ポータルでこの機能を手動で有効にする必要があります。Web ベースのバージョンでは、すぐに使えるように含まれています。

AI Copilot は、移行ポータルでのスキーマ評価と変換中に発生する可能性のある問題の解決を支援する AI 駆動のチャットボットツールです。サードパーティの AI サービスを使用し、EDB の製品ドキュメントとサポートナレッジベースの両方でトレーニングされて、ユースケース固有のソリューションを提供します。

AI コパイロットは、HM ではデフォルトで有効になっていません。

- 移行ポータルクエリーに AI コパイロットを使用するには、HM 管理者に enable and configure it with the AI/LLM of your organization を依頼します。
- 詳細については、 AI Copilot を参照してください。

プロジェクトコラボレーション

!!!note

プロジェクトコラボレーション機能は、ハイブリッドマネージャー移行ポータルでのみ利用できます。

移行ポータルは、共同 DDL 編集をサポートし、同じ HM プロジェクト内の複数のユーザーが同じスキーマファイルで同時に作業できるようにします。あなたが編集している DDL を別のユーザーが変更すると、あなたの変更は競合解決ビューをトリガーします。このビューには、他のユーザーの更新が **Update** ビューに表示され、自分の編集が **Target** ビューに表示されます。次に、 **Use Updated** を選択して他のユーザーの修正を採用するか、 **Reassess** を選択して独自の変更を適用および保存し、以前の更新を上書きできます。このアプローチにより、効率的な同時スキーマ互換性調整が保証されます。

ポータルアクセス

移行ポータルインターフェイスへのアクセスは、使用しているバージョンによって異なります。

- EDB portal を介して Web ベースの移行ポータルにアクセスします。
- HM コンソールから HM に組み込まれている移行ポータルにアクセスします。
-

project-scoped access を使用して、HM エージェントによってロードされたスキーマを移行します。

-

user's workspace access を使用して、外部提供されたスキーマを移行します。

ユーザー権限とロールベースのアクセス

- Web ベースの移行ポータルは、各 EDB アカウントに専用のワークスペースを提供します。つまり、他のユーザーとプロジェクトを共有することはできません。
- HM に組み込まれている移行ポータルは、プロジェクトスコープの共同アクセスを提供します。特定の HM プロジェクトへのアクセスを持つユーザーは、それに関連付けられている移行ポータルプロジェクトへのアクセスを取得します。

- HM では、移行ポータルのユーザー権限は HM を介して直接管理されます。たとえば、Viewer 権限を持つユーザーは、スキーマを移行したり、DDL を再評価したりすることはできません。

Accessing Migration Portal

Hybrid ManagerHM の移行ポータルは、使用する方法に応じて異なる方法でアクセスする 2 つの異なるワークスペースを提供します。

プロジェクトスコープのワークスペースとユーザーのワークスペース

HM の展開には、2 種類のワークスペースを持つ移行ポータルのビルトインインスタンスが含まれます。

- 移行ポータルの **プロジェクトスコープワークスペース** は、 HM agent によってロードされたスキーマの HM コンソールで作成された移行ポータルプロジェクトを管理する場所です。

この共有ポータル環境は、移行ポータルプロジェクトを HM プロジェクトと直接調整します。この共有ポータルでは、自分と同じ HM プロジェクトの一部である他のユーザーが作成したすべての移行ポータルプロジェクトを表示できます。ただし、他の HM プロジェクトで実行された移行から移行ポータルプロジェクトを表示することはできません。

- 移行ポータルの **ユーザーのワークスペース**は、分析、変換、および移行のために手動でアップロードする外部スキーマの移行プロジェクトを作成および管理する場所です。

これは共有環境ではありません。このワークスペースで作成された移行ポータルプロジェクトには、自分だけがアクセスできます。

この分離により、移行ポータルは、さまざまな種類と規模の移行作業に対応できます。

- 組織化された大規模なエンタープライズ主導のプロジェクトプロジェクトスコープ
- より柔軟、小規模、または個人主導の移行ユーザーのワークスペース

アクセス

移行ポータルには、さまざまな目的に応じた 2 種類のワークスペースがあります。

- HM エージェントによって HM にロードされたスキーマのプロジェクトを作成および管理するには、*Assessing schemas for HM projects* を参照してください。これは、移行ポータルへのプロジェクトスコープのアクセスを使用します。
- 外部提供された DDL ファイルを手動でアップロードし、移行ポータルでスキーマを評価するには、*Assessing schemas for external projects* を参照してください。これには、ユーザーのワークスペースから移行ポータルへのアクセスが使用されます。

Assessing schemas for HM projects

移行ポータルの協調的な project-scoped workspace を使用して、データベースの移行のコンテキストでスキーマを変換および移行します。HM は、HM コンソールを使用して移行ポータルプロジェクトを作成できる統合を提供します。このプロジェクトでは、データベースの移行の実行中に、 ***HM での AWS RDS 展開をモニタするための HM エージェントの設定*** によってプリロードされたソースデータベーススキーマを追加できます。

Oracle データベースのみが移行ワークフローへの移行ポータルの統合を含みます。スキーマは宛先データベースに移行する前に互換性評価と多くの場合変換が必要です。

外部提供された DDL スキーマファイルをアップロードして評価するには、*use the user's workspace* 。

前提条件

- HM を使用した

Register an Oracle database

- **オプション**移行ポータルでのスキーマの非互換性の解決を促進するには、HM 管理者は enable and configure the AI Copilot with the AI/LLM of your organization する必要があります。

Migration Portal へのアクセス

Oracle データベースの場合のみ、移行ポータル統合を使用してスキーマを評価および移行します。

1. **Estate** ページで、 **Migrations** タブを選択します。
2. データベースの詳細ビューを開くには、移行するデータベースの名前を選択します。

1 つ、複数、またはすべてのスキーマのスキーマ移行プロジェクトを既に作成している場合、HM コンソールの**移行ポータルプロジェクト**列にプロジェクトの名前のリンクが表示されます。リンクを選択して、そのプロジェクトのすべてのスキーマを含む移行ポータルワークスペースを開きます。

スキーマ移行のプロジェクトをまだ作成していない場合は、グローバルの **New Migration Portal Project** ボタンまたはスキーマスコープの **+** 記号を選択します。この選択により、*create a Migration Portal* プロジェクトへの新しいワークフローが開始されます。

Migration Portal プロジェクトの作成

HM コンソールから移行ポータルプロジェクトを作成できます。

1. **Estate** ページで、 **Migrations** タブを選択します。
2. データベースの詳細ビューを開くには、移行するデータベースの名前を選択します。
3. データベースの詳細ビューで、 **Schemas** タブを選択します。
4. **New Migration Portal Project** または、移行するスキーマの 1 つの横にあるプラス記号を選択します。

!!!tip **Dependencies** 列には、現在のスキーマが依存する他のスキーマが表示されます。スキーマとそのすべての依存関係を単一の移行ポータルプロジェクトにグループ化して、参照されるすべてのオブジェクトが宛先データベースに存在することを確認することをお勧めします。

1. ポップアップウィンドウで、スキーマの評価、変換、および移行をホストする移行ポータルプロジェクトの詳細を入力します。**続行** を選択します。
2. この移行ポータルプロジェクトに追加するスキーマをすべて選択します。HM エージェントが正常に取り込んだスキーマと、別の移行ポータルプロジェクトの一部になっていないスキーマのみを選択できます。

!!!note 既存のプロジェクトにスキーマを後で追加することはできません。ただし、別のプロジェクトを使用して、他のスキーマを同じ宛先データベースに移行できます。

1. **Create New Project** を選択します。

これにより、バックグラウンドで、移行ポータルプロジェクトが作成され、分析のためにソーススキーマがロードされます。

HM がスキーマのロードと移行ポータルでのプロジェクトの作成が終了すると、プロジェクトの名前のリンクが追加されます。リンクを選択して Migration Portal を開きます。

Migration Portal での作業

選択したスキーマは、移行ポータルによってアップロードされ、評価されました。AI Copilot または QuickHelp を使用して、非互換性の解決に役立ちます。

1. すべての互換性問題を解決したら、[**Migrate to**] を選択します。
2. **移行** オプションを選択します推奨。
3. 移行するスキーマを選択しますが、データベースの下のオブジェクトのリストから**コンストレイン**をクリアします。 **Next** を選択します。

!!!note

コンストレインを無効にする理由およびその他の考慮事項の詳細については、Schema integrity and performance considerations を参照してください。

1. **Connect to your cluster** ページで、宛先として使用するデータベースクラスターのホスト名を入力します。

!!!ヒント HM 管理のデータベースクラスターに移行する場合、クラスター**ページから読み取り/書き込みホストおよびその他の接続データを見つけることができます。クラスターを選択し、**Connect**を選択します。 .. ::**

!!!warning HM の分散高可用性および高度な高可用性 PGD 宛先 この警告は、移行ポータルのオンライン移行オプションを使用して、 EDB を実行している HM 管理の分散高可用性または高度な高可用性データベースクラスターにスキーマを移行する場合に適用されます。 EDB Postgres Advanced Server。この場合、 **Target Database**と**Maintenance Database**フィールドは両方とも、デフォルトの `bdrdb` データベースを指している必要があります。これにより、移行されたスキーマがクラスター全体でレプリケートされるプライマリ PGD データベースに作成されます。 .. :: 1. ** Next** を選択し、スキーマが正常にインポートされるまで待ちます。

1. ターゲットデータベースクラスターに接続し、スキーマが移行されたことを確認します。

Migration Portal プロジェクトの削除

移行ポータルのプロジェクトスコープのワークスペースを使用する場合、統合ポータルではなく、HM コンソールを介してのみプロジェクトを削除できます。

1. **Estate** ページから、 **Migrations** タブを選択します。
2. 移行したデータベースの名前を選択します。
3. データベースの詳細ビューで、 **Schemas** タブを選択します。スキーマと割り当てられた移行ポータルプロジェクトのリストが表示されます。
4. 削除する移行ポータルプロジェクトに割り当てられたスキーマを見つけ、同じ行の**+** プラス記号を選択します。

!!!note

プロジェクトを使用して複数のスキーマを評価している場合、割り当てられたスキーマのいずれかからプロジェクトを削除すると、すべてのスキーマのプロジェクトが削除されます。

5. 展開されたビューの **Actions** の下で、ゴミ箱アイコンを選択します。

6. プロジェクトを完全に削除するには、 **delete project** と入力し、**confirm Delete** を選択します。

ユーザーのワークスペースに切り替える

移行ポータルでスキーマの作業中に、手動 DDL アップロードのために **プロジェクトスコープのワークスペースとユーザーのワークスペース** に切り替えることができます。

これを行うには、 **ユーザーのワークスペースに切り替える** を選択します。

ただし、このワークスペースには、プロジェクトの作成中に選択したスキーマは表示されません。これは、タスクをすばやく切り替える方法としてのみ機能します。詳細については、*Managing external schemas for external DDL* を参照してください。

関連トピック

- 宛先データベースに変換されたスキーマの Perform an online or offline migration 。
- Migration Portal documentation ポータルの使用方法に関する推奨事項とガイドラインについては。

Assessing schemas for external projects

移行ポータルの **プロジェクトスコープのワークスペースとユーザーのワークスペース** を使用して、Oracle データベースの移行用に外部から提供された DDL ファイルを評価、変換、および移行します。

HM エージェント *access Migration Portal from the detailed view of the Oracle databases* を活用するハイブリッドマネージャー HM 移行のスコープで、移行ポータルのプロジェクトスコープのワークスペースにアクセスします。

前提条件

- **オプション**移行ポータルでのスキーマの非互換性の解決を促進するには、HM 管理者は enable and configure the AI Copilot with the AI/LLM of your organization する必要があります。

Migration Portal へのアクセス

Launchpad を使用して、Migration Portal にアクセスします。

1. Access the Launchpad 。

2. Migration Portal タブを開くには、 **Migration Portal** タイルにカーソルを合わせ、 **View Service** を選択します。

次のステップ

Migration Portal を使用して次のことを行います。

- 非互換性を解決し、スキーマの変換を完了します。
- 宛先データベースに変換されたスキーマの Perform an online or offline migration 。

使用方法に関する推奨事項とガイドラインについては、Migration Portal documentation を参照してください。

Using tools to migrate Postgres schemas

import MigrationSchemasPostgres ’../../migration_guides/snippet_sources/4_migration_schemas/preparing_postgres_schemas.mdx’ から

Using the HM console to perform data migration

データの移行を実行するには、Hybrid ManagerHM コンソールのデータ移行機能を活用できます。この参考資料では、プロジェクトの左側のナビゲーションの **Migrate** セクションで使用可能なオプションについて説明します。

- **追加リソース**リーダーモードで DMS エージェントを使用して追加した移行データベースソースを表示および管理します。
- **多様な移行先** ライタモードで DMS エージェントを使用して追加した移行データベース [宛先] を表示および管理します。
- **マイグレーション**既存のデータ移行を表示するか、既存のソースから既存の宛先への新しいデータ移行を開始します。
- **レジストリの資格情報**ソースと宛先を DMS エージェントに接続するために必要なプロジェクトスコープの資格情報を取得します。

Sources

ソースは、

Data Migration Service, or DMS, を使用してハイブリッドマネージャー HM に接続したデータベースであるため、移行のソースとして使用できます。データベースをこのビューに表示するには、EDB DMS agent in reader mode をインストール、構成、および実行する必要があります。

このビューには、プロジェクト内で構成されているすべてのソースが表示されます。

各ソースのビューには、データベース名、データベースタイプ Postgres または Oracle、cdcagent ID、現在の **ステップ *4* 展開ステータスを監視する** 、ソースタイプなどが表示されます。

ステータス

HM は、ソースの次のステータスを表示できます。

Status	Description
Creating	データベースソース接続は初期化中です。これは過渡状態です。
Waiting for tables metadata	EDB DMS エージェントは、テーブルに関する情報たとえば、名前、列、データ型を取得しています。ソースがこの状態でスタックしている場合、データベースに接続の問題がある可能性があります。
Ready to start a new migration	EDB DMS エージェントはソースデータベースを正常に読み取ることができ、移行に使用できます。
Running	このソースを使用した移行プロセスは現在アクティブであり、データは転送中です。
Paused	このソースを使用している進行中の移行プロセスは、一時的に停止されました。
Stopping	このソースで現在実行中の移行は終了中です。
Connectivity error	EDB DMS エージェントは、外部管理データベースへの接続を確立できません。これは、誤った接続の詳細ホスト、ポート、データベース名、ユーザー名、パスワードまたはネットワークの問題が原因である可能性があります。
Error	未指定の問題。 `cdcagent`ログを参照して、特定の原因を理解します。

!!!note

アクティブな移行中または一時停止を含む任意の状態の一部であるソースまたは宛先を、別の移行に使用することはできません。ソースまたは宛先を再利用する前に、既存の移行を完了するか削除する必要があります。

Destinations

宛先は、

Data Migration Service, or DMS, を使用してハイブリッドマネージャー HM に接続した外部管理データベースであるため、移行の宛先として使用できます。データベースをこのビューに表示するには、EDB DMS agent in writer mode をインストールして構成する必要があります。

このビューには、プロジェクト内で設定されているすべての宛先が表示されます。

各宛先のビューには、データベース名、データベースタイプ、cdcagent ID、現在の **ステップ *4* 展開ステータスを監視する** 、宛先タイプなどが表示されます。

ステータス

HM は、宛先の次のステータスを表示できます。

Status	Description
Waiting for tables metadata	EDB DMS エージェントは、テーブルに関する情報たとえば、名前、列、データ型を取得しています。宛先がこの状態でスタックしている場合、データベースに接続の問題がある可能性があります。
Ready to start a new migration	EDB DMS エージェントは宛先データベースに正常にアクセスでき、移行に使用できます。

次のページに続く

表 12 – 前のページからの続き

Status	Description
Running	この宛先を使用した移行プロセスは現在アクティブであり、データが転送されています。
Paused	この宛先を使用している進行中の移行プロセスは、一時的に停止されました。
Connectivity error	EDB DMS エージェントは、外部管理データベースへの接続を確立できません。これは、誤った接続の詳細ホスト、ポート、データベース名、ユーザー名、パスワードまたはネットワークの問題が原因である可能性があります。
Error	未指定の問題。 HM コンソールと `cdcagent`ログを確認して、特定の原因を理解します。

!!!note

アクティブな移行中または一時停止を含む任意の状態の一部であるソースまたは宛先を、別の移行に使用することはできません。ソースまたは宛先を再利用する前に、既存の移行を完了するか削除する必要があります。

Migrations

移行は、ハイブリッドマネージャー HM プロジェクトで開始して、データベースから別のデータベースにデータを移動できるプロセスです。これらのプロセスには、外部 Oracle データベースの HM 管理クラスタおよび外部管理の Postgres インスタンスなどへの移行が含まれます。

既存の移行の表示

1 つ以上の移行を作成すると、このビューには、プロジェクト内のすべての移行が表形式で表示されます。テーブルには次の列が含まれています。

- **Name** —移行に割り当てた名前。
- **Source** — ソースの名前。 **Sources** ページで構成します。
- **宛先** — 宛先の名前。HM 管理対象クラスター内のデータベースまたは**宛先** ページで構成された宛先のいずれか。
- **Status** —移行の現在のステータス。

ステータス

HM は、移行の次のステータスを表示できます。

Status	Description
Creating	HM は、移行を実行するために必要なリソースたとえば、Kafka ノードなどを作成しています。
Starting	実行の初期フェーズが開始され、EDB DMS エージェントがソースおよび/または宛先データベースに接続します。
Running	移行はアクティブで、データは転送中です。

次のページに続く

表 13 – 前のページからの続き

Status	Description
Pausing	移行は一時的に停止されています。
Paused	移行は一時的に停止されましたが、再開できます。
Error	未指定の問題。 **Migrations** の下で、移行の名前を選択して詳細ページを開きます。原因の詳細については、 **Error log** セクションを参照してください。
Completed	移行は正常に完了しました。
Failed	移行でエラーが発生し、完了できませんでした。 HM コンソールログを確認して、特定の原因を理解します。
Deleting	移行は終了されています。
Deleted	移行は正常に終了および削除されました。

!!!note

アクティブな移行中または一時停止を含む任意の状態の一部であるソースまたは宛先を、別の移行に使用することはできません。ソースまたは宛先を再利用する前に、既存の移行を完了するか削除する必要があります。

移行の詳細ページ

このページに表示される各移行について、移行の名前を選択することにより、より詳細なビューを開くことができます。

Mark as Completed オプション creating a migration のときに**Snapshot + Streaming** オプションを選択した場合、 **Mark as Completed** を使用してストリーミングプロシージャを停止できます。

概要タブ 概要タブには、移行のステータス、移行の種類、ソースおよび宛先データベース情報、およびトラブルシューティングに役立つエラーログに関する追加情報が含まれています。

Tables タブ Tables タブには、移行のために選択されたすべてのテーブルのリストが表示されます。テーブルごとに、ソースおよび宛先テーブル名、スキーマ、スナップショットステータス、および選択された列数を表示できます。ステータスは、スナップショットのみのジョブの全体的な移行状態、またはスナップショット+ストリーミングジョブのスナップショットフェーズステータスを反映します。各行を展開して、ソースから宛先までの詳細な列レベルのマッピングを表示できます。

移行の作成

移行を作成するには、 **Create Migration** を選択します。例については、Creating migration を参照してください。

移行中のマッピングスキーマ、テーブル、および列

HM コンソールで移行を作成する場合、データ移行サービス DMS を使用すると、ソーススキーマ、テーブル、列の名前をさまざまな宛先スキーマ、テーブル、列にマッピングできます。これは、移行中にデータベースを再構成する場合に特に役立つため、テーブル名を新しい命名規則に合わせて調整したり、明確化したりできます。

前提条件

DMS では、データの移行が開始される前に、すべての宛先オブジェクトが宛先データベースに存在する必要があります。ソースオブジェクトスキーマ、テーブル、または列を宛先の別の名前にマッピングすることにより、データを再構成する予定があるとします。たとえば、ソーステーブル `loc` から宛先テーブル `location` など。データの移行を開始する前に、宛先データベースで宛先オブジェクト `location` を手動で作成または名前変更します。

- **スキーマ移行ツール** スキーマのインポートに Migration Toolkit または pg_dump / pg_restore などのユーティリティを使用する場合、オブジェクトはソースと同じ名前で作成されます。次に、目的のマッピングと一致するように、宛先内のオブジェクトの名前を手動で変更する必要があります。たとえば、移行ツールキットのオフラインオプションまたは pg_dump を使用して SQL スクリプトを生成し、スクリプト内のオブジェクト名を直接編集し、変更したスクリプトを宛先に適用できます。
- **手動/代替作成**宛先オブジェクトを手動で作成する場合、マップされたすべての列と互換性のあるデータ型でオブジェクトが定義されていることを確認します。DMS は別の名前のオブジェクトにデータを移行できますが、宛先オブジェクトの構造はソースデータと一致する必要があります。

オブジェクトマッピング

1. HM コンソールで、**移行>移行>Create Migration** を選択します。詳細については、Creating migration を参照してください。
2. **Create Migrations** ウィンドウで、移行の **Tables** タブを選択します。
3. スキーマを名前変更した宛先スキーマにマップするには
    1. リスト内でスキーマの名前を変更するテーブル<schema>.<table> を見つけます。
    2. <schema>.<table> 名前の横にあるペンアイコンを選択します。
    3. スキーマの新しい名前を入力します。
4. 名前を変更した宛先テーブルにテーブルをマップするには
    1. リスト<schema>.<table> で名前を変更するテーブルを見つけます。
    2. <schema>.<table> 名前の横にあるペンアイコンを選択します。
    3. テーブルの新しい名前を入力します。
5. 名前を変更した宛先列に列をマップするには
    1. リストからいずれかの項目 <schema>.<table> を展開します。
    2. **ソース列** リストで名前を変更する列を見つけます。
    3. 列の新しい名前を **宛先列** に入力します。

このプロセスでは、移行定義の一部としてスキーマ、テーブル、および列をマップできます。

Credentials

移行を実行するには資格情報が必要です。移行のソースまたは宛先をハイブリッドマネージャー HM に追加する前に、移行の資格情報を作成する必要があります。次に、それらを使用して、EDB DMS エージェントが HM に接続できるようにします。

Credentials ビューには、プロジェクト内のすべての資格情報が表形式で表示されます。各資格情報のエントリには、その名前、ステータス、作成日、および有効期限が含まれます。

移行資格情報の作成

移行資格情報を作成およびダウンロードするには、

Getting credentials を参照してください。

解凍された資格情報フォルダーには、次のファイルが含まれています。

- `int.crt`
- `int.truststore.p12`
- `client.keystore.p12`
- `client-key.pem`
- `client-cert.pem`

Using the Migration API

Migration API は、データベースの移行タスクを自動化するための重要なツールです。プログラム的に移行を管理および監視でき、移行ワークフローを既存の CI / CD パイプラインまたは自動スクリプトに統合する強力な方法を提供します。UI をナビゲートする代わりに、API を使用して、移行プロジェクトの管理、詳細なスキーマレポートの表示、データの移行ステータスを確認し、その他の多くの管理タスクを実行できます。

この API は、大量の移行を処理するか、より広範な自動化戦略に統合するためのスケーラブルで自動化された方法を必要とする開発者、DevOps エンジニア、および DBA を対象としています。

Hybrid ManagerHM プラットフォームは、評価、スキーマ変換、データ移行を含むエンドツーエンドのデータベース移行をサポートするためのより広範な API セットを提供します。ただし、例 には、サブセットのみが表示されます。

例

Migration Portal プロジェクトの作成

この API エンドポイントは、移行ポータルで新しい移行プロジェクトを作成するために使用されます。これは、JSON ペイロードを送信してプロジェクトのプロパティと構成を定義する `POST` 要求です。

- **パス** `/api/v1/projects/prj_LTIfqp81u6MslAk4/migration-portal-project`
- **メソッド** `POST`

URL パスパラメーター

- prj_LTIfqp81u6MslAk4 プロジェクトのスコープの一意の識別子。

```
Request Body:
{
    "projectId": "prj_LTIfqp81u6MslAk4",
    "resourceId": "oracleSource",
    "schemaIds": [
        "cea06ec8-8641-4dff-9631-4b19fcba5212",
        "5770b6fa-295c-40af-bd02-1acbe02c9eb6"
    ],
    "mpProjectName": "my_project_name",
    "applicationInterface": "JDBC",
    "targetDbVersion": "EPAS_17",
    "useUniqueSchemaObjectSuffixes": true,
    "useOracleDefaultCase": true,
    "assessDefaultProfile": true,
    "description": "My Project Description (Optional)"
}
```

要求本文の説明

- projectId HM 内のプロジェクトの一意の識別子。
- resourceId ソースデータベースの一意の識別子たとえば oracleSource 。
- schemaIds 移行プロジェクトに含めるスキーマの一意の ID の配列。
- mpProjectName 新しい移行ポータルプロジェクトに割り当てるカスタム名。
- applicationInterface ソースデータベースへの接続に使用されるインターフェイスのタイプたとえば、JDBC 、ODBC 、.NET 、OCI 、ProC 、Other など。
- targetDbVersion 移行先のターゲットデータベースのバージョンたとえば、EDB Postgres Advanced Server バージョン 17 の場合は EPAS_17 。
- useUniqueSchemaObjectSuffixes true に設定すると、ターゲットスキーマ内のオブジェクト名に一意のサフィックスを持つことを保証するブール値、競合を回避します。
- useOracleDefaultCase true に設定されている場合、オブジェクト名に大文字を使用する Oracle のデフォルト動作を尊重するブール値。
- assessDefaultProfile true に設定されている場合、評価にデフォルトプロファイルを含むブール値。
- description 移行プロジェクトのカスタム、オプションの説明。

Migration Portal プロジェクトの削除

この API エンドポイントは、既存の移行プロジェクトを移行ポータルから削除します。これは、ID で特定のプロジェクトをターゲットとする DELETE 要求です。

- **パス**/api/v1/projects/prj_LTIfqp81u6MslAk4/migration-portal-project/42

- メソッド `DELETE`

URL パスパラメーター

- `prj_LTIfqp81u6MslAk4` プロジェクトのスコープの一意の識別子。
- `42` 削除する特定の移行ポータルプロジェクトの一意の ID。この ID は、プロジェクトが正常に作成された後に割り当てられます。

Installing EDB Data Migration Service agent on Linux

> **注釈**
>
> Hybrid Manager バージョン 1.3 以降を使用して EDB DMS エージェントをインストールおよび構成します。 Hybrid Manager 1.2 以前のバージョンの場合、EDB DMS Reader および EDB DMS Writer を使用してください。

リンクを選択して、該当するインストール手順にアクセスします。

Linux x86-64 (amd64)

Red Hat Enterprise Linux RHEL とその派生

- *Installing EDB Data Migration Service agent on RHEL 9 or OL 9 x86_64* 、*Installing EDB Data Migration Service agent on RHEL 8 or OL 8 x86_64*
- *Installing EDB Data Migration Service agent on RHEL 9 or OL 9 x86_64* 、*Installing EDB Data Migration Service agent on RHEL 8 or OL 8 x86_64*
- *Installing EDB Data Migration Service agent on AlmaLinux 9 or Rocky Linux* 、 *Installing EDB Data Migration Service agent on AlmaLinux 8 or Rocky Linux*
- *Installing EDB Data Migration Service agent on AlmaLinux 9 or Rocky Linux* 、 *Installing EDB Data Migration Service agent on AlmaLinux 8 or Rocky Linux*

SUSE Linux Enterprise SLES

- *Installing EDB Data Migration Service agent on SLES 15 x86_64*

Debian とデリバティブ

- *Installing EDB Data Migration Service agent on Ubuntu 24.04 x86_64* 、 *Installing EDB Data Migration Service agent on Ubuntu 22.04 x86_64*
- *Installing EDB Data Migration Service agent on Debian 12 arm64*、*Installing EDB Data Migration Service agent on Debian 11 x86_64*

Linux Installing EDB Data Migration Service agent on Linux AArch64 (ARM64)

Debian とデリバティブ

- *Installing EDB Data Migration Service agent on Debian 12 arm64*

Installing EDB Data Migration Service agent on Linux x86 (amd64)

オペレーティングシステム固有のインストール手順は、対応するドキュメントに記載されています。

Red Hat Enterprise Linux RHEL とその派生

- *Installing EDB Data Migration Service agent on RHEL 9 or OL 9 x86_64*
- *Installing EDB Data Migration Service agent on RHEL 8 or OL 8 x86_64*
- *Installing EDB Data Migration Service agent on RHEL 9 or OL 9 x86_64*
- *Installing EDB Data Migration Service agent on RHEL 8 or OL 8 x86_64*
- *Installing EDB Data Migration Service agent on AlmaLinux 9 or Rocky Linux*
- *Installing EDB Data Migration Service agent on AlmaLinux 8 or Rocky Linux*
- *Installing EDB Data Migration Service agent on AlmaLinux 9 or Rocky Linux*
- *Installing EDB Data Migration Service agent on AlmaLinux 8 or Rocky Linux*

SUSE Linux Enterprise SLES

- *Installing EDB Data Migration Service agent on SLES 15 x86_64*

Debian とデリバティブ

- *Installing EDB Data Migration Service agent on Ubuntu 24.04 x86_64*
- *Installing EDB Data Migration Service agent on Ubuntu 22.04 x86_64*
- *Installing EDB Data Migration Service agent on Debian 12 arm64*
- *Installing EDB Data Migration Service agent on Debian 11 x86_64*

Installing EDB Data Migration Service agent on Debian 11 x86_64

前提条件

インストールプロセスを開始する前に

- EDB リポジトリの設定をします。

リポジトリのセットアップは 1 回限りのタスクです。すでにリポジトリを設定している場合、この手順を実行する必要はありません。

リポジトリが存在するかどうかを判断するには、次のように入力します。

```
apt-cache search enterprisedb
```

出力が生成されない場合、リポジトリはインストールされます。

EDB リポジトリをセットアップするには

1. EDB repositories に移動します .
2. EDB リポジトリへのアクセスを提供するボタンを選択します。
3. ダウンロードするプラットフォームとソフトウェアを選択します。
4. EDB リポジトリの設定手順に従ってください。

パッケージをインストールする

EDB DMS エージェント cdcagent としてパッケージ化されたものをインストールします。

```
sudo apt-get install cdcagent
```

Installing EDB Data Migration Service agent on Debian 12 x86_64

前提条件

インストールプロセスを開始する前に

- EDB リポジトリの設定をします。

リポジトリのセットアップは 1 回限りのタスクです。すでにリポジトリを設定している場合、この手順を実行する必要はありません。

リポジトリが存在するかどうかを判断するには、次のように入力します。

```
apt-cache search enterprisedb
```

出力が生成されない場合、リポジトリはインストールされます。

EDB リポジトリをセットアップするには

1. EDB repositories に移動します .
2. EDB リポジトリへのアクセスを提供するボタンを選択します。
3. ダウンロードするプラットフォームとソフトウェアを選択します。
4. EDB リポジトリの設定手順に従ってください。

パッケージをインストールする

EDB DMS エージェント cdcagent としてパッケージ化されたものをインストールします。

```
sudo apt-get install cdcagent
```

Installing EDB Data Migration Service agent on AlmaLinux 8 or Rocky Linux

前提条件

インストールプロセスを開始する前に

- EDB リポジトリの設定をします。

リポジトリのセットアップは 1 回限りのタスクです。すでにリポジトリを設定している場合、この手順を実行する必要はありません。

リポジトリが存在するかどうかを判断するには、次のように入力します。

```
dnf repolist | grep enterprisedb
```

出力が生成されない場合、リポジトリはインストールされます。

EDB リポジトリをセットアップするには

1. EDB repositories に移動します .
2. EDB リポジトリへのアクセスを提供するボタンを選択します。
3. ダウンロードするプラットフォームとソフトウェアを選択します。
4. EDB リポジトリの設定手順に従ってください。

パッケージをインストールする

EDB DMS エージェント `cdcagent` としてパッケージ化されたものをインストールします。

```
sudo dnf install cdcagent
```

Installing EDB Data Migration Service agent on AlmaLinux 9 or Rocky Linux

前提条件

インストールプロセスを開始する前に

- EDB リポジトリの設定をします。

リポジトリのセットアップは 1 回限りのタスクです。すでにリポジトリを設定している場合、この手順を実行する必要はありません。

リポジトリが存在するかどうかを判断するには、次のように入力します。

```
dnf repolist | grep enterprisedb
```

出力が生成されない場合、リポジトリはインストールされます。

EDB リポジトリをセットアップするには

1. EDB repositories に移動します .
2. EDB リポジトリへのアクセスを提供するボタンを選択します。
3. ダウンロードするプラットフォームとソフトウェアを選択します。

4. EDB リポジトリの設定手順に従ってください。

パッケージをインストールする

EDB DMS エージェント cdcagent としてパッケージ化されたものをインストールします。

```
sudo dnf install cdcagent
```

Installing EDB Data Migration Service agent on RHEL 8 or OL 8 x86_64

前提条件

インストールプロセスを開始する前に

- EDB リポジトリの設定をします。

リポジトリのセットアップは 1 回限りのタスクです。すでにリポジトリを設定している場合、この手順を実行する必要はありません。

リポジトリが存在するかどうかを判断するには、次のように入力します。

```
dnf repolist | grep enterprisedb
```

出力が生成されない場合、リポジトリはインストールされます。

EDB リポジトリをセットアップするには

1. EDB repositories に移動します .
2. EDB リポジトリへのアクセスを提供するボタンを選択します。
3. ダウンロードするプラットフォームとソフトウェアを選択します。
4. EDB リポジトリの設定手順に従ってください。

パッケージをインストールする

EDB DMS エージェント cdcagent としてパッケージ化されたものをインストールします。

```
sudo dnf install cdcagent
```

Installing EDB Data Migration Service agent on RHEL 9 or OL 9 x86_64

前提条件

インストールプロセスを開始する前に

- EDB リポジトリの設定をします。

リポジトリのセットアップは 1 回限りのタスクです。すでにリポジトリを設定している場合、この手順を実行する必要はありません。

リポジトリが存在するかどうかを判断するには、次のように入力します。

```
dnf repolist | grep enterprisedb
```

出力が生成されない場合、リポジトリはインストールされます。

EDB リポジトリをセットアップするには

1. EDB repositories に移動します .
2. EDB リポジトリへのアクセスを提供するボタンを選択します。
3. ダウンロードするプラットフォームとソフトウェアを選択します。
4. EDB リポジトリの設定手順に従ってください。

パッケージをインストールする

EDB DMS エージェント cdcagent としてパッケージ化されたものをインストールします。

```
sudo dnf install cdcagent
```

Installing EDB Data Migration Service agent on SLES 15 x86_64

前提条件

インストールプロセスを開始する前に

- EDB リポジトリの設定をします。

リポジトリのセットアップは 1 回限りのタスクです。すでにリポジトリを設定している場合、この手順を実行する必要はありません。

リポジトリが存在するかどうかを判断するには、次のように入力します。

```
zypper lr -E | grep enterprisedb
```

出力が生成されない場合、リポジトリはインストールされます。

EDB リポジトリをセットアップするには

1. EDB repositories に移動します .
2. EDB リポジトリへのアクセスを提供するボタンを選択します。
3. ダウンロードするプラットフォームとソフトウェアを選択します。
4. EDB リポジトリの設定手順に従ってください。

- 必要な SUSE モジュールをアクティブにします。

```
sudo SUSEConnect -p PackageHub/15.7/x86_64
```

- メタデータを更新します。

```
sudo zypper refresh
```

パッケージをインストールする

EDB DMS エージェント cdcagent としてパッケージ化されたものをインストールします。

```
sudo zypper install cdcagent
```

Installing EDB Data Migration Service agent on Ubuntu 22.04 x86_64

前提条件

インストールプロセスを開始する前に

- EDB リポジトリの設定をします。

リポジトリのセットアップは 1 回限りのタスクです。すでにリポジトリを設定している場合、この手順を実行する必要はありません。

リポジトリが存在するかどうかを判断するには、次のように入力します。

```
apt-cache search enterprisedb
```

出力が生成されない場合、リポジトリはインストールされます。

EDB リポジトリをセットアップするには

1. EDB repositories に移動します .
2. EDB リポジトリへのアクセスを提供するボタンを選択します。
3. ダウンロードするプラットフォームとソフトウェアを選択します。
4. EDB リポジトリの設定手順に従ってください。

パッケージをインストールする

EDB DMS エージェント cdcagent としてパッケージ化されたものをインストールします。

```
sudo apt-get install cdcagent
```

Installing EDB Data Migration Service agent on Ubuntu 24.04 x86_64

前提条件

インストールプロセスを開始する前に

- EDB リポジトリの設定をします。

リポジトリのセットアップは 1 回限りのタスクです。すでにリポジトリを設定している場合、この手順を実行する必要はありません。

リポジトリが存在するかどうかを判断するには、次のように入力します。

```
apt-cache search enterprisedb
```

出力が生成されない場合、リポジトリはインストールされます。

EDB リポジトリをセットアップするには

1. EDB repositories に移動します .
2. EDB リポジトリへのアクセスを提供するボタンを選択します。
3. ダウンロードするプラットフォームとソフトウェアを選択します。
4. EDB リポジトリの設定手順に従ってください。

パッケージをインストールする

EDB DMS エージェント cdcagent としてパッケージ化されたものをインストールします。

```
sudo apt-get install cdcagent
```

Installing EDB Data Migration Service agent on Linux AArch64 (ARM64)

オペレーティングシステム固有のインストール手順は、対応するドキュメントに記載されています。

Debian とデリバティブ

- *Installing EDB Data Migration Service agent on Debian 12 arm64*

Installing EDB Data Migration Service agent on Debian 12 arm64

前提条件

インストールプロセスを開始する前に

- EDB リポジトリの設定をします。

リポジトリのセットアップは 1 回限りのタスクです。すでにリポジトリを設定している場合、この手順を実行する必要はありません。

リポジトリが存在するかどうかを判断するには、次のように入力します。

```
apt-cache search enterprisedb
```

出力が生成されない場合、リポジトリはインストールされます。

EDB リポジトリをセットアップするには

1. EDB repositories に移動します .
2. EDB リポジトリへのアクセスを提供するボタンを選択します。
3. ダウンロードするプラットフォームとソフトウェアを選択します。
4. EDB リポジトリの設定手順に従ってください。

パッケージをインストールする

EDB DMS エージェント cdcagent としてパッケージ化されたものをインストールします。

```
sudo apt-get install cdcagent
```

Asset management

Hybrid ManagerHM コンソールの Asset Management セクションは、クラスターで使用可能な資産を管理するためのツールを提供します。これには、コンテナイメージと AI モデル、およびこれらのアセットを含むリポジトリの管理が含まれます。

HM Postgres とモデルアセットの管理

- *Asset library*

HM コンソールでアセットを管理するためのメインページ。コンテナイメージとナレッジベースを表示および管理できます。

- *Sync container images to your registry*

イメージの詳細を検索、フィルタリング、表示することにより、アセットライブラリ内のコンテナイメージを探索および管理できます。

- *Manage repositories in the asset library*

リポジトリの登録とルールを管理することにより、アセットライブラリ内のリポジトリと画像の可視性を制御します。

- *RAG* のナレッジベースを作成するにはどうすればよいですか*?*

クラスターで使用する AI モデルを含むナレッジベースの作成と管理を含む、アセットライブラリの AI モデルを管理します。

Asset library

アセットライブラリは、ハイブリッドマネージャークラスターで使用可能なコンテナイメージとモデルを管理する場所です。

アセットライブラリへのアクセス

アセットライブラリにアクセスするには、Hybrid ManagerHM コンソールの上部にある **Asset ライブラリー**タブを選択します。

アセットライブラリーページ

アセットライブラリーページには、ライブラリで使用可能な画像の概要数、検索およびフィルタバー、およびライブラリで使用可能な主要画像のリストが表示されます。

アセットライブラリは、 **コンテナイメージ** と **モデル**の 2 つのメインセクションに分割されています。各セクションでは、HM クラスターで使用可能なアセットを表示および管理できます。

コンテナイメージ

アセットライブラリの **Container Images** セクションでは、HM クラスターで使用可能なコンテナイメージを表示および管理できます。これには、特定の画像の検索、さまざまな基準で画像リストをフィルタリング、および各画像に関する詳細情報を表示する機能が含まれます。このタブの使用方法の詳細については、*Sync container images to your registry* を参照してください。

モデル

モデルは、HM で使用できる AI モデルです。これらのモデルを特定のデータセットで使用して、自然言語処理、画像認識などを含むさまざまなタスクを実行できます。このタブの使用方法の詳細については、*RAG のナレッジベースを作成するにはどうすればよいですか?* を参照してください。

Container images in the asset library

アセットライブラリの **Container Images** セクションでは、ハイブリッドマネージャー HM クラスターで使用可能なコンテナイメージを表示および管理できます。

コンテナイメージ

コンテナイメージは、さまざまな拡張機能を含むデータベースの構成です。各画像は、名前と画像タグによって識別されます。

名前ごとに、イメージにロードされた拡張機能のさまざまなバージョンとさまざまな組み合わせを表す複数のイメージタグが存在する場合があります。

たとえば、公式 HM イメージには、イメージ名が `postgresql` と、バージョン 17.4 のコミュニティ PostgreSQL イメージを表すイメージタグ `17.4-2503191025` があります。 `2503191025` は、イメージのビルド日時 2025/03/19 10:25 です。後置 `-full` は、このイメージに HM Postgres ディストリビューションで使用可能なすべての拡張機能が含まれていることを示します。

このタグ形式は公式 HM 画像に固有であり、他の画像とは異なる場合があります。イメージ名は、Postgres の特定のフレーバーなど、画像のタイプを識別するために使用されます。イメージタグは、イメージの特定のバージョンを識別するために使用されます。

画像統計

ライブラリで使用可能な画像の数の概要がタイプごとにリストされています。画像の種類は次のとおりです。

- **合計イメージタグ**ライブラリで使用可能なイメージタグの合計数。タグは画像の特定のバージョンを表します。
- **PostgreSQL Images** ライブラリで使用可能な Postgres イメージの数。
- **AI モデル画像** ライブラリで使用可能な AI モデル画像の数。

コンテナイメージの検索とフィルターバー

このビューでは、**Search** フィールドを使用して、イメージリポジトリの名前または説明の一部である検索用語を入力します。**Filter** フィールドを使用して、タイプ Postgres または AI モデルでリポジトリをフィルタリングします。

Sort フィールドを使用して、名前、説明、タイプ、またはリポジトリが更新された日でリポジトリを並べ替えます。

検索とフィルターバーで、**Manage Repositories** を選択して、ライブラリで使用可能なリポジトリを管理します。ダイアログボックスを使用して、リポジトリを追加、編集、または削除します。**Manage Repositories** ページでは、アセットライブラリで使用可能なリポジトリとイメージを管理できます。リポジトリの管理方法の詳細については、*Manage repositories in the asset library* を参照してください。

コンテナイメージカード

各イメージリポジトリは、次の情報を含むカード形式で表示されます。

- **Name** イメージリポジトリの名前。
- **Description** イメージリポジトリの簡単な説明。
- **タグ**この画像リポジトリに関連付けられた HM タグ。イメージリポジトリにタグが関連付けられていない場合、**Add Tag** が表示されます。
- **Updated** イメージリポジトリが最後に更新されてからの日数。

イメージリポジトリの名前を選択して、そのイメージの詳細を含むページに移動します。

画像の詳細

画像の **画像の詳細** ページには、 **概要** と **イメージタグ**の 2 つのタブがあります。 **Add Tag** ボタンを使用して、HM タグを画像リポジトリに追加できます。既存のタグはボタンの前に表示され、タグを選択して詳細を表示できます。

[概要] タブ

[**概要**] タブは、以下を含むイメージリポジトリの概要を提供します。

- **README** イメージリポジトリのテキストによる説明。これには、画像の使用方法、含まれる内容、その他の役立つ詳細など、画像に関する関連情報が含まれます。
- **最近の画像タグ**この画像リポジトリで最近作成された 10 個の画像タグ。各タグを選択して、そのタグの**イメージタグの詳細** ページに移動できます。

クラスターを使用してこのイメージセクションには、このリポジトリのイメージを使用しているエステート内のクラスターがリストされます。このイメージを使用しているクラスターがない場合、「No Data」というメッセージが表示されます。

検索とフィルターバーを使用して、クラスターを検索します。 **Search** フィールドを使用して、クラスターの名前または説明の一部である検索用語を入力します。**Filter** フィールドを使用して、属性クラスタータイプ、クラスターアーキテクチャ、Postgres フレーバー、および Postgres バージョンでクラスターをフィルタリングします。**Sort** フィールドを使用して、名前、説明、またはクラスターが作成された日で並べ替えるクラスターを表示します。

クラスター名を選択して、そのクラスターの **クラスターの詳細の概要**ページに移動します。

画像タグ

Image Tags タブを選択して、このイメージリポジトリに関連付けられているすべてのイメージタグのリストを表示します。

各タグは、次の情報を含む表形式で表示されます。

- **Name** 画像タグの名前。
- **Created** 画像タグが作成された日時。
- **Last Pushed** イメージタグがリポジトリにプッシュされた日時。
- **プラットフォームアーキテクチャ**イメージタグが使用可能なプラットフォームアーキテクチャ、X86 または AMD64 のいずれか。
- **SHA** 画像の SHA256 ハッシュ。画像の一意の識別子。これを使用して、他のリポジトリと比較してイメージを検証できます。ハッシュの最後の 8 桁が表示されます。これらの数字にカーソルを合わせると、完全なハッシュが表示されます。**Copy** ボタンを使用して、完全なハッシュをコピーできます。

イメージタグのリストをナビゲートするには、テーブルの上部にある検索とフィルタバーを使用できます。**Search** フィールドに、画像タグ名の一部である検索用語を入力します。**Filter** フィールドを使用して、属性 EDB 公式リリース、利用可能な場所、プラットフォームアーキテクチャによってイメージタグをフィルタリングします。**Sort** フィールドを使用して、バージョン、作成日、または最後にプッシュされた日でイメージタグを並べ替えます。

イメージタグの名前を選択して、そのタグの **イメージタグの詳細**ページに移動します。

イメージタグの詳細

Image Tag Details ページには、特定の画像タグに関する最も詳細な情報が表示されます。

ページの上部にあるのは、イメージタグの名前です。その下は、このイメージタグに関連付けられている HM タグです。これには、EDB 公式タグ、プラットフォームによって生成された透過的データ暗号化タグ、および作成したタグが含まれる場合があります。**Add Tag** を選択して、イメージタグにタグを追加または削除し、必要に応じて新しいタグを作成します。

次は次のとおりです。

- **README** イメージリポジトリのテキスト説明。これには、画像の使用方法、含まれる内容、およびその他の役立つ詳細など、画像に関する関連情報が含まれます。
- 次の情報を表示するイメージ属性パネル
  - **バージョン** イメージタグのバージョン
  - **ベースイメージ** イメージの基になる OS イメージ
  - **イメージ SHA** : image.
  - の SHA256 ハッシュリポジトリへ。
  - **Created** イメージタグが作成された日時。

- **プラットフォームアーキテクチャ**イメージタグが使用可能なプラットフォームアーキテクチャ、X86 または AMD64 のいずれか。
- **Transparent Data Encryption** イメージタグが透過的データ暗号化 TDE をサポートするかどうかを示します。

Deployment Locations というラベルのフィールドは、イメージタグが利用可能な場所をリストしています。

表には、このイメージタグに含まれる拡張子が示されています。各拡張機能は、名前、バージョン、および説明とともに表示されます。**Search** フィールドを使用して、名前または説明で特定の拡張機能を検索します。このテーブルを名前でソートできます。

このイメージタグを使用しているクラスターセクションには、このイメージタグを使用しているエステート内のクラスターがリストされています。

Manage repositories in the asset library

基になるリポジトリへの接続はシステム管理者が構成しますが、アセットライブラリで使用可能なリポジトリを管理できます。

これは、特定のパターンに一致するリポジトリとタグ付けされたイメージをシステムに含められるようにすることで実現できます。各パターンはルールとして定義されます。次に、どのリポジトリをアセットライブラリに含めるか、およびどのイメージをハイブリッドマネージャー HM クラスターで使用できるかを制御できます。

デフォルトでは、アセットライブラリには、すべての公式 HM リポジトリと、それらのリポジトリ内のすべてのタグ付け画像が含まれます。

アセットライブラリで使用可能なリポジトリとイメージをより詳細に制御するには、プライベートレジストリでホストできる独自のプライベートリポジトリを管理できます。これにより、レジストリに含まれる内容を完全に制御でき、独自のイメージを追加して管理できます。どのルートを選択しても、アセットライブラリのリポジトリ包含ルールが動作して、アセットライブラリで使用可能なものを制御します。

リポジトリの管理

アセットライブラリの **Container Images** タブで、**Manage Repositories** を選択します。 **Manage Repositories** ページを使用して、アセットライブラリで使用可能なリポジトリ登録を追加、編集、または削除します。

現在アセットライブラリに登録されているリポジトリのテーブルには、次の列があります。

- **Repository Name Regex** リポジトリの名前が一致する必要がある正規表現。
- **ルールの数** このリポジトリ登録に関連付けられているルールの数。
- **Created** リポジトリ登録が作成された日時。

データが入力されると、各行には、リポジトリ登録を編集するための **Edit** ボタンと、リポジトリ登録を削除するための **Delete** ボタンが含まれます。

検索フィールドを使用して、特定のリポジトリ登録を名前で検索します。ソートコントロールを使用して、リポジトリ名または作成日でテーブルをソートします。

画面の右上隅にある **Register New Repository** を選択して、リポジトリ登録を追加します。

新しいリポジトリの登録

Register New Repository を選択して、新しいリポジトリ登録の詳細を入力できるダイアログボックスを開きます。

- **Repository Name Regex** リポジトリの名前と一致する正規表現。これは、アセットライブラリに含まれるリポジトリをフィルタリングするために使用されます。

このフィールドは、アセットライブラリに含めるリポジトリの名前と一致する必要があります。たとえば、`edb-postgres-advanced-` で始まるすべてのリポジトリを含める場合は、正規表現として`^edb-postgres-advanced-.*$` を入力します。プレーンテキストを入力できますが、正規表現として扱われます。正規表現を使用して、リポジトリ名が目的のパターンと一致することを確認することをお勧めします。

- **ルール**このリポジトリ登録に関連付けられているルールのリスト。各ルールは、画像に関連付けられているタグと一致する正規表現で構成されています。ルールは、アセットライブラリに含まれる画像をフィルタリングするために使用されます。たとえば、タグ `latest` を持つすべての画像を含める場合は、正規表現として`^latest$` と入力します。プレーンテキストを入力できますが、正規表現として扱われます。正規表現を使用して、タグが目的のパターンと正確に一致することを確認することをお勧めします。

ルールを追加するには、 **Add Rule** を選択します。これにより、ルールのリストに新しい行が追加されます。ルールを削除するには、ルールの横にある **Delete** を選択します。

新しいリポジトリ登録の詳細を入力したら、[**Register Repository**] を選択して登録を保存します。新しいリポジトリの登録は、登録済みリポジトリのリストに追加され、アセットライブラリで利用できるようになります。システムはリポジトリをスキャンして、定義したルールと一致するイメージをスキャンする必要があるため、これには数分かかる場合があります。

操作が完了すると、 **Manage Repositories** ページに戻ります。登録済みリポジトリのリストに新しいリポジトリの登録が表示されます。

Asset library に戻って、新しいリポジトリールールの効果と使用可能になったイメージを確認します。

Knowledge bases in the asset library

アセットライブラリの **ナレッジベース**ページでは、ハイブリッドマネージャー HM クラスターで使用できる AI モデルのコレクションを管理できます。

Knowledge Bases ページは、検索とフィルターバー、並べ替えコントロール、およびナレッジベースのテーブルで構成されています。各ナレッジベースは独自の行に表示され、次の情報が表示されます。

- **Name** ナレッジベースの名前。
- **ソースデータ** ナレッジベースの構築の元になるソースデータ。
- **データベース** ナレッジベースの構築元のデータベース。
- **クラスター** ナレッジベースが構築されるクラスター。
- **モデル** ナレッジベースの構築に使用される AI モデル。

Backups and restores

バックアップと復元の基本的なタスクは次のとおりです。

- *HM バックアップと DR*
- *Backups and restores*

Backups

Hybrid ManagerHM は、さまざまなテクノロジーを使用したコンポーネントのバックアップおよびリカバリー機能を備えています。2 つのバックアップ戦略を提供します。

- Postgres バックアップ データをバックアップします
- HM バックアップ HM システムのコア機能をバックアップします

Postgres バックアップデータプレーン

Postgres バックアップは、Postgres データベースに保存されているユーザーデータの保護に焦点を当てています。

- ベースバックアップは、すべての HM 展開で利用できます。
- スナップショットバックアップは、クラウドサービスプロバイダー CSP 環境と、適切なストレージとドライバーが構成されているオンプレミス展開で利用できます。

Postgres バックアップは、ログ先行書き込み WAL アーカイブなどの堅牢なデータベース機能を活用して、きめの細かいポイントインタイムリカバリー PITR を提供します。このアプローチでは、きめの細かい目標リカバリーポイント RPO が可能になります。

> **注釈**
>
> ローカルストレージは、スナップショットバックアップで現在サポートされていません。

HM バックアップコントロールプレーン

HM バックアップは、メタデータやシステムデータベースを含む HM システムを保護します。これらは別のスケジュールで動作し、Postgres バックアップとは異なるバックアップファイルを生成します。HM バックアップはファイルシステムベースであり、デフォルトでメタデータの時間ごとのバックアップとシステムの Postgres データベースのバックアップを行います。

Postgres バックアップとは異なり、HM バックアップは、WAL アーカイブを介した継続的なバックアップを提供しません。したがって、復旧は時間ごとまたはユーザー定義のバックアップ間隔に制限されます。

主な違いとリカバリーの影響

これらの違いにより、RPO と目標復旧時間 RTO は、ユーザーデータ Postgres と HM システムで異なります。

- Postgres データは、PITR および WAL アーカイブにより強力な RPO 機能を提供し、非常に最近のポイント・インタイムへのリカバリーを可能にする可能性があります。

- HM システムの場合、RPO はユーザー変更可能なバックアップスケジュールに制限されます。デフォルトでは時間ごとに。

Component	RPO	RTO	Method	Location
HM	1 hour (default)	30 mins or less	Velero	Object storage
Postgres data plane	5 minutes or less	Based on data size, backup method, and hardware.	Snapshot or base backup	Object storage

データサイズ

ベースバックアップを使用する Postgres クラスターの場合、500 GB 未満のデータサイズが有効なサイズ制限であると予想されます。ハードウェアのパフォーマンスによっては、最大 1 TB になる場合があります。

スナップショットバックアップを使用する Postgres クラスターの場合、1 TB を超えるデータサイズが、サービス品質に悪影響を与えるバックアップおよび復元プロセスなしでサポートされる予定です。

バックアップの可搬性とディザスターリカバリー DR

最も単純な構成では、たとえば TopoLVM バージョン 1.1 の GA では使用できません。スナップショットのバックアップとリカバリーはローカルのみであることが想定されており、スナップショットバックアップを別のアプライアンス、データセンター、または場所に移動できないことを意味します。この制限により、スナップショットの最も基本的な実装を使用した DR が効果的に除外されます。

HM がエンタープライズストレージと、スナップショットをサポートする CSI ドライバーで構成されている場合、HM はその機能を利用してクロスロケーション DR を提供できます。

スナップショットの可用性

ボリュームスナップショットは、Kubernetes 互換のボリュームスナップショット機能を提供するすべての環境でサポートされています。

スナップショットのトランスポート性

スナップショットのトランスポート可能性は、特定の CSI ドライバー実装に依存します。

AWS の詳細

- デフォルトでは、AWS ボリュームスナップショットはトランスポートできません。
- 同じ AWS リージョン内の新しいホストへの復元は可能ですが、クロスリージョンの復元には追加の手動手順が必要です。

TopoLVM の詳細

TopoLVM スナップショットはトランスポート不可で、別のホストに復元できません。

構成

次のストレージ、オブジェクトストレージ、およびバックアップを構成できます。

- シングルノードクラスター

・プライマリ/スタンバイクラスター

- EDB Postgres 分散 PGD クラスター

シングルノードクラスター

シングルノードクラスターのバックアップには、次の特徴がある場合があります。

- シングルノードクラスターは、HM を使用して展開されます。
- 保持期間とバックアップ時間を変更するか、デフォルト値を使用できます。
- 自動バックアップは、クラスター展開後、毎日発生します。
- 手動バックアップを実行できます。

プライマリ/スタンバイクラスターのバックアップ

プライマリ/スタンバイクラスターのバックアップには、次の特徴がある場合があります。

- プライマリ/スタンバイクラスターは、HM を使用して展開されます。
- 保持期間とバックアップ時間を変更するか、デフォルト値を使用できます。
- 自動バックアップは、クラスター展開後、毎日発生します。
- 手動バックアップを実行できます。
- バックアップはレプリカプライマリ/セカンダリから取得されます。

バックアップ PGD クラスター

PGD クラスターのバックアップには、次の特徴がある場合があります。

- PGD クラスターは、HM を使用して展開されます。
- 保持期間とバックアップ時間を変更するか、デフォルト値を使用できます。
- 自動バックアップは、クラスター展開後、毎日発生します。
- 手動バックアップを実行できます。
- バックアップはレプリカプライマリ/セカンダリから取得されます。

HM バックアップ

HM バックアップには、HM システムを復元するために必要なメタデータが含まれています。ユーザーデータベースのバックアップは含まれません。HM バックアップは、Velero を使用してオブジェクトストレージにバックアップします。デフォルトのバックアップスケジュールは 1 時間ごとです。スケジュールを調整できます。

デフォルトのスケジュールを使用する場合、RPO は 1 時間です。これは、クラッシュが発生した場合に最大 1 時間の HM データが失われることが予想されることを意味します。ユーザーデータベースは Postgres の継続的なバックアップで保護されているため、これはユーザーデータベースレベルでのデータ損失を意味するものではありません。

Postgres のバックアップの取得

HM コンソール、API、または CLI を使用して、Postgres バックアップを取得できます。API または CLI を使用して Postgres バックアップを取得するには、HM コンソールの API ドキュメントリンクを参照してください。

HM コンソールを使用して Postgres バックアップを取得するには

1. **Backups** タブを選択して、既存のバックアップを表示します。
2. **Create Backup** を選択します。
3. 場所、バックアップ名、およびバックアップ方法を入力し、 **OK** を選択します。

HM バックアップの取得

HM システムのバックアップを構成するには、*Hybrid Manager HA/DR recovery* を参照してください。

用語

次の用語は、Postgres および HM でのバックアップとリカバリーに使用されます。

バックアップ

元の場所からオブジェクトストレージのようなセカンダリの場所への 1 つ以上のファイルのコピー。バックアップは、コンポーネントタイプたとえば、コントロールプレーンとデータプレーンなど、展開タイプたとえば、CSP、エンジニアドシステム、または RHOS などによって異なります。

バックアップ保持期間

データのバックアップコピーが削除または上書きされるまでに保存される時間。バックアップを復旧のために利用できる期間を定義します。

Barman

主に EDB によって維持およびサポートされている open-source Postgres backup tool 。単独で使用される場合、Barman という用語は、 *Barman* **クラウドユーティリティスクリプト** ではなく、Barman サーバーまたはコアバックアップツールを指します。Barman は、エンタープライズクラスのバックアップとリカバリーに必要な Postgres のバックアップとリカバリープリミティブの多くを自動化します。Barman は、インクリメンタルバックアップ Postgres が提供したブロックレベルのインクリメンタルバックアップに対して、

RPO=0 またはほぼゼロを達成するための WAL ストリーミング、バックアップレポート、自動バックアップファイル保持管理、その他の便利な機能などの革新的な機能を提供します。

Barman クラウドユーティリティスクリプト

Barman サーバーとは別に使用できる collection of scripts 。彼らの目的は、Postgres バックアップを S3 互換オブジェクトストレージにコピーすることです。クラウドの Barman は、CNP オペレーターと EDB Cloud Service が Postgres バックアップをオブジェクトストレージにコピーするために使用されます。

ベースバックアップ

Postgres バックアップユーティリティ pg_basebackup は、Postgres インスタンスの復元に必要なファイルのバックアップまたはファイルコピーを自動化します。

クラウド

パブリッククラウドプロバイダーつまり AWS、Microsoft Azure、および Google Compute PlatformGCP。

クラスター

単一の Postgres インスタンス Postgres クラスターとしても知られるデータベースのコレクション。HM では、展開タイプたとえば、シングルノード、PSR、PGD クラスターなども参照します。

クラウドネイティブ PostgresCNP

open-source Postgres operator EDB によって開発および保守されています。HM の Kubernetes での Postgres の基盤を提供します。

Postgres クラスター

Postgres インスタンスは、Postgres クラスターを管理します。データベースクラスターは、共通のファイルシステムの場所データディレクトリに保存されているデータベースのコレクションです。

pg_dump

データのエクスポートおよびインポートユーティリティ *dump* とも呼ばれます。データベースの一部または全体をエクスポートする機能を提供しますが、エンタープライズクラスのスケーラブルなバックアップツールではありません。ただし、pg_dump は、テストデータベースの作成、データのアーカイブ、データベースから別のデータベースまたは DBaaS プロバイダーへの移行など、さまざまな目的でデータをエクスポートするために広く使用されています。HM ユーザーは pg_dump を使用して Postgres データにアクセスします。pg_dump ユーティリティは Postgres クライアントとともにインストールされ、ユーザーのクライアントマシンから使用できます。

リカバリー

バックアップデータファイルと WAL ファイルに記録された変更を適用することにより、復元データを一貫性のある最新の状態にするプロセス。

目標リカバリポイント RPO

リカバリーに使用できるデータベースタイムラインのポイントまたは時間たとえば 5 分。これは、一般的に「どれくらいのデータを失ってもよいですか?」という質問に答えるものだと言われています。RPO は、時間的に測定された、中断後に組織が耐えることができるデータ損失の最大許容量です。システムとデータを復旧する必要がある時点を定義します。

目標復旧時間 RTO

中断後に組織が耐えることができるダウンタイムの最大許容量。システムとデータを復元する必要がある目標時間を定義します。たとえば、実稼働データベースの RTO は 30 分である場合があります。これは、復元とリカバリーを実行するダウンタイムが 30 分を超えてはいけないことを意味します。

リストア

復旧の準備として、バックアップ場所から元の場所または代替の場所にファイルをコピーするプロセス。例は、バックアップファイルをオブジェクトストレージからデータベースサーバーにコピーすることです。

スナップショット

データのボリュームレベルのスナップショットまたはファイルシステムレベルのバックアップ。スナップショットは、使用可能な最も速く、最もスケーラブルなバックアップおよびリカバリー方法です。スナップショット機能は、HM 展開タイプとトランスポート性などのストレージ機能の可用性によって異なります。

トランスポータブルボリュームのスナップショット

別のストレージシステム、場所、またはクラウドリージョンに移動またはコピーできるデータのポイントインタイムコピー。このコピーにより、ディザスターリカバリー、データ移行、およびオフサイトバックアップが可能になります。

トランスポート不能なボリュームのスナップショット

特定のストレージシステムまたはデータが作成された場所に制限されるデータのポイントインタイムコピー。これは、主にそのシステムのローカルデータの復旧に使用されます。

Restores

Postgres バックアップの復元

次を使用して復元できます。

- Hybrid ManagerHM コンソール
- API
- CLI

HM コンソールを使用した復元

1. **Backups** タブを選択して、復元可能な既存のバックアップを表示します。
2. 復元するバックアップを選択します。
3. 復元するクラスタータイプを選択します。
4. オプションで、リストアを構成します。
   - **クラスター設定**で、リストアされたクラスターの一意の名前を指定します。
   - **リストアタイプ** の場合、**完全なリストア** または**ポイントインタイムリストアを選択します。**ポイントインタイムリストア**を選択した場合は、日付ピッカーを使用して、復元するポイントを選択します。-** DB 構成 **および**追加設定**では、優先するデータベース構成を選択します。
5. **Restore Cluster** を選択します。復元されたクラスターは、コンソールの **クラスター**リストに表示されます。

API を使用した復元

API を使用して復元するには、次のものが必要です。

- 復元構成を指定する JSON ファイル。
- カールを使用して JSON ファイルを呼び出すシェルスクリプト。

> **Tip**
>
> EDB GitHub repository でポイントインタイムリストアのファイルテンプレートを取得できます。リストアのテンプレートには、*pitr_* プレフィックスが付いています。

API を使用してポイント インタイムに復元するには

1. JSON ファイルで、次のパラメーターを提供します。

`projectID` クラスターの URL から `projectID` を取得します。たとえば、URL が https://portal/projects/prj_6EOsaHpfS8RLshWH/clusters の場合、`projectID` は `prj_6EOsaHpfS8RLshWH` です。

`source_cluster_id` クラスターの URL から `source_cluster_id` を取得します。たとえば、URL が https://portal/projects/prj_6EOsaHpfS8RLshWH/clusters/p-nubw4gnbv9>の場合、`source_cluster_id` は `p-nubw4gnbv9` です。

`point_in_time` クラスターを復元するポイント・インタイムのタイムスタンプ値を提供します。

復元するクラスターに必要に応じて、他の構成パラメーターの追加情報を提供します。例

1. JSON ファイルを呼び出すシェルスクリプトを作成します。 例

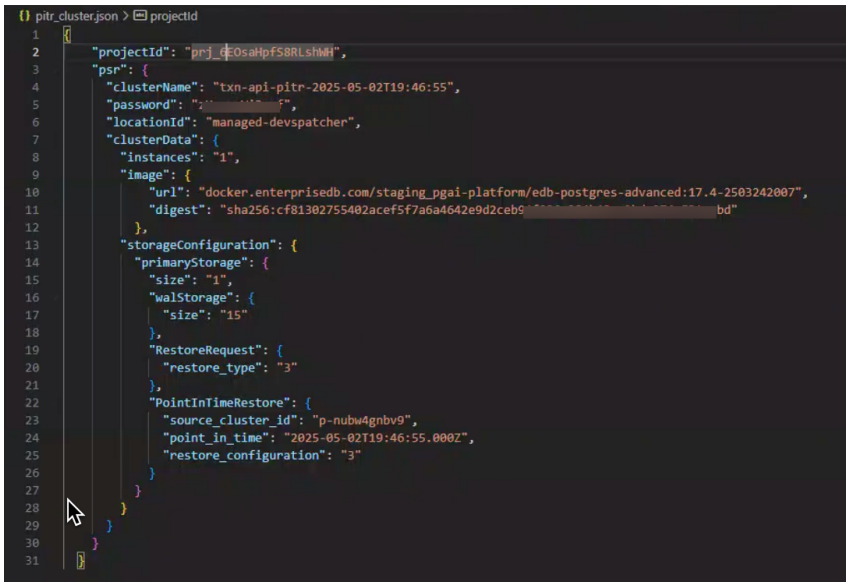
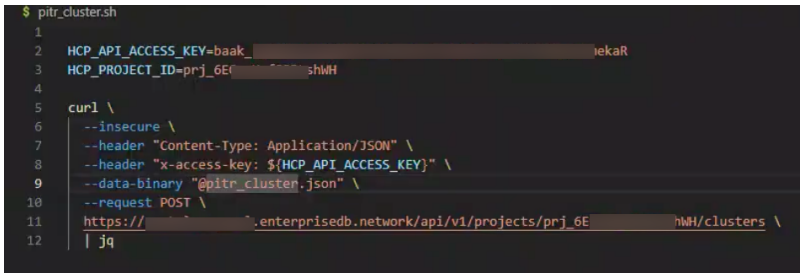


図 8: PGAIHM Restore via API JSON file



2. シェルスクリプトを実行します。スクリプトが出力としてクラスター ID を返した場合、復元されたクラスターはコンソールの**クラスター** リストに表示されます。例

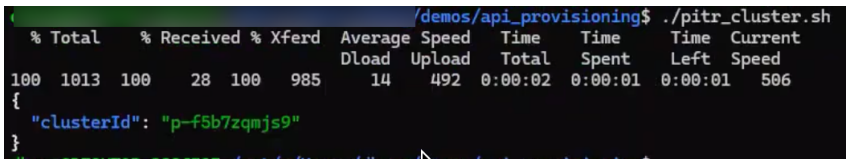


図 9: PGAIHM Restore via API shell file conf

CLI を使用した復元

CLI を使用してポイント インタイムに復元できます。CLI を使用して復元するには、次のものが必要です。

- リストア構成を指定する YAML ファイル。
- CLI で `edbctl` を使用して YAML ファイルを呼び出すには。

1. YAML ファイルで、次のパラメーターを提供します。

> **Tip**
>
> resume コマンドで使用できるパラメーターの詳細については、 CLI で `edbctl cluster restore -h` を使用

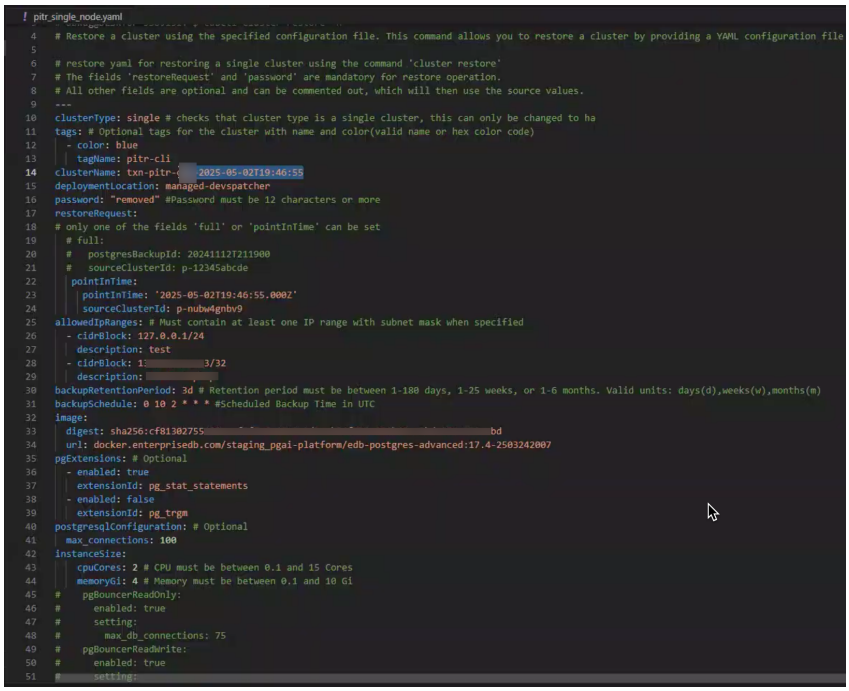


図 10: HM Restore via CLI YAML file

してヘルプを表示します。

1. YAML ファイルを指定して CLI コマンドを実行します。クラスターの復元が開始されたことを示すメッセージが返されます。例

```
~$ edbctl cluster restore -F $DEMOS/cli_pitr/pitr_single_node.yaml
```

図 11: PGAIHM Restore via CLI YAML file

Hybrid Manager CLI - edbctl

edbctl は、ハイブリッドマネージャー HM リソースを管理し、データベースクラスター操作を実行するためのコマンドラインツール CLI です。

前提条件

Hybrid Manager CLI は、ほとんどの Linux ディストリビューションで利用できます。edbctl を使用するには、`pgai-platform` リポジトリからパッケージをインストールする必要があります。

EDB サブスクリプショントークンを取得する

`pgai-platform` リポジトリには、ダウンロードを有効にするトークンが必要です。トークンを取得するには、EDB Repos 2.0 にログインします。 EDB Repos 2.0 ページに初めてアクセスする場合は、 **Request Access** を選択してトークンを生成する必要があります。生成されたトークンが利用できるようになったら、**Copy** アイコンを選択してクリップボードにコピーするか、目のアイコンを選択して表示します。

EDB_SUBSCRIPTION_TOKEN 環境変数を設定する

トークンを取得したら、オペレーティングシステムに表示されているコマンドを実行して、<your-token> トークンを置き換えます。

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

pgai-platform リポジトリからのインストール

オペレーティングシステムに合わせて適切なコマンドを実行します。

Debian または Ubuntu

Debian または Ubuntu では、apt パッケージマネージャーを使用して HM CLI をインストールできます。

```
curl -1sSLf "https://downloads.enterprisedb.com/$EDB_SUBSCRIPTION_TOKEN/pgai-platform/
↪setup.deb.sh" | sudo -E bash
```

このコマンドが curl: (22) The requested URL returned error: 404 のようなエラーを返した場合、正しいトークンを入力したことを確認してください。

コマンドが成功すると、次のような出力が表示されます。

```
Executing the  setup script for the enterprisedb/pgai-platform repository ...
...
```

次のコマンドを使用して edbctl パッケージをインストールできるようになりました。

```
sudo apt-get install edbctl
```

RHEL、Rocky、AlmaLinux、または Oracle Linux

RHEL、Rocky、AlmaLinux、または Oracle Linux では、yum パッケージマネージャーを使用して HM CLI をインストールできます。 RHEL 8 以降のデフォルトパッケージマネージャーである dnf パッケージマネージャーを使用することもできます。

```
curl -1sSLf "https://downloads.enterprisedb.com/$EDB_SUBSCRIPTION_TOKEN/pgai-platform/
↪setup.rpm.sh" | sudo -E bash
```

このコマンドが curl: (22) The requested URL returned error: 404 のようなエラーを返した場合、正しいトークンを入力したことを確認してください。

コマンドが成功すると、次のような出力が表示されます。

```
Executing the  setup script for the enterprisedb/pgai-platform repository ...
...
```

次のコマンドを使用して edbctl パッケージをインストールできるようになりました。

```
sudo dnf install edbctl
```

```
sudo yum install edbctl
```

HM CLI の設定

パッケージをインストールした後、 credential コマンドを使用して *Access key* を構成します。これらの資格情報により、サポートされている HM リソースにアクセスし、CLI を使用して管理できます。CLI は、プロジェクト、クラスター、バックアップ、資格情報などを管理するためのさまざまなコマンドをサポートしています。

構成例

```
edbctl credential import-access-key --name user1 --access-key your_HM_access_key --
↪address https://YOUR_HM_INSTANCE_URL:443
edbctl config set context_project "Your Project Name"
```

オプション

edbctl --help or -h コマンドを使用して、コマンドの完全なリストを結果ます。

```
edbctl --help
__OUTPUT__
Usage:
  edbctl [options]
  edbctl [command]

Management Commands:
  credential        Manage authentication credentials.
  project           Manage projects associated with the database clusters.
  cluster           Manage database cluster resources.
  backup            Manage cluster backups.

Configuration Commands:
  config            Manages configuration settings for edbctl tool
  help              Help about any command
  completion        Generate the autocompletion script for the specified shell

Setup Commands:
  image             Asset management commands
  operand           Operand image management commands
  image-pull-secret Pull Secret management commands

Options:
  -h, --help      help for edbctl
  -v, --version   version for edbctl
```

(次のページに続く)

(前のページからの続き)

```
Use "edbctl [command] --help" for more information about a command.
```

詳細情報

- *edbctl credential* — 認証資格情報を管理します。
- *edbctl image* — 資産管理コマンド
- *edbctl image-pull-secret* — シークレット管理コマンドをプルします。
- *edbctl operand* — オペランドイメージ管理コマンド。
- *edbctl project* —データベースクラスターに関連付けられたプロジェクトを管理します。
- *edbctl cluster* — データベースクラスターリソースを管理します。
- *edbctl backup* — クラスターバックアップを管理します。
- *edbctl config* — edbctl ツールの構成設定を管理します。

edbctl backup

このユーティリティは、クラスターバックアップを管理します。コマンドを使用して、クラスターのバックアップを作成、削除、およびリストします。サブコマンドを使用して、特定のバックアップ操作を実行します。

コマンド

- *edbctl* **バックアップリスト** — 使用可能なすべてのバックアップのリストを表示します。
- *edbctlbackup create* — 指定されたクラスターのバックアップを作成します。
- *edbctlbackup delete* — 指定されたクラスターのバックアップを削除します。

edbctl バックアップリスト

指定されたクラスターで使用可能なすべてのバックアップをリストします。バックアップ ID、名前、開始時刻、場所、ステータス、タグを含む詳細情報を表示します。

```
edbctl backup list [options]
```

options は次のとおりです。

```
    --cluster-id string   Cluster ID (required). The immutable unique identifier for␣
↪a cluster.
    --location string     Location ID.
- c, --credential string   The credential which you created via the `credential␣
↪create` command. The default is fetched from context_credential.
- P, --project string      The project that groups your clusters and other resources.␣
↪The default is taken from context_project.
```

(次のページに続く)

(前のページからの続き)

```
- o, --output string       [table json yaml xml] (default "table")
- h, --help                Help for list.
```

この例は、クラスターのバックアップをリストする方法を示しています。

```
edbctl backup list
```

edbctlbackup create

指定されたクラスターのバックアップを作成します。クラスター ID、バックアップ名、場所 ID、およびオプションのタグを指定して、バックアップを開始します。

```
edbctl backup create [options]
```

options は次のとおりです。

```
    --cluster-id string   Cluster ID (required). The unique identifier for the␣
↪cluster to back up.
    --method string       Backup method (required). Specify the backup method: BARMAN_
↪CLOUD or VOLUME_SNAPSHOT.
    --name string         Backup name (required). The name of the backup to create.
    --location string     Location ID (required).
    --group-id string     PGD cluster group ID to back up. Required for PGD clusters.
    --tags strings        Assign tags to the backup in the format name=color (color␣
↪is optional).
- c, --credential string   The credential that you created via `credential create`␣
↪command. The default is fetched from `context_credential`.
- P, --project string      The project that groups your clusters and other resources,␣
↪the default is taken from `context_project`
- y, --yes                 auto-confirm all confirmations
- h, --help                help for create
```

これらの例は、クラスターのバックアップを作成する方法を示しています。

```
edbctl backup create --cluster-id p-8nllnskncq --name "My Backup" --method "BARMAN_
↪CLOUD" --location "managed-devspatcher"
edbctl backup create --cluster-id p-8nllnskncq --name "My Backup" --method "BARMAN_
↪CLOUD" --location "managed-devspatcher"  --tags name2=blue
edbctl backup create --cluster-id p-8nllnskncq --name "My Backup" --method "VOLUME_
↪SNAPSHOT" --location "managed-devspatcher"  --tags name2=#FF0000
```

edbctlbackup delete

バックアップ ID を使用して、指定されたクラスターからバックアップを削除します。このアクションは元に戻せないため、続行する前に正しいバックアップ ID があることを確認してください。

```
edbctl backup delete {--id} [options]
```

次の例は、バックアップを削除する方法を示しています。

```
edbctl backup delete --id my_backup_id --cluster-id p-8nllnskncq
```

options は次のとおりです。

```
- i, --id string           Backup ID (required). The immutable unique identifier for␣
↪the clusters backup.
    --cluster-id string   Cluster ID (required). The immutable unique identifier for␣
↪a cluster.
- c, --credential string   The credential that you created via the `credential␣
↪create` command. The default is fetched from `context_credential`.
- P, --project string      The project that groups your clusters and other resources.␣
↪The default is taken from `context_project`.
- y, --yes                 auto-confirm all confirmations.
- h, --help                Help for delete.
```

edbctl catalog

edbctl catalog コマンドは、Hybrid Manager クラスター内のデータベースカタログリソースの管理に関連するすべての操作のエントリポイントです。データベースカタログのリスト、追加、更新、削除などのさまざまなタスクを実行できます。

コマンド

次のコマンドは edbctl catalog 名前空間に該当し、特定のカタログ管理機能を提供します。

- *edbctlcatalog create* 新しいデータベースカタログリソースをプロビジョニングおよび登録するために使用されます。
- *edbctl* **カタログ削除** 指定されたカタログを完全に削除するために使用されます。
- *edbctl* **カタログ編集**既存のカタログの構成またはメタデータを変更するために使用されます。
- *edbctl* **カタログリスト**プロジェクトで現在展開されているまたは使用可能なすべてのカタログのリストを表示するために使用されます。

edbctlcatalog create

edbctl catalog create コマンドは、ハイブリッドマネージャークラスター内の新しいデータベースカタログリソースをプロビジョニングおよび登録します。このコマンドは、リソースグループ化のプロジェクトカタログと、EDB Postgres 分散 PGD および Advanced High AvailabilityAHA クラスター専用のクラスターカタログの両方の作成をサポートし、組織とメタデータのカスタムタグを割り当てる機能を備えています。

```
edbctl catalog create [options]
```

options は次のとおりです。

```
    --type string               Type of catalog to create. The valid values are␣
↪`project`, `pgd`, `aha`. The default is `project`  (required).
    --tags strings              Comma-separated list of tags to assign to the catalog␣
↪in `name=value` format, where `value` is optional.
    --cluster-id string         The unique ID of the cluster associated with the␣
↪catalog (required for `pgd` and `aha` cluster types).
    --datagroup-id string       The unique ID of the Data Group associated with the␣
↪catalog (required for `pgd` and `aha` cluster types).
    --database-name string      The name of the database associated with the catalog␣
↪(required for `pgd` and `aha` cluster types).
-P, --project string            The name of the project associated with the catalog␣
↪(required for `pgd` and `aha` cluster types).
-y, --yes                       Auto-confirms all prompts and confirmations, enabling␣
↪non-interactive operation.
-h, --help                      Displays help information for the `create` command.
```

この例は、デフォルトのプロジェクトカタログを作成し、それにタグを割り当てる方法を示しています。

```
edbctl catalog create --tags env=dev
```

この例は、特定のプロジェクト内でプロジェクトカタログを作成する方法を示しています。

```
edbctl catalog create --project "prj_123" --tags name1=blue --type project
```

この例は、特定のプロジェクト、クラスター、データグループ、およびデータベースに関連付けられた PGD クラスターカタログを作成する方法を示しています。

```
edbctl catalog create --tags env=prod --type pgd --cluster-id "p-123" --datagroup-id
↪"p-123-a" --database-name "mydb"
```

この例は、指定されたプロジェクト内で PGD タイプのカタログを作成する方法を示しています。

```
edbctl catalog create --project "prj_123" --tags env=prod --type pgd --cluster-id "p-
↪456" --datagroup-id "p-456-b" --database-name "bdrdb"
```

edbctl カタログ削除

`edbctl catalog delete` コマンドは、一意の識別子を提供することにより、登録されたカタログリソースから指定されたカタログを完全に削除するために使用されます。このアクションは元に戻すことができないため、注意して使用する必要があります。

このコマンドには、正しいリソースが削除されるように一意のカタログ ID が必要です。

```
edbctl catalog delete --id [options]
```

`options` は次のとおりです。

```
    --id string                The unique ID of the catalog to delete (required).
-P, --project string           The name of the project that groups your clusters and␣
↪other resources. The default is taken from the current configuration context␣
↪(required).
-c, --credential string        The name of the credential to use for authentication.␣
↪The default is taken from the current configuration context (required).
-y, --yes                      Auto-confirms all prompts and confirmations, enabling␣
↪non-interactive operation.
-h, --help                     Displays help information for the `delete` command.
```

この例は、一意の ID を提供してカタログを削除する方法を示しています。

```
edbctl catalog delete --id 11ee9cd2-7696-11f0-800c-b7b1f62bfc07
```

この例は、非対話型操作を使用して一意の ID を提供することにより、カタログを削除する方法を示しています。

```
edbctl catalog delete --id 11ee9cd2-7696-11f0-800c-b7b1f62bfc07 -y
```

edbctl カタログ編集

edbctl catalog edit コマンドは、主にタグおよびその他のメタデータを更新することにより、既存のカタログの構成を変更するために使用されます。

このコマンドには、正しいリソースが編集されることを保証するための一意のカタログ ID と、新しいタグを指定するための必須--tags オプションが必要です。

```
edbctl catalog edit [options]
```

options は次のとおりです。

```
-i, --id string                The unique ID of the catalog to edit (required).
    --tags strings             Comma-separated list of tags to assign to the catalog␣
↪in `name=value` format, where `value` is optional.
-P, --project string           The name of the project that groups your clusters and␣
↪other resources. The default is taken from the current configuration context␣
↪(required).
-c, --credential string        The name of the credential to use for authentication.␣
↪The default is taken from the current configuration context (required).
-I, --interactive              Enables interactive mode for editing.
-y, --yes                      Auto-confirms all prompts and confirmations, enabling␣
↪non-interactive operation.
-h, --help                     Displays help information for the `edit` command.
```

重要

--tags オプションは置換ベースです。コマンドに含まれないカタログ上の既存のタグは削除されます。既存のタグを維持するには、追加する新しいタグとともにそれらを明示的にリストする必要があります。すべてのタグを削除するには、空の文字列 *--tags ""* を使用します。

この例は、一意の ID を提供することにより、既存のすべてのタグをカタログの新しいリスト env=blue および team に置き換える方法を示しています。

```
edbctl catalog edit --id 12c4-7c4a-11f0-932a-cd765vg --tags env=blue,team
```

この例は、一意の ID を提供することにより、カタログからすべてのタグを削除する方法を示しています。

```
edbctl catalog edit --id 12c4-7c4a-11f0-932a-cd765vg --tags ""
```

edbctl カタログリスト

edbctl catalog list コマンドは、登録されているすべてのカタログのリストを表示するために使用されます。このコマンドは、カタログタイプ、ID、クラスター情報、およびスコープ現在のプロジェクトまたは不動産全体に基づいてフィルタリングオプションを提供します。

デフォルトでは、このコマンドは、現在のプロジェクトコンテキスト内のすべてのカタログをリストします。

```
edbctl catalog list [options]
```

options は次のとおりです。

```
    --type string               Filter the list by catalog type. Valid values:␣
↪`project`, `pgd`.
-i, --id string                 The unique ID of a specific catalog to display.
    --cluster-id string         Filter PGD-type catalogs by a comma-separated list of␣
↪Cluster IDs.
    --datagroup-id string       Filter PGD-type catalogs by a comma-separated list of␣
↪Data Group IDs.
    --estate                    If present, lists catalogs across all projects in the␣
↪estate, rather than just the current project. You will see catalogs for projects␣
↪where they have a role.
-P, --project string            Specifies the project to query. The default project␣
↪is taken from the current configuration context.
-o, --output string             Sets the output format. The valid values are `table`␣
↪(default), `json`, `yaml`, `xml`, and `json-raw`.
-h, --help                      Displays help information for the `list` command.
```

これらの例は、 edbctl catalog list コマンドのさまざまなオプションを使用する方法を示しています。

```
#  List all catalogs in the current project context:

edbctl catalog list
```

(次のページに続く)

(前のページからの続き)

```
#  Filter the list to show only catalogs of type `project`:

edbctl catalog list --type project

#  Filter the list to show only catalogs of type `pgd`:

edbctl catalog list --type pgd

#  List the details of a single catalog by its unique ID:

edbctl catalog list --id "12c4-7c4a-11f0-932a-cd765vg"

#  List catalogs specifically in the project with ID `prj_123`:

edbctl catalog list --project "prj_123"

#  List catalogs across all projects in the entire estate:

edbctl catalog list --estate

#  List only PGD catalogs associated with the cluster ID `p-123`:

edbctl catalog list --type pgd --cluster-id "p-123"

#  List catalogs in JSON format instead of the default table view:

edbctl catalog list --output json
```

edbctl cluster template

`edbctl cluster-template` コマンドグループは、事前定義されたクラスターテンプレートを管理するために必要なツールを提供します。これらのテンプレートはブループリントとして機能し、構成を標準化し、環境全体で一貫した設定でデータベースクラスターを迅速に展開できます。

コマンド

- *edbctl cluster-template create* 新しいデータベースクラスターテンプレートを定義および登録するために使用されます。
- *edbctl cluster-template list* プロジェクトまたはエステート内で使用可能なすべてのクラスターテンプレートを表示するために使用されます。

edbctl cluster-template create

edbctl cluster-template create コマンドを使用すると、データベースクラスター構成を再利用可能なテンプレートとして定義および保存できます。YAML 構成ファイルを提供することにより、シングル、HA、PGD、分析クラスターを含む複雑なアーキテクチャを標準化して、組織全体で一貫した展開を保証できます。

このコマンドを使用して、クラスターの技術要件 CPU、メモリ、ストレージ、バックアップ、Postgres 構成を指定するローカル YAML ファイルを指定することにより、新しいテンプレートを登録します。

```
edbctl cluster-template create [options]
```

options は次のとおりです。

```
-F, --config-file string        Path to the cluster configuration YAML file␣
↪(required). For more details, see [Configuration file structure](#configuration-
↪file-structure).
-n, --name string               Name of the cluster template (required).
-d, --description string        A brief summary of the templates purpose or␣
↪configuration.
-P, --project string            The project context. Defaults to `context_project`.
-c, --credential string         The name of the credential to use for authentication.␣
↪The default is taken from the current configuration context (required).
-y, --yes                       Auto-confirms all prompts and confirmations, enabling␣
↪non-interactive operation.
-h, --help                      Displays help information for the `create` command.
```

この例は、サンプル YAML 構成ファイルを使用してクラスターテンプレートを作成する方法を示しています。

単一ノードのサンプル YAML 構成ファイル

```
#  cluster details

clusterType: single # Required. Type of the cluster
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code)
  - color: blue
    tagName: tag-1
clusterName: create-single-name # Required.
password: password@1234 # Required. Password must be 12 characters or more

#  primary details

primaryPgName: Primary-Replica
imageId: 3e2af4f1dba3b9f8fb2bb6ae4834a5bd # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`
deploymentLocation: managed-devspatcher # Required.
```

(次のページに続く)

(前のページからの続き)

```
networking: private # Required. Network access type for the cluster. Valid values:␣
↪private, public
# allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#   - cidrBlock: 127.0.0.1/24

#     description: test

backupRetentionPeriod: 30d # Required. Retention period must be between 1-180 days, 1-
↪25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
backupSchedule: 39 13 23  # Required. Scheduled Backup Cron
# image:  # only one of `imageId` or `image` should be set

#   url: docker.example.com/platform/example/postgres:latest

#   digest: sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

postgresqlConfiguration: # Optional
  max_connections: 100
instanceSize: # Required.
  cpuCores: 1 # CPU must be between 0.1 and 15 Cores
  memoryGi: 1 # Memory must be between 0.1 and 10 Gi
storage:
  databaseStorage: # Required.
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
    storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field
  walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs (Recommended for high write workloads)
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
    storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
# transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.

#   passphrase: mypassphrase

#   keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4

#  replica details

replicaClusters:
  - replicaPgName: replica-1
    imageId: 3e2af4f1dba3b9f8fb2bb6ae4834a5bd # Image ID can be fetched with command␣
```

(次のページに続く)

(前のページからの続き)

```
↪`edbctl image list-image-tags`
    deploymentLocation: managed-devspatcher # Required.
    networking: private # Required. Network access type for the cluster. Valid␣
↪values: private, public
    #    allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified
    #      - cidrBlock: 127.0.0.1/24
    #        description: test
    backupRetentionPeriod: 30d # Required. Retention period must be between 1-180␣
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    backupSchedule: 39 13 23  # Required. Scheduled Backup Cron
    #    image: # only one of `imageId` or `image` should be set
    #      url: docker.example.com/platform/example/postgres:latest
    #      digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
    postgresqlConfiguration: # Optional
      max_connections: 100
    instanceSize: # Required.
      cpuCores: 1 # CPU must be between 0.1 and 15 Cores
      memoryGi: 1 # Memory must be between 0.1 and 10 Gi
    storage:
      databaseStorage: # Required.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field
      walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs (Recommended for high write workloads)
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
    #transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.
    #  passphrase: mypassphrase
    #  keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4
  - replicaPgName: replica-2 # Required.
    imageId: 3e2af4f1dba3b9f8fb2bb6ae4834a5bd # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`
    deploymentLocation: managed-devspatcher # Required.
    networking: private # Required. Network access type for the cluster. Valid␣
↪values: private, public
    #    allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified
    #      - cidrBlock: 127.0.0.1/24
    #        description: test
    backupRetentionPeriod: 30d # Required. Retention period must be between 1-180␣
```

(次のページに続く)

(前のページからの続き)

```
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    backupSchedule: 39 13 23  # Required. Scheduled Backup Cron
    #    image: # only one of `imageId` or `image` should be set
    #     url: docker.example.com/platform/example/postgres:latest
    #     digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
    postgresqlConfiguration: # Optional
      max_connections: 100
    instanceSize: # Required.
      cpuCores: 1 # CPU must be between 0.1 and 15 Cores
      memoryGi: 1 # Memory must be between 0.1 and 10 Gi
    storage:
      databaseStorage: # Required.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field
      walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs (Recommended for high write workloads)
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
    #transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.
    #  passphrase: mypassphrase
    #  keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4
```

この例は、サンプル YAML ファイルを使用して高可用性クラスターテンプレートを作成する方法を示しています。

HA クラスターのサンプル YAML 構成ファイル

```
#  cluster details

clusterType: ha # Required. Type of the cluster
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code)
  - color: blue
    tagName: tag-1
clusterName: create-ha-name # Required.
password: password@1234 # Required. Password must be 12 characters or more

#  primary details

primaryPgName: Primary-Replica
imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
```

(次のページに続く)

(前のページからの続き)

```
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
standbyReplicas: 2 # Required for HA cluster
deploymentLocation: managed-devspatcher # Required.
networking: private # Required. Network access type for the cluster. Valid values:␣
↪private, public
# allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#   - cidrBlock: 127.0.0.1/24

#     description: test

backupRetentionPeriod: 30d # Required. Retention period must be between 1-180 days, 1-
↪25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
backupSchedule: 39 13 23  # Required. Scheduled Backup Cron
# image: # only one of `imageId` or `image` should be set

#   url: docker.example.com/platform/example/postgres:latest

#   digest: sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

postgresqlConfiguration: # Optional
  max_connections: 100
instanceSize: # Required.
  cpuCores: 1 # CPU must be between 0.1 and 15 Cores
  memoryGi: 1 # Memory must be between 0.1 and 10 Gi
storage:
  databaseStorage: # Required.
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
    storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field
  walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs (Recommended for high write workloads)
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
    storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field
# transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.

#   passphrase: mypassphrase

#   keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4

#  replica details
```

(次のページに続く)

(前のページからの続き)

```
replicaClusters:
  - replicaPgName: replica-1
    standbyReplicas: 2 # Required for HA cluster
    imageId: 3e2af4f1dba3b9f8fb2bb6ae4834a5bd # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`
    deploymentLocation: managed-devspatcher # Required.
    networking: private # Required. Network access type for the cluster. Valid␣
↪values: private, public
    #    allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified
    #      - cidrBlock: 127.0.0.1/24
    #        description: test
    backupRetentionPeriod: 30d # Required. Retention period must be between 1-180␣
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    backupSchedule: 39 13 23  # Required. Scheduled Backup Cron
    #    image: # only one of `imageId` or `image` should be set
    #      url: docker.example.com/platform/example/postgres:latest
    #      digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
    postgresqlConfiguration: # Optional
      max_connections: 100
    instanceSize: # Required.
      cpuCores: 1 # CPU must be between 0.1 and 15 Cores
      memoryGi: 1 # Memory must be between 0.1 and 10 Gi
    storage:
      databaseStorage: # Required.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field
      walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs (Recommended for high write workloads)
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
    #transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.
    #  passphrase: mypassphrase
    #  keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4
  - replicaPgName: replica-2 # Required.
    standbyReplicas: 2 # Required for HA cluster
    imageId: 3e2af4f1dba3b9f8fb2bb6ae4834a5bd # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`
    deploymentLocation: managed-devspatcher # Required.
    networking: private # Required. Network access type for the cluster. Valid␣
↪values: private, public
```

(次のページに続く)

(前のページからの続き)

```
    #    allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified
    #      - cidrBlock: 127.0.0.1/24
    #        description: test
    backupRetentionPeriod: 30d # Required. Retention period must be between 1-180␣
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    backupSchedule: 39 13 23  # Required. Scheduled Backup Cron
    #    image: # only one of `imageId` or `image` should be set
    #     url: docker.example.com/platform/example/postgres:latest
    #     digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
    postgresqlConfiguration: # Optional
      max_connections: 100
    instanceSize: # Required.
      cpuCores: 1 # CPU must be between 0.1 and 15 Cores
      memoryGi: 1 # Memory must be between 0.1 and 10 Gi
    storage:
      databaseStorage: # Required.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field
      walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs (Recommended for high write workloads)
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
    #transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.
    #  passphrase: mypassphrase
    #  keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4
```

この例は、サンプル YAML 構成ファイルを使用して分析クラスターテンプレートを作成する方法を示しています。

アナリティクスのサンプル YAML 構成ファイル

```
clusterType: analytics # required, type of the cluster
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code)
  - color: blue
    tagName: tag-1
imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
clusterName: create-analytics-name # required
deploymentLocation: managed-devspatcher # required
```

(次のページに続く)

(前のページからの続き)

```
password: password@1234 #Password must be 12 characters or more
networking: private # Network access type for the cluster. Valid values: private,␣
↪public
# allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#   - cidrBlock: 127.0.0.1/24

#     description: test

backupRetentionPeriod: 30d # Required. Retention period must be between 1-180 days, 1-
↪25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
backupSchedule: 39 13 23  # Required. Scheduled Backup Cron
# image: # only one of `imageId` or `image` should be set

#   url: docker.example.com/platform/example/postgres:latest

#   digest: sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

postgresqlConfiguration: # Optional
  max_connections: 100
instanceSize: # Required
    cpuCores: 1 # CPU must be between 0.1 and 15 Cores
    memoryGi: 1 # Memory must be between 0.1 and 10 Gi
```

この例は、サンプル YAML 構成ファイルを使用して PGD クラスターテンプレートを作成する方法を示しています。

PGD のサンプル YAML 構成ファイル

```
clusterType: pgd # Type of the cluster
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code)
  - color: blue
    tagName: tag-1
clusterName: create-pgd-name # required
password: Password@1234 #Password must be 12 characters or more
dataGroups:
  - groupName: Data Group A # required
    deploymentLocation: managed-devspatcher # required
    imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
    networking: private # Network access type for the cluster. Valid values: private,␣
↪public
```

(次のページに続く)

(前のページからの続き)

```
#     allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#       - cidrBlock: 9.9.9.9/28

#         description: Allow traffic from App A

#       - cidrBlock: 10.10.10.10/27

#         description: Allow traffic from App B

    backupRetentionPeriod: 30d # Required. Retention period must be between 1-180␣
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    nodes: 3 # Required. Only can be set to 2 or 3

#     image: # only one of `imageId` or `image` should be set

#       url: docker.example.com/platform/example/postgres:latest

#       digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

    postgresqlConfiguration:
      max_connections: 100
      max_locks_per_transaction: 64
    backupSchedule: 27 26 0  # Required. Scheduled Backup Cron
    instanceSize: # Required
        cpuCores: 2 # CPU must be between 0.1 and 15 Cores
        memoryGi: 2 # Memory must be between 0.1 and 10 Gi. For PGD clusters, the␣
↪recommended memory is 2 Gi or more.
    storage:
      databaseStorage: # Required
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). For PGD␣
↪clusters, the recommended size is 10 Gi or more.
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
      walStorage:   # Optional, WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field

#     transparentDataEncryption: # Optional. Only one of passphrase or KeyId should␣
↪be set, adds a layer of security by encrypting supported data stored at rest.
```

(次のページに続く)

(前のページからの続き)

```
#       passphrase: mypassphrase

#       keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4

  - groupName: Data Group B
    deploymentLocation: managed-devspatcher # required
    imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
    networking: private # Network access type for the cluster. Valid values: private,␣
↪public

#     allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#       - cidrBlock: 9.9.9.9/28

#         description: Allow traffic from App A

#       - cidrBlock: 10.10.10.10/27

#         description: Allow traffic from App B

    backupRetentionPeriod: 30d  # Required. Retention period must be between 1-180␣
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    nodes: 3 # only can be set to 2 or 3

#     image: # only one of `imageId` or `image` should be set

#       url: docker.example.com/platform/example/postgres:latest

#       digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

    postgresqlConfiguration:
      max_connections: 100
      max_locks_per_transaction: 64
    backupSchedule: 6 27 0  # Required. Scheduled Backup Cron
    instanceSize: # Required
        cpuCores: 2 # CPU must be between 0.1 and 15 Cores
        memoryGi: 2 # Memory must be between 0.1 and 10 Gi. For PGD clusters, the␣
↪recommended memory is 2 Gi or more.
    storage:
      databaseStorage: # Required
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). For PGD␣
↪clusters, the recommended size is 10 Gi or more.
```

(次のページに続く)

(前のページからの続き)

```
      storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field
    walStorage: # Optional field, but default value is 1 Gi
      sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
      storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field

#     transparentDataEncryption: # Optional. Only one of passphrase or KeyId should␣
↪be set, adds a layer of security by encrypting supported data stored at rest.

#       passphrase: mypassphrase

#       keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4

witnessGroups:
  - groupName: Witness Group  # required
    deploymentLocation: managed-devspatcher # required
    imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set

#     image: # only one of `imageId` or `image` should be set

#       url: docker.example.com/platform/example/postgres:latest

#       digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

    instanceSize:
      cpuCores: 2 # CPU must be between 0.1 and 15 Cores
      memoryGi: 2 # Memory must be between 0.1 and 10 Gi. For PGD clusters, the␣
↪recommended memory is 2 Gi or more.
```

この例は、サンプル YAML 構成ファイルを使用して高度な高可用性 AHA クラスターテンプレートを作成する方法を示しています。

AHA クラスターのサンプル YAML 構成ファイル

```
clusterType: aha # required, type of the cluster
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code)
  - color: blue
    tagName: tag-1
clusterName: create-aha-name # required
password: Password@1234 # Password must be 12 characters or more
dataGroup:
```

(次のページに続く)

(前のページからの続き)

```
  deploymentLocation: managed-devspatcher # required
  groupName: Data Group A # required
  networking: private # Network access type for the cluster. Valid values: private,␣
↪public

#   allowedIpRanges: # Optional, must contain at least one IP range with subnet mask␣
↪when specified

#     - cidrBlock: 9.9.9.9/28

#       description: Allow traffic from App A

#     - cidrBlock: 10.10.10.10/27

#       description: Allow traffic from App B

  backupRetentionPeriod: 30d # Required. Retention period must be between 1-180 days,␣
↪1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
  nodes: 3 # Required. Only can be set to 1, 2 or 3
  imageId: e791cfd57486281653d3d6135872125c # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set

#   image: # only one of `imageId` or `image` should be set

#     url: docker.example.com/platform/example/postgres:latest

#     digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

  postgresqlConfiguration:
    max_connections: 100
    max_locks_per_transaction: 64
  backupSchedule: 27 26 0  # Required. Scheduled Backup Cron
  instanceSize: # Required
      cpuCores: 2 # CPU must be between 0.1 and 47.51 Cores
      memoryGi: 2 # Memory must be between 0.1 and 178.51 Gi. For Aha clusters, the␣
↪recommended memory is 2 Gi or more.
  storage:
    databaseStorage: # Required
      sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). For Aha␣
↪clusters, the recommended size is 10 Gi or more.
      storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
    walStorage:
      sizeGi: 20 # Size must be between 0 and Unlimited (0-Unlimited Gi)
```

(次のページに続く)

(前のページからの続き)

```
    storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field

#   transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.

#     passphrase: mypassphrase

#     keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4
```

構成ファイルの構造

構成ファイルは、複数のクラスタータイプをサポートしています。以下は、各タイプのコアパラメーターです。

クラスタータイプとアーキテクチャ

- **Single** シングルノードデータベースクラスター。
- **HA** スタンバイレプリカを備えた高可用性クラスター。
- **PGD** : EDB Postgres マルチマスターまたはマルチリージョンセットアップ用の分散クラスター。
- **Analytics** 分析ワークロード用に最適化されたクラスター。

主要な構成パラメーター

- `clusterType` クラスターのタイプを指定します - `Single` 、`HA` 、`PGD` 、または `Analytics` 必須。
- `instanceSize` CPU コア 0.1-15 と Gi 単位のメモリ 0.1-10 必須。
- `storage` データベースストレージサイズと `storageClass` `ssd-csi` のような `storageClass` 必須。
- `backupSchedule` 自動バックアップの cron 式必須。
- `imageId` データベースクラスターのイメージ UUID。
- `postgresqlConfiguration` `postgresql.conf` のキーと値のペア。

edbctl cluster-template list

`edbctl cluster-template list` コマンドは、保存したデータベースクラスターブループリントを表示および検査する方法を提供します。このコマンドは、使用可能なテンプレートの高レベルの検出、テンプレートの仕様の詳細な検査、またはテンプレートの構成を YAML 形式にエクスポートして戻すために使用できます。

```
edbctl cluster-template list [--id [--config-yaml]] [options]
```

`options` は次のとおりです。

```
- i, --id string              The unique ID of the cluster template to inspect.
- n, --name string           Filter the list by a specific template name.
- y, --config-yaml           Prints the template' s configuration in a raw YAML format␣
↪(used with `--id`).
- o, --output string        Format options: table (default), json, yaml, xml, json-raw.
- P, --project  -P string     The project context. Defaults to `context_project`.
- -credential   -c  string  Authentication credential name. Defaults to `context_
↪credential`.
- -help -h                   Displays help for the `list` command.
```

これらの例は、現在のプロジェクトコンテキストにあるすべてのクラスターテンプレートをリストする方法を示しています。

```
#  List all templates:

edbctl cluster-template list

#  List templates filtered by name:

edbctl cluster-template list --name "prod-standard"

#  List templates using ID:

edbctl cluster-template list --id "p-c5fh47nfjcg4ci71uv32"

#  Export the list of templates in YAML format:

edbctl cluster-template list --id "p-c5fh47nfjcg4ci71uv32" --config-yaml
```

edbctl cluster

データベースクラスターリソースを管理します。このコマンドを使用して、すべての種類のクラスターを作成、編集、表示、削除、および復元します。

コマンド

`edbctl cluster` コマンドには次のオプションがあります。

- *edbctl cluster list* — アクティブまたは削除されたクラスターを表示します。
- *edbctl cluster list-connection-info* — 指定されたクラスターの接続文字列をリストします。
- *edbctl cluster create* — 新しいクラスターを作成します。
- *edbctl cluster edit* — 既存のクラスターを編集します。
- *edbctl cluster delete* — 指定されたクラスターを削除します。
- *edbctl cluster resume* — 指定されたクラスターを復元します

- *edbctl cluster switchover-primary* —指定されたレプリカをプライマリにスイッチオーバーします。

edbctl cluster list

アクティブまたは削除されたクラスターをすべてリストします。[--deleted] フラグを使用して、削除されたクラスターを表示します。クラスター ID またはアーキテクチャで結果をフィルタリングすることもできます。[--id] フラグを指定すると、インスタンスサイズ、ストレージ、ノード数を含むクラスターに関する詳細情報が表示されます。

```
edbctl cluster list [--id] [--deleted] [options]
```

options は次のとおりです。

```
- i, --id string                    Cluster ID. Filter the list by a specific␣
↪cluster ID.
- n, --name string                  Cluster name. Filter the list by a specific␣
↪cluster name.
- d, --deleted                      The cluster is a deleted cluster.
- A, --cluster-architecture string  Cluster architecture ID. Filter the list by␣
↪architecture (for example, single/ha).
- o, --output string                table json yaml xml json-raw (default "table").
- c, --credential string            The credential that you created via `credential␣
↪create` command. The default is fetched from `context_credential`
- P, --project string               The project that groups your clusters and other␣
↪resources. The default is taken from `context_project`.
- h, --help                         Help for list.
```

例

```
edbctl cluster list
edbctl cluster list --id p-c5fh47nfjcg4ci71uv32
edbctl cluster list --deleted
```

edbctl cluster list-connection-info

クラスター ID を使用して、指定された Postgres クラスターの接続文字列を取得および表示します。出力には、読み取り/書き込み URI、データベース名、ホスト、ポート、ユーザーが含まれます。

```
edbctl cluster list-connection-info {--id} [options]
```

options は次のとおりです。

```
-i, --id string           Cluster ID (required). The unique identifier for the␣
↪cluster to list connection string.
-o, --output string       [table json yaml xml json-raw] (default "table")
-c, --credential string   The credential that you created via `credential create`␣
↪command. The default is fetched from `context_credential`.
```

(次のページに続く)

(前のページからの続き)

```
-P, --project string      The project that groups your clusters and other resources.␣
↪the default is taken from `context_project`.
-h, --help                Help for list-connection-info.
```

これらの例は、特定のクラスターの接続文字列を取得および表示する方法を示しています。

```
edbctl cluster list-connection-info --id <pg_id>
edbctl cluster list-connection-info -i <pg_id>
```

edbctl cluster create

指定された構成ファイルを使用してクラスターを作成します。

このコマンドは、YAML 構成ファイルで定義された設定に基づいて、新しいクラスターをプロビジョニングします。ファイルは、次のような重要な詳細を指定します。

- **クラスタータイプ** single、ha、pgd、aha、または analytics
- **PostgreSQL 構成** pgConfig
- **リソース割り当て** ストレージ、メモリ
- **バックアップ設定**
- **画像**

```
edbctl cluster create [options]
```

options は次のとおりです。

```
- F, --config-file string   Path to the cluster config YAML file.
- c, --credential string    The credential that you created via the `credential␣
↪create` command. The default is fetched from `context_credential`.
- h, --help                 Help for create.
- P, --project string       The project that groups your clusters and other resources.
 The default is taken from `context_project`.
- y, --yes                  Auto-confirm all confirmations.
```

この例は、サンプル YAML ファイルを使用して PSR - シングルノードクラスターを作成する方法を示しています。

```
edbctl cluster create -F <Sample YAML Configuration file for single node>
```

単一ノードのサンプル YAML 構成ファイル

```
#  cluster details

clusterType: single # Required. Type of the cluster
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
```

(次のページに続く)

(前のページからの続き)

```
↪code)
  - color: blue
    tagName: tag-1
clusterName: create-single-name # Required.
password: password@1234 # Required. Password must be 12 characters or more

#  primary details

primaryPgName: Primary-Replica
imageId: 3e2af4f1dba3b9f8fb2bb6ae4834a5bd # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`
deploymentLocation: managed-devspatcher # Required.
networking: private # Required. Network access type for the cluster. Valid values:␣
↪private, public
# allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#   - cidrBlock: 127.0.0.1/24

#     description: test

backupRetentionPeriod: 30d # Required. Retention period must be between 1-180 days, 1-
↪25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
backupSchedule: 39 13 23  # Required. Scheduled Backup Cron
# image:  # only one of `imageId` or `image` should be set

#   url: docker.example.com/platform/example/postgres:latest

#   digest: sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

postgresqlConfiguration: # Optional
  max_connections: 100
instanceSize: # Required.
  cpuCores: 1 # CPU must be between 0.1 and 15 Cores
  memoryGi: 1 # Memory must be between 0.1 and 10 Gi
storage:
  databaseStorage: # Required.
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
    storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field
  walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs (Recommended for high write workloads)
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
    storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
```

(次のページに続く)

(前のページからの続き)

```
# transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.

#   passphrase: mypassphrase

#   keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4

#  replica details

replicaClusters:
  - replicaPgName: replica-1
    imageId: 3e2af4f1dba3b9f8fb2bb6ae4834a5bd # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`
    deploymentLocation: managed-devspatcher # Required.
    networking: private # Required. Network access type for the cluster. Valid␣
↪values: private, public
    #    allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified
    #      - cidrBlock: 127.0.0.1/24
    #        description: test
    backupRetentionPeriod: 30d # Required. Retention period must be between 1-180␣
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    backupSchedule: 39 13 23  # Required. Scheduled Backup Cron
    #    image: # only one of `imageId` or `image` should be set
    #      url: docker.example.com/platform/example/postgres:latest
    #      digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
    postgresqlConfiguration: # Optional
      max_connections: 100
    instanceSize: # Required.
      cpuCores: 1 # CPU must be between 0.1 and 15 Cores
      memoryGi: 1 # Memory must be between 0.1 and 10 Gi
    storage:
      databaseStorage: # Required.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field
      walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs (Recommended for high write workloads)
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
    #transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.
    #  passphrase: mypassphrase
```

(次のページに続く)

(前のページからの続き)

```
    #  keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4
  - replicaPgName: replica-2 # Required.
    imageId: 3e2af4f1dba3b9f8fb2bb6ae4834a5bd # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`
    deploymentLocation: managed-devspatcher # Required.
    networking: private # Required. Network access type for the cluster. Valid␣
↪values: private, public
    #    allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified
    #      - cidrBlock: 127.0.0.1/24
    #        description: test
    backupRetentionPeriod: 30d # Required. Retention period must be between 1-180␣
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    backupSchedule: 39 13 23  # Required. Scheduled Backup Cron
    #    image: # only one of `imageId` or `image` should be set
    #     url: docker.example.com/platform/example/postgres:latest
    #     digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
    postgresqlConfiguration: # Optional
      max_connections: 100
    instanceSize: # Required.
      cpuCores: 1 # CPU must be between 0.1 and 15 Cores
      memoryGi: 1 # Memory must be between 0.1 and 10 Gi
    storage:
      databaseStorage: # Required.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field
      walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs (Recommended for high write workloads)
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
    #transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.
    #  passphrase: mypassphrase
    #  keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4
```

この例は、 YAML 構成ファイルを使用して PSR - 高可用性クラスターを作成する方法を示しています。

```
edbctl cluster create -F <Sample YAML configuration file for High Availability>
```

高可用性のためのサンプル YAML 構成ファイル

```
#  cluster details
```

(次のページに続く)

(前のページからの続き)

```
clusterType: ha # Required. Type of the cluster
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code)
  - color: blue
    tagName: tag-1
clusterName: create-ha-name # Required.
password: password@1234 # Required. Password must be 12 characters or more

#  primary details

primaryPgName: Primary-Replica
imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
standbyReplicas: 2 # Required for HA cluster
deploymentLocation: managed-devspatcher # Required.
networking: private # Required. Network access type for the cluster. Valid values:␣
↪private, public
# allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#   - cidrBlock: 127.0.0.1/24

#     description: test

backupRetentionPeriod: 30d # Required. Retention period must be between 1-180 days, 1-
↪25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
backupSchedule: 39 13 23  # Required. Scheduled Backup Cron
# image: # only one of `imageId` or `image` should be set

#   url: docker.example.com/platform/example/postgres:latest

#   digest: sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

postgresqlConfiguration: # Optional
  max_connections: 100
instanceSize: # Required.
  cpuCores: 1 # CPU must be between 0.1 and 15 Cores
  memoryGi: 1 # Memory must be between 0.1 and 10 Gi
storage:
  databaseStorage: # Required.
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
    storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field
  walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
```

(次のページに続く)

(前のページからの続き)

```
↪Write-Ahead Logs (Recommended for high write workloads)
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
    storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field
# transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.

#   passphrase: mypassphrase

#   keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4

#  replica details

replicaClusters:
  - replicaPgName: replica-1
    standbyReplicas: 2 # Required for HA cluster
    imageId: 3e2af4f1dba3b9f8fb2bb6ae4834a5bd # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`
    deploymentLocation: managed-devspatcher # Required.
    networking: private # Required. Network access type for the cluster. Valid␣
↪values: private, public
    #    allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified
    #      - cidrBlock: 127.0.0.1/24
    #        description: test
    backupRetentionPeriod: 30d # Required. Retention period must be between 1-180␣
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    backupSchedule: 39 13 23  # Required. Scheduled Backup Cron
    #    image: # only one of `imageId` or `image` should be set
    #      url: docker.example.com/platform/example/postgres:latest
    #      digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
    postgresqlConfiguration: # Optional
      max_connections: 100
    instanceSize: # Required.
      cpuCores: 1 # CPU must be between 0.1 and 15 Cores
      memoryGi: 1 # Memory must be between 0.1 and 10 Gi
    storage:
      databaseStorage: # Required.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field
      walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs (Recommended for high write workloads)
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
```

(次のページに続く)

(前のページからの続き)

```
      storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
   #transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.
   #  passphrase: mypassphrase
   #  keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4
 - replicaPgName: replica-2 # Required.
   standbyReplicas: 2 # Required for HA cluster
   imageId: 3e2af4f1dba3b9f8fb2bb6ae4834a5bd # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`
   deploymentLocation: managed-devspatcher # Required.
   networking: private # Required. Network access type for the cluster. Valid␣
↪values: private, public
   #    allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified
   #      - cidrBlock: 127.0.0.1/24
   #        description: test
   backupRetentionPeriod: 30d # Required. Retention period must be between 1-180␣
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
   backupSchedule: 39 13 23  # Required. Scheduled Backup Cron
   #    image: # only one of `imageId` or `image` should be set
   #     url: docker.example.com/platform/example/postgres:latest
   #     digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
   postgresqlConfiguration: # Optional
     max_connections: 100
   instanceSize: # Required.
     cpuCores: 1 # CPU must be between 0.1 and 15 Cores
     memoryGi: 1 # Memory must be between 0.1 and 10 Gi
   storage:
     databaseStorage: # Required.
       sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
       storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field
     walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs (Recommended for high write workloads)
       sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
       storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
   #transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.
   #  passphrase: mypassphrase
   #  keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4
```

この例は、サンプル YAML 構成ファイルを使用して分析クラスターを作成する方法を示しています。

```
edbctl cluster create -F <Sample YAML configuration file for Analytics>
```

アナリティクスのサンプル YAML 構成ファイル

```
clusterType: analytics # required, type of the cluster
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code)
  - color: blue
    tagName: tag-1
imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
clusterName: create-analytics-name # required
deploymentLocation: managed-devspatcher # required
password: password@1234 #Password must be 12 characters or more
networking: private # Network access type for the cluster. Valid values: private,␣
↪public
# allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#   - cidrBlock: 127.0.0.1/24

#     description: test

backupRetentionPeriod: 30d # Required. Retention period must be between 1-180 days, 1-
↪25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
backupSchedule: 39 13 23  # Required. Scheduled Backup Cron
# image: # only one of `imageId` or `image` should be set

#   url: docker.example.com/platform/example/postgres:latest

#   digest: sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

postgresqlConfiguration: # Optional
  max_connections: 100
instanceSize: # Required
    cpuCores: 1 # CPU must be between 0.1 and 15 Cores
    memoryGi: 1 # Memory must be between 0.1 and 10 Gi
```

この例は、サンプル YAML 構成ファイルを使用して PGD クラスターを作成する方法を示しています。

```
edbctl cluster create -F <Sample YAML configuration file for PGD>
```

PGD のサンプル YAML 構成ファイル

```
clusterType: pgd # Type of the cluster
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
```

(次のページに続く)

(前のページからの続き)

```
↪code)
  - color: blue
    tagName: tag-1
clusterName: create-pgd-name # required
password: Password@1234 #Password must be 12 characters or more
dataGroups:
  - groupName: Data Group A # required
    deploymentLocation: managed-devspatcher # required
    imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
    networking: private # Network access type for the cluster. Valid values: private,␣
↪public

#     allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#       - cidrBlock: 9.9.9.9/28

#         description: Allow traffic from App A

#       - cidrBlock: 10.10.10.10/27

#         description: Allow traffic from App B

    backupRetentionPeriod: 30d # Required. Retention period must be between 1-180␣
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    nodes: 3 # Required. Only can be set to 2 or 3

#     image: # only one of `imageId` or `image` should be set

#       url: docker.example.com/platform/example/postgres:latest

#       digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

    postgresqlConfiguration:
      max_connections: 100
      max_locks_per_transaction: 64
    backupSchedule: 27 26 0  # Required. Scheduled Backup Cron
    instanceSize: # Required
        cpuCores: 2 # CPU must be between 0.1 and 15 Cores
        memoryGi: 2 # Memory must be between 0.1 and 10 Gi. For PGD clusters, the␣
↪recommended memory is 2 Gi or more.
    storage:
      databaseStorage: # Required
```

(次のページに続く)

(前のページからの続き)

```
       sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). For PGD␣
↪clusters, the recommended size is 10 Gi or more.
       storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
     walStorage:   # Optional, WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs.
       sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
       storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field

#     transparentDataEncryption: # Optional. Only one of passphrase or KeyId should␣
↪be set, adds a layer of security by encrypting supported data stored at rest.

#       passphrase: mypassphrase

#       keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4

  - groupName: Data Group B
    deploymentLocation: managed-devspatcher # required
    imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
    networking: private # Network access type for the cluster. Valid values: private,␣
↪public

#     allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#       - cidrBlock: 9.9.9.9/28

#         description: Allow traffic from App A

#       - cidrBlock: 10.10.10.10/27

#         description: Allow traffic from App B

    backupRetentionPeriod: 30d  # Required. Retention period must be between 1-180␣
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    nodes: 3 # only can be set to 2 or 3

#     image: # only one of `imageId` or `image` should be set

#       url: docker.example.com/platform/example/postgres:latest

#       digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
```

(次のページに続く)

(前のページからの続き)

```
    postgresqlConfiguration:
      max_connections: 100
      max_locks_per_transaction: 64
    backupSchedule: 6 27 0  # Required. Scheduled Backup Cron
    instanceSize: # Required
        cpuCores: 2 # CPU must be between 0.1 and 15 Cores
        memoryGi: 2 # Memory must be between 0.1 and 10 Gi. For PGD clusters, the␣
↪recommended memory is 2 Gi or more.
    storage:
      databaseStorage: # Required
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). For PGD␣
↪clusters, the recommended size is 10 Gi or more.
        storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field
      walStorage: # Optional field, but default value is 1 Gi
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field

#     transparentDataEncryption: # Optional. Only one of passphrase or KeyId should␣
↪be set, adds a layer of security by encrypting supported data stored at rest.

#       passphrase: mypassphrase

#       keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4

witnessGroups:
  - groupName: Witness Group  # required
    deploymentLocation: managed-devspatcher # required
    imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set

#     image: # only one of `imageId` or `image` should be set

#       url: docker.example.com/platform/example/postgres:latest

#       digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

    instanceSize:
        cpuCores: 2 # CPU must be between 0.1 and 15 Cores
        memoryGi: 2 # Memory must be between 0.1 and 10 Gi. For PGD clusters, the␣
↪recommended memory is 2 Gi or more.
```

この例は、サンプル YAML 構成ファイルを使用して高度な高可用性 AHA クラスターを作成する方法を示しています。

```
edbctl cluster create -F <Sample YAML configuration file for AHA>
```

AHA クラスターのサンプル YAML 構成ファイル

```
clusterType: aha # required, type of the cluster
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code)
  - color: blue
    tagName: tag-1
clusterName: create-aha-name # required
password: Password@1234 # Password must be 12 characters or more
dataGroup:
  deploymentLocation: managed-devspatcher # required
  groupName: Data Group A # required
  networking: private # Network access type for the cluster. Valid values: private,␣
↪public

#   allowedIpRanges: # Optional, must contain at least one IP range with subnet mask␣
↪when specified

#     - cidrBlock: 9.9.9.9/28

#       description: Allow traffic from App A

#     - cidrBlock: 10.10.10.10/27

#       description: Allow traffic from App B

  backupRetentionPeriod: 30d # Required. Retention period must be between 1-180 days,␣
↪1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
  nodes: 3 # Required. Only can be set to 1, 2 or 3
  imageId: e791cfd57486281653d3d6135872125c # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set

#   image: # only one of `imageId` or `image` should be set

#     url: docker.example.com/platform/example/postgres:latest

#     digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

  postgresqlConfiguration:
    max_connections: 100
```

(次のページに続く)

(前のページからの続き)

```
    max_locks_per_transaction: 64
  backupSchedule: 27 26 0  # Required. Scheduled Backup Cron
  instanceSize: # Required
      cpuCores: 2 # CPU must be between 0.1 and 47.51 Cores
      memoryGi: 2 # Memory must be between 0.1 and 178.51 Gi. For Aha clusters, the␣
↪recommended memory is 2 Gi or more.
  storage:
    databaseStorage: # Required
      sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). For Aha␣
↪clusters, the recommended size is 10 Gi or more.
      storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
    walStorage:
      sizeGi: 20 # Size must be between 0 and Unlimited (0-Unlimited Gi)
      storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field

#   transparentDataEncryption: # Optional. Only one of passphrase or KeyId should be␣
↪set, adds a layer of security by encrypting supported data stored at rest.

#     passphrase: mypassphrase

#     keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4
```

edbctl cluster edit

YAML 構成ファイルを使用して、既存のクラスターを編集します。このコマンドを使用すると、クラスター設定、データベース構成、バックアップポリシー、拡張機能、およびリソース割り当てを変更できます。

```
edbctl cluster edit [options]
```

options は次のとおりです。

```
- F, --config-file string   Path to the cluster config YAML file.
- c, --credential string    The credential that you created via the `credential␣
↪create` command. The default is fetched from `context_credential`.
- h, --help                 Help for edit.
- P, --project string       The project that groups your clusters and other resources.
 The default is taken from `context_project`.
- y, --yes                  Auto-confirm all confirmations.
```

この例は、YAML 構成ファイルを使用してクラスターを編集する方法を示しています。

!!!note 次のフィールドは必須です。 - clusterID - clusterType 他のフィールドはすべてオプショナル。指定しない場合、値は変更されません。

```
edbctl cluster edit -F <Example YAML configuration file>
```

YAML 構成ファイルの例

```
clusterType: single # checks that cluster type is a single cluster, this can only be␣
↪changed to ha
clusterId: p-abcdefg # cluster id of the cluster to be edited
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code). Remove tags to preserve existing tags. tags values will overwrite existing␣
↪tags. set tags: [] to remove all tags.
  - color: blue
    tagName: production
  - color: green
    tagName: staging
  - color: red
    tagName: development
name: edit-single-name
primaryPgName: edited-primary-replica-name
imageId: 6535c8bc9e358f0afaf5a706ea1bc722 # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
password: password@1234 #Password must be 12 characters or more
networking: private # Network access type for the cluster. Valid values: private,␣
↪public
# allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#   - cidrBlock: "192.168.1.0/24"

#     description: "Office network"

#   - cidrBlock: "10.0.0.0/16"

#     description: "VPN network"

backupRetentionPeriod: 40d # Retention period must be between 1-180 days, 1-25 weeks,␣
↪or 1-6 months. Valid units: days(d),weeks(w),months(m)
# image: # only one of `imageId` or `image` should be set

#   url: docker.example.com/platform/example/postgres:latest

#   digest: sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

postgresqlConfiguration: # Optional
  max_connections: 500
  shared_buffers: 256
  application_name: beaconator
```

(次のページに続く)

(前のページからの続き)

```
backupSchedule: 0 6    # Scheduled Backup Time in Cron format (UTC)
instanceSize:
  cpuCores: 4 # CPU must be between 0.1 and 15 Cores
  memoryGi: 16 # Memory must be between 0.1 and 10 Gi
storage:
  databaseStorage:  # Primary Storage size in Gi
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size needs to␣
↪be the same or greater than the existing databaseStorage size.
    storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
  walStorage:   # Optional field. WAL Storage size in Gi, a separate storage volume␣
↪for Write-Ahead Logs (Recommended for high write workloads) (Optional)
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size needs to␣
↪be the same or greater than the existing walStorage size.
    storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field

#  to edit: if you provide a replica with a replicaPgId then that existing replica␣
↪will be preserved/edited

#  to add a replica: if you do provide a replica without a replicaPgId then a new␣
↪replica will be created

#  to remove a replica: if you do not provide a replica with a replicaPgId that␣
↪exists then that existing replica will be removed

replicaClusters:
  # this is editing a replica cluster because it provides a replicaPgId
  - replicaPgId: p-abcdefg-b
    replicaPgName: test replica d
    imageId: 6535c8bc9e358f0afaf5a706ea1bc722 # Image ID can be fetched with command␣
↪`edbctl image list-image-tags --location-id <location-id>`, only one of `imageId` or␣
↪`image` should be set
    networking: private # Network access type for the cluster. Valid values: private,␣
↪public
    #allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified
    #  - cidrBlock: "192.168.1.0/24"
    #    description: "Office network"
    #  - cidrBlock: "10.0.0.0/16"
    #    description: "VPN network"
    backupRetentionPeriod: 40d # Retention period must be between 1-180 days, 1-25␣
↪weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    #image: # only one of `imageId` or `image` should be set
    #  url: docker.example.com/platform/example/postgres:latest
```

(次のページに続く)

(前のページからの続き)

```
    #  digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
    postgresqlConfiguration: # Optional
      max_connections: 500
      shared_buffers: 256
      application_name: beaconator
    backupSchedule: 0 6    # Scheduled Backup Time in Cron format (UTC)
    instanceSize:
      cpuCores: 4 # CPU must be between 0.1 and 15 Cores
      memoryGi: 16 # Memory must be between 0.1 and 10 Gi
    storage:
      databaseStorage:  # Primary Storage size in Gi
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size␣
↪needs to be the same or greater than the existing databaseStorage size.
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
      walStorage:   # Optional field. WAL Storage size in Gi, a separate storage␣
↪volume for Write-Ahead Logs (Recommended for high write workloads) (Optional)
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size␣
↪needs to be the same or greater than the existing walStorage size.
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
  # this is adding a new replica because it does not provide a replicaPgId
  - replicaPgName: test replica e
    locationId: managed-devspatcher
    imageId: 6535c8bc9e358f0afaf5a706ea1bc722 # Image ID can be fetched with command␣
↪`edbctl image list-image-tags --location-id <location-id>`, only one of `imageId` or␣
↪`image` should be set
    networking: private # Network access type for the cluster. Valid values: private,␣
↪public
    #allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified
    #  - cidrBlock: "192.168.1.0/24"
    #    description: "Office network"
    #  - cidrBlock: "10.0.0.0/16"
    #    description: "VPN network"
    backupRetentionPeriod: 40d # Retention period must be between 1-180 days, 1-25␣
↪weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    #image: # only one of `imageId` or `image` should be set
    #  url: docker.example.com/platform/example/postgres:latest
    #  digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
    postgresqlConfiguration: # Optional
      max_connections: 500
      shared_buffers: 256
```

(次のページに続く)

(前のページからの続き)

```
    application_name: beaconator
  backupSchedule: 0 6    # Scheduled Backup Time in Cron format (UTC)
  instanceSize:
    cpuCores: 4 # CPU must be between 0.1 and 15 Cores
    memoryGi: 16 # Memory must be between 0.1 and 10 Gi
  storage:
    databaseStorage:  # Primary Storage size in Gi
      sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size␣
↪needs to be the same or greater than the existing databaseStorage size.
      storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
    walStorage:   # Optional field. WAL Storage size in Gi, a separate storage␣
↪volume for Write-Ahead Logs (Recommended for high write workloads) (Optional)
      sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size␣
↪needs to be the same or greater than the existing walStorage size.
      storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
  # any replicas that are not provided with a replicaPgId will be removed
```

この例は、サンプル YAML ファイルを使用して高可用性クラスターを編集する方法を示しています。

!!!note 次のフィールドは必須です。

- clusterID
- clusterType

他のフィールドはすべてオプショナル。指定しない場合、値は変更されません。

```
clusterType: ha # checks that cluster type is a ha cluster, this can only be changed␣
↪to single
clusterId: p-abcdefg # cluster id of the cluster to be edited
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code). Remove tags to preserve existing tags. tags values will overwrite existing␣
↪tags. set tags: [] to remove all tags.
  - color: blue
    tagName: production
  - color: green
    tagName: staging
  - color: red
    tagName: development
name: edit-ha-name
imageId: 6535c8bc9e358f0afaf5a706ea1bc722 # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
password: password@1234 #Password must be 12 characters or more
networking: private # Network access type for the cluster. Valid values: private,␣
↪public
```

(次のページに続く)

(前のページからの続き)

```
# allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#   - cidrBlock: "192.168.1.0/24"

#     description: "Office network"

#   - cidrBlock: "10.0.0.0/16"

#     description: "VPN network"

backupRetentionPeriod: 40d # Retention period must be between 1-180 days, 1-25 weeks,␣
↪or 1-6 months. Valid units: days(d),weeks(w),months(m)
# image: # only one of `imageId` or `image` should be set

#   url: docker.example.com/platform/example/postgres:latest

#   digest: sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

postgresqlConfiguration: # Optional
  max_connections: 500
  shared_buffers: 256
  application_name: beaconator
backupSchedule: 0 6    # Scheduled Backup Time in Cron format (UTC)
instanceSize:
  cpuCores: 4 # CPU must be between 0.1 and 15 Cores
  memoryGi: 16 # Memory must be between 0.1 and 10 Gi
storage:
  databaseStorage:  # Primary Storage size in Gi
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size needs to␣
↪be the same or greater than the existing databaseStorage size.
    storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
  walStorage:   # Optional field. WAL Storage size in Gi, a separate storage volume␣
↪for Write-Ahead Logs (Recommended for high write workloads) (Optional)
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size needs to␣
↪be the same or greater than the existing walStorage size.
    storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field

#  to edit: if you provide a replica with a replicaPgId then that existing replica␣
↪will be preserved/edited

#  to add a replica: if you do provide a replica without a replicaPgId then a new␣
↪replica will be created
```

(次のページに続く)

(前のページからの続き)

```
#  to remove a replica: if you do not provide a replica with a replicaPgId that␣
↪exists then that existing replica will be removed

standbyReplicas: 3          # for HA cluster
replicaClusters:
  # this is editing a replica cluster because it provides a replicaPgId
  - replicaPgId: p-abcdefg-b
    replicaPgName: test replica d
    imageId: 6535c8bc9e358f0afaf5a706ea1bc722 # Image ID can be fetched with command␣
↪`edbctl image list-image-tags --location-id <location-id>`, only one of `imageId` or␣
↪`image` should be set
    networking: private # Network access type for the cluster. Valid values: private,␣
↪public
    #allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified
    #  - cidrBlock: "192.168.1.0/24"
    #    description: "Office network"
    #  - cidrBlock: "10.0.0.0/16"
    #    description: "VPN network"
    backupRetentionPeriod: 40d # Retention period must be between 1-180 days, 1-25␣
↪weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    #image: # only one of `imageId` or `image` should be set
    #  url: docker.example.com/platform/example/postgres:latest
    #  digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
    postgresqlConfiguration: # Optional
      max_connections: 500
      shared_buffers: 256
      application_name: beaconator
    backupSchedule: 0 6    # Scheduled Backup Time in Cron format (UTC)
    instanceSize:
      cpuCores: 4 # CPU must be between 0.1 and 15 Cores
      memoryGi: 16 # Memory must be between 0.1 and 10 Gi
    storage:
      databaseStorage:  # Primary Storage size in Gi
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size␣
↪needs to be the same or greater than the existing databaseStorage size.
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
      walStorage:   # Optional field. WAL Storage size in Gi, a separate storage␣
↪volume for Write-Ahead Logs (Recommended for high write workloads) (Optional)
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size␣
↪needs to be the same or greater than the existing walStorage size.
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
```

(次のページに続く)

(前のページからの続き)

```
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
    standbyReplicas: 3          # for HA cluster
  # this is adding a new replica because it does not provide a replicaPgId
  - replicaPgName: test replica e
    locationId: managed-devspatcher
    imageId: 6535c8bc9e358f0afaf5a706ea1bc722 # Image ID can be fetched with command␣
↪`edbctl image list-image-tags --location-id <location-id>`, only one of `imageId` or␣
↪`image` should be set
    networking: private # Network access type for the cluster. Valid values: private,␣
↪public
    #allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified
    #  - cidrBlock: "192.168.1.0/24"
    #    description: "Office network"
    #  - cidrBlock: "10.0.0.0/16"
    #    description: "VPN network"
    backupRetentionPeriod: 40d # Retention period must be between 1-180 days, 1-25␣
↪weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    #image: # only one of `imageId` or `image` should be set
    #  url: docker.example.com/platform/example/postgres:latest
    #  digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
    postgresqlConfiguration: # Optional
      max_connections: 500
      shared_buffers: 256
      application_name: beaconator
    backupSchedule: 0 6    # Scheduled Backup Time in Cron format (UTC)
    instanceSize:
      cpuCores: 4 # CPU must be between 0.1 and 15 Cores
      memoryGi: 16 # Memory must be between 0.1 and 10 Gi
    storage:
      databaseStorage:  # Primary Storage size in Gi
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size␣
↪needs to be the same or greater than the existing databaseStorage size.
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
      walStorage:   # Optional field. WAL Storage size in Gi, a separate storage␣
↪volume for Write-Ahead Logs (Recommended for high write workloads) (Optional)
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size␣
↪needs to be the same or greater than the existing walStorage size.
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
    standbyReplicas: 3          # for HA cluster
  # any replicas that are not provided with a replicaPgId will be removed
```

この例は、サンプル YAML 構成ファイルを使用して分析クラスターを編集する方法を示しています。

!!!note 次のフィールドは必須です。

- clusterID
- clusterType

他のフィールドはすべてオプショナル。指定しない場合、値は変更されません。

```
clusterType: analytics # checks that cluster type is an analytics cluster, cannot␣
↪change this field
clusterId: p-pd6kf9unmi # required, cluster ID of the cluster to be edited
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code). Remove tags to preserve existing tags. tags values will overwrite existing␣
↪tags. set tags: [] to remove all tags.
  - color: blue
    tagName: production
  - color: green
    tagName: staging
  - color: red
    tagName: development
imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
name: edit-analytics-name
password: password@1234 #Password must be 12 characters or more
networking: private # Network access type for the cluster. Valid values: private,␣
↪public
# allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#   - cidrBlock: 192.168.1.0/24

#     description: Office network

#   - cidrBlock: 10.0.0.0/16

#     description: VPN network

backupRetentionPeriod: 30d # Retention period must be between 1-180 days, 1-25 weeks,␣
↪or 1-6 months. Valid units: days(d),weeks(w),months(m)
backupSchedule: 0 6    # Scheduled Backup Time in Cron format (UTC)
# image: # only one of `imageId` or `image` should be set

#   url: docker.example.com/platform/example/postgres:latest

#   digest: sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
```

(次のページに続く)

(前のページからの続き)

```
postgresqlConfiguration: # Optional
  max_connections: 100
  max_locks_per_transaction: 64
instanceSize:
    cpuCores: 4 # CPU must be between 0.1 and 15 Cores
    memoryGi: 16 # Memory must be between 0.1 and 10 Gi
```

この例は、サンプル YAML 構成ファイルを使用して PGD クラスターを編集する方法を示しています。

!!!note フィールド clusterType および clusterId は必須 YAML ファイルを形成するロジックは次のとおりです。- groupId のみをメンションした場合、グループは保持されます。 - いくつかのフィールドとともに groupId をメンションすると、グループが保持され、指定された値で更新されます。 - groupId を指定せず、groupName 、image などの詳細を追加すると、そのグループが追加されます。 - groupId について言及しない場合、グループは削除されます。

```
clusterType: pgd # checks that cluster type is a pgd cluster, cannot change this␣
↪field
clusterId: p-12345abcde
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code). Remove tags to preserve existing tags. tags values will overwrite existing␣
↪tags. set tags: [] to remove all tags.
  - color: blue
    tagName: tag-1
name: edit-pgd-name
password: Password@1234
dataGroups:
  # add new data group A
  - groupName: Data Group A # required
    imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
    networking: private # Network access type for the cluster. Valid values: private,␣
↪public

#     allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#       - cidrBlock: 9.9.9.9/28

#         description: Allow traffic from App A

#       - cidrBlock: 10.10.10.10/27

#         description: Allow traffic from App B

    backupRetentionPeriod: 30d # Retention period must be between 1-180 days, 1-25␣
```

(次のページに続く)

(前のページからの続き)

```
↪weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    nodes: 2 # only can be set to 2 or 3

#     image: # only one of `imageId` or `image` should be set

#       url: docker.example.com/platform/example/postgres:latest

#       digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

    postgresqlConfiguration: # Optional
      max_connections: 100
      max_locks_per_transaction: 64
    backupSchedule: 0 4 8  # Scheduled Backup Cron
    instanceSize:
        cpuCores: 2 # CPU must be between 0.1 and 15 Cores
        memoryGi: 2 # Memory must be between 0.1 and 10 Gi. For PGD clusters, the␣
↪recommended memory is 2 Gi or more.
    storage:
      databaseStorage: # Size must be between 0 and Unlimited (0-Unlimited Gi)
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). For PGD␣
↪clusters, the recommended size is 10 Gi or more. Size needs to be the same or␣
↪greater than the existing databaseStorage size.
      walStorage: # Optional, WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size␣
↪needs to be the same or greater than the existing walStorage size.
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
  - groupId: p-12345abcde-x
    networking: private # Network access type for the cluster. Valid values: private,␣
↪public
    groupName: Data Group B # required
    imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set

#     allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#       - cidrBlock: 9.9.9.9/28

#         description: Allow traffic from App A

#       - cidrBlock: 10.10.10.10/27
```

(次のページに続く)

(前のページからの続き)

```
#         description: Allow traffic from App B

    backupRetentionPeriod: 30d # Retention period must be between 1-180 days, 1-25␣
↪weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    nodes: 2 # only can be set to 2 or 3

#     image: # only one of `imageId` or `image` should be set

#       url: docker.example.com/platform/example/postgres:latest

#       digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

    postgresqlConfiguration: # Optional
      max_connections: 100
      max_locks_per_transaction: 64
    backupSchedule: 0 4 8  # Scheduled Backup Cron
    instanceSize:
        cpuCores: 2 # CPU must be between 0.1 and 15 Cores
        memoryGi: 2 # Memory must be between 0.1 and 10 Gi. For PGD clusters, the␣
↪recommended memory is 2 Gi or more.
    storage:
      databaseStorage:
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size␣
↪needs to be the same or greater than the existing databaseStorage size.
      walStorage: # Optional, WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). Size␣
↪needs to be the same or greater than the existing walStorage size.
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
witnessGroups:
  # add new witness group
  - groupName: Witness Group A # required when adding a new witness group
    imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set

#     image: # only one of `imageId` or `image` should be set

#       url: docker.example.com/platform/example/postgres:latest

#       digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

    instanceSize:
```

(次のページに続く)

(前のページからの続き)

```
        cpuCores: 2 # CPU must be between 0.1 and 15 Cores
        memoryGi: 2 # Memory must be between 0.1 and 10 Gi. For PGD clusters, the␣
↪recommended memory is 2 Gi or more.
```

この例は、サンプル YAML 構成ファイルを使用してグループを保持しながら PGD クラスターを編集する方法を示しています。

PGD の 2 番目のサンプル YAML 構成ファイル

```
clusterType: pgd # checks that cluster type is a pgd cluster, cannot change this␣
↪field
clusterId: p-12345abcde # this will ensure the cluster will be retained
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code). Remove tags to preserve existing tags. tags values will overwrite existing␣
↪tags. set tags: [] to remove all tags.
  - color: blue
    tagName: tag-1
name: edit-pgd-name
password: Password@1234 #Password must be 12 characters or more
dataGroups:
  - groupId: p-12345abcde-a # this will ensure the group will be retained
  - groupId: p-12345abcde-b
    imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
    backupRetentionPeriod: 30d # Retention period must be between 1-180 days, 1-25␣
↪weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
witnessGroups: # this will ensure the witness group will be retained
  - groupId: p-12345abcde-c
    imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
    groupName: Witness Group JFK

#     image: # only one of `imageId` or `image` should be set, should match the image␣
↪of the data group

#       url: docker.example.com/platform/example/postgres:latest

#       digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

    instanceSize:
        cpuCores: 2 # CPU must be between 0.1 and 15 Cores
        memoryGi: 2 # Memory must be between 0.1 and 10 Gi. For PGD clusters, the␣
↪recommended memory is 2 Gi or more.
```

edbctl cluster delete

クラスター ID を使用して、指定されたクラスターを削除します。このアクションは、プロジェクトからクラスターを完全に削除し、元に戻すことはできないため、続行する前に正しいクラスター ID があることを確認してください。

```
edbctl cluster delete [options]
```

options は次のとおりです。

```
- i, --id string           Cluster ID (required). The unique identifier for the␣
↪cluster to delete.
- c, --credential string   The credential that you created via the `credential␣
↪create` command. The default is fetched from `context_credential`.
- y, --yes                 Auto-confirm all confirmations.
- P, --project string      The project that groups your clusters and other resources,␣
↪the default is taken from `context_project`.
- h, --help                Help for delete
```

これらの例は、クラスターを削除する方法を示しています。

```
edbctl cluster delete
edbctl cluster delete --id p-8nllnskncq
edbctl cluster delete -i p-8nllnskncq
```

edbctl cluster resume

必要な仕様を含む YAML 構成ファイルを提供することにより、クラスターを復元できます。cluster restore コマンドを使用し、 --file または-f オプションを使用してファイルを渡します。復元中に、次のいずれかの復元モードを指定する必要があります。

- full
- pointInTime

一度に 1 つのモードのみのコメントを外してください。

!!!note restoreRequest および password フィールドは、PGD および AHA を除くすべてのクラスターで必須です。他のフィールドはすべてオプショナルであり、指定しない場合、ソースクラスターの値がデフォルトになります。

```
edbctl cluster restore [options]
```

options は次のとおりです。

```
- F, --config-file string   Path to the cluster config YAML file (required).
- c, --credential string    The credential that you created via the `credential␣
↪create` command. The default is fetched from `context_credential`.
- y, --yes                  Auto-confirm all confirmations.
```

(次のページに続く)

(前のページからの続き)

```
- P, --project string       The project that groups your clusters and other resources.
 The default is taken from `context_project`.
- h, --help                 Help for restore.
```

この例は、YAML 構成ファイルを使用して単一のクラスターを復元する方法を示しています。

!!!note 次のフィールドは必須です。

- clusterType
- restoreRequest
- password

他のすべてのプライマリクラスターフィールドはオプショナル。指定されない場合、ソース値を使用します。レプリカの詳細は必要に応じて追加できますが、必須ではありません。

restore-single.yaml

```
clusterType: single # checks that cluster type is a single cluster, this can only be␣
↪changed to ha
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code). Remove tags to use source cluster tags. tags values will use exactly these␣
↪tags only. set tags: [] to use no tags.
  - color: blue
    tagName: tag-1
clusterName: restore-single-name
restoreRequest:
  # only one of the fields full or pointInTime can be set
  full:
    postgresBackupId: 20241112T211900
    sourceClusterId: p-12345abcde
  #  pointInTime:
  #    pointInTime: 2025-01-01T13:40:04.000Z
  #    sourceClusterId: p-12345abcde

#  primary replica details

primaryPgName:  Primary Cluster # Required.
deploymentLocation: managed-devspatcher
password: password@1234 #Password must be 12 characters or more
imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
networking: private # Network access type for the cluster. Valid values: private,␣
↪public
# allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified
```

(次のページに続く)

(前のページからの続き)

```
#   - cidrBlock: 127.0.0.1/24

#     description: test

backupRetentionPeriod: 30d # Retention period must be between 1-180 days, 1-25 weeks,␣
↪or 1-6 months. Valid units: days(d),weeks(w),months(m)
backupSchedule: 39 13 23  # Scheduled Backup Cron
# image: # only one of `imageId` or `image` should be set

#   url: docker.example.com/platform/example/postgres:latest

#   digest: sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

postgresqlConfiguration: # Optional
  max_connections: 100
instanceSize:
    cpuCores: 1 # CPU must be between 0.1 and 15 Cores
    memoryGi: 1 # Memory must be between 0.1 and 10 Gi
storage:
  databaseStorage:
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
    storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field
  walStorage:   # Optional field. WAL Storage size in Gi, a separate storage volume␣
↪for Write-Ahead Logs (Recommended for high write workloads) (Optional)
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
    storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field
# transparentDataEncryption: # Optional. Only one of passphrase or keyId should be␣
↪set. If TDE is enabled on source cluster, configuration must match source or␣
↪provide passphrase.

#   passphrase: mypassphrase # Required if source cluster uses TDE with passphrase

#   #keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4 # Uses KMS key from source if source␣
↪cluster uses KMS

#  replica details

replicaClusters:
  - replicaPgName: replica-1
    imageId: 3e2af4f1dba3b9f8fb2bb6ae4834a5bd # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`
    deploymentLocation: managed-devspatcher # Required.
    networking: private # Required. Network access type for the cluster. Valid␣
```

(次のページに続く)

(前のページからの続き)

```
↪values: private, public
    #    allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified
    #      - cidrBlock: 127.0.0.1/24
    #        description: test
    backupRetentionPeriod: 30d # Required. Retention period must be between 1-180␣
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    backupSchedule: 39 13 23  # Required. Scheduled Backup Cron
    #    image: # only one of `imageId` or `image` should be set
    #      url: docker.example.com/platform/example/postgres:latest
    #      digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
    postgresqlConfiguration: # Optional
      max_connections: 100
    instanceSize: # Required.
      cpuCores: 1 # CPU must be between 0.1 and 15 Cores
      memoryGi: 1 # Memory must be between 0.1 and 10 Gi
    storage:
      databaseStorage: # Required.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field
      walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs (Recommended for high write workloads)
        sizeGi: 1 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
    transparentDataEncryption: # Optional. Only one of passphrase or keyId should be␣
↪set. If TDE is enabled on source cluster, configuration must match source or␣
↪provide passphrase.
      passphrase: mypassphrase # Required if source cluster uses TDE with passphrase
      #keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4 # Uses KMS key from source if␣
↪source cluster uses KMS
```

この例は、YAML 構成ファイルを使用して高可用性クラスターを復元する方法を示しています。

!!!note 次のフィールドは必須です。

- clusterType
- restoreRequest
- password

他のすべてのプライマリクラスターフィールドはオプショナル。指定されない場合、ソース値を使用します。レプリカの詳細は必要に応じて追加できますが、必須ではありません。

restore-ha.yaml

```
clusterType: ha # checks that cluster type is a ha cluster, this can only be changed␣
↪to single
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code). Remove tags to use source cluster tags. tags values will use exactly these␣
↪tags only. set tags: [] to use no tags.
  - color: blue
    tagName: tag-1
clusterName: restore-ha-name
restoreRequest:
  # only one of the fields full or pointInTime can be set
  full:
    postgresBackupId: 20241112T211900
    sourceClusterId: p-12345abcde
  #  pointInTime:
  #    pointInTime: 2025-01-01T13:40:04.000Z
  #    sourceClusterId: p-12345abcde

#  primary replica details

primaryPgName:  Primary Cluster # Required.
deploymentLocation: managed-devspatcher
password: password@1234 #Password must be 12 characters or more
imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
networking: private # Network access type for the cluster. Valid values: private,␣
↪public
# allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#   - cidrBlock: 127.0.0.1/24

#     description: test

backupRetentionPeriod: 30d # Retention period must be between 1-180 days, 1-25 weeks,␣
↪or 1-6 months. Valid units: days(d),weeks(w),months(m)
backupSchedule: 39 13 23  # Scheduled Backup Cron
# image: # only one of `imageId` or `image` should be set

#   url: docker.example.com/platform/example/postgres:latest

#   digest: sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

postgresqlConfiguration: # Optional
  max_connections: 100
instanceSize:
```

(次のページに続く)

(前のページからの続き)

```
    cpuCores: 1 # CPU must be between 0.1 and 15 Cores
    memoryGi: 1 # Memory must be between 0.1 and 10 Gi
storage:
  databaseStorage:
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
    storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field
  walStorage:   # Optional field. WAL Storage size in Gi, a separate storage volume␣
↪for Write-Ahead Logs (Recommended for high write workloads) (Optional)
    sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
    storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field
# transparentDataEncryption: # Optional. Only one of passphrase or keyId should be␣
↪set. If TDE is enabled on source cluster, configuration must match source or␣
↪provide passphrase.

#   passphrase: mypassphrase # Required if source cluster uses TDE with passphrase

#   #keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4 # Uses KMS key from source if source␣
↪cluster uses KMS

standbyReplicas: 2 # for HA cluster

#  replica details

replicaClusters:
  - replicaPgName: replica-1
    imageId: 3e2af4f1dba3b9f8fb2bb6ae4834a5bd # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`
    deploymentLocation: managed-devspatcher # Required.
    networking: private # Required. Network access type for the cluster. Valid␣
↪values: private, public
    #    allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified
    #      - cidrBlock: 127.0.0.1/24
    #        description: test
    backupRetentionPeriod: 30d # Required. Retention period must be between 1-180␣
↪days, 1-25 weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    backupSchedule: 39 13 23  # Required. Scheduled Backup Cron
    #    image: # only one of `imageId` or `image` should be set
    #      url: docker.example.com/platform/example/postgres:latest
    #      digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890
    postgresqlConfiguration: # Optional
      max_connections: 100
```

(次のページに続く)

(前のページからの続き)

```
    instanceSize: # Required.
      cpuCores: 1 # CPU must be between 0.1 and 15 Cores
      memoryGi: 1 # Memory must be between 0.1 and 10 Gi
    storage:
      databaseStorage: # Required.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field
      walStorage:   # Optional. WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs (Recommended for high write workloads)
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field

#     transparentDataEncryption: # Optional. Only one of passphrase or keyId should␣
↪be set. If TDE is enabled on source cluster, configuration must match source or␣
↪provide passphrase.

#       passphrase: mypassphrase # Required if source cluster uses TDE with passphrase

#       #keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4 # Uses KMS key from source if␣
↪source cluster uses KMS

    standbyReplicas: 2 # Required for HA cluster
```

この例は、YAML 構成ファイルを使用して分析クラスターを復元する方法を示しています。

!!!note 次のフィールドは必須です。

- clusterType
- restoreRequest
- password

他のすべてのクラスターフィールドはオプショナル。指定されない場合、ソース値を使用します。

restore-analytics.yaml

```
clusterType: analytics # checks that cluster type is a analytics cluster, cannot␣
↪change this field
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code). Remove tags to use source cluster tags. tags values will use exactly these␣
↪tags only. set tags: [] to use no tags.
  - color: blue
    tagName: tag-1
clusterName: restore-analytics-name
deploymentLocation: managed-devspatcher
```

(次のページに続く)

(前のページからの続き)

```
password: password@1234 #Password must be 12 characters or more
imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
restoreRequest:

#  only one of the fields full or pointInTime can be set

  full:
    postgresBackupId: 20241112T211900
    sourceClusterId: p-12345abcde
  #  pointInTime:
  #    pointInTime: 2025-01-01T13:40:04.000Z
  #    sourceClusterId: p-12345abcde
networking: private # Network access type for the cluster. Valid values: private,␣
↪public
# allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#   - cidrBlock: 127.0.0.1/24

#     description: test

backupRetentionPeriod: 30d # Retention period must be between 1-180 days, 1-25 weeks,␣
↪or 1-6 months. Valid units: days(d),weeks(w),months(m)
backupSchedule: 39 13 23  # Scheduled Backup Cron
# image: # only one of `imageId` or `image` should be set

#   url: docker.example.com/platform/example/postgres:latest

#   digest: sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

postgresqlConfiguration: # Optional
  max_connections: 100
instanceSize:
    cpuCores: 1 # CPU must be between 0.1 and 15 Cores
    memoryGi: 1 # Memory must be between 0.1 and 10 Gi
```

この例は、YAML 構成ファイルを使用してリストアオペレーションで PGD クラスターの構成を定義する方法を示しています。

!!!note 次のフィールドは必須です。

- sourceClusterID
- clusterType
- password

- dataGroups
- restorePgdGroupRequest

他のフィールドはすべてオプショナル。省略すると、ソースクラスターの値が使用されます。

restore-pgd.yaml

```
clusterType: pgd # checks that cluster type is a pgd cluster, cannot change this␣
↪field
sourceClusterId: p-12345abcde # required
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code). Remove tags to use source cluster tags. tags values will use exactly these␣
↪tags only. set tags: [] to use no tags.
  - color: blue
    tagName: tag-1
name: restore-pgd-name
password: Password@1234 #Password must be 12 characters or more
dataGroups: # Currently, only one data group can be restored at a time
  - groupName: Data Group A
    imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
    deploymentLocation: managed-devspatcher
    networking: private # Network access type for the cluster. Valid values: private,␣
↪public
    backupRetentionPeriod: 30d # Retention period must be between 1-180 days, 1-25␣
↪weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
    nodes: 2 # only can be set to 2 or 3

#     image: # only one of `imageId` or `image` should be set

#       url: docker.example.com/platform/example/postgres:latest

#       digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

    postgresqlConfiguration: # Optional
      max_connections: 100
      max_locks_per_transaction: 64
    backupSchedule: 0 4 8  # Scheduled Backup Cron
    restorePgdGroupRequest:
      full: # for pointInTime restore mode use the pointInTime configuration as below␣
↪and comment out the full configuration
        postgresBackupId: 20241228T055553
        sourceGroupId: p-12345abcde-b
      # pointInTime:
      #   pointInTime: 2025-01-01T13:40:04.000Z
      #   sourceGroupId: p-12345abcde-b
```

(次のページに続く)

(前のページからの続き)

```
    instanceSize:
        cpuCores: 1 # CPU must be between 0.1 and 15 Cores
        memoryGi: 2 # Memory must be between 0.1 and 10 Gi. For PGD clusters, the␣
↪recommended memory is 2 Gi or more.
    storage:
      databaseStorage:
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi). For PGD␣
↪clusters, the recommended size is 10 Gi or more.
        storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field
      walStorage:   # Optional, WAL Storage size in Gi, a separate storage volume for␣
↪Write-Ahead Logs.
        sizeGi: 10 # Size must be between 0 and Unlimited (0-Unlimited Gi)
        storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
    transparentDataEncryption: # Optional. Only one of passphrase or keyId should be␣
↪set. If TDE is enabled on source cluster, configuration must match source or␣
↪provide passphrase.
      passphrase: mypassphrase # Required if source cluster uses TDE with passphrase

#       keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4 # Uses KMS key from source if␣
↪source cluster uses KMS
```

この例は、YAML 構成ファイルを使用してリストアオペレーションで AHA クラスターの構成を定義する方法を示しています。

!!!note 次のフィールドは必須です。

- sourceClusterID
- clusterType
- password
- dataGroups
- restoreAhagroupRequest

他のフィールドはすべてオプショナル。省略すると、ソースクラスターの値が使用されます。

restore-aha.yaml

```
clusterType: aha # checks that cluster type is a aha cluster, cannot change this␣
↪field
sourceClusterId: p-12345abcde # required
tags: # Optional tags for the cluster with name and color(valid name or hex color␣
↪code). Remove tags to use source cluster tags. tags values will use exactly these␣
↪tags only. set tags: [] to use no tags.
  - color: blue
```

(次のページに続く)

(前のページからの続き)

```
    tagName: tag-1
name: restore-aha-name
password: Password@1234 #Password must be 12 characters or more
dataGroup:
  groupName: Data Group A
  imageId: 6138b8f8e2293a87ebf7ba8dfd1f74da # Image ID can be fetched with command␣
↪`edbctl image list-image-tags`, only one of `imageId` or `image` should be set
  deploymentLocation: managed-devspatcher
  networking: private # Network access type for the cluster. Valid values: private,␣
↪public

#   allowedIpRanges: # Must contain at least one IP range with subnet mask when␣
↪specified

#     - cidrBlock: 9.9.9.9/28

#       description: Allow traffic from App A

#     - cidrBlock: 10.10.10.10/27

#       description: Allow traffic from App B

  backupRetentionPeriod: 30d # Retention period must be between 1-180 days, 1-25␣
↪weeks, or 1-6 months. Valid units: days(d),weeks(w),months(m)
  nodes: 2 # only can be set to 2 or 3

#   image: # only one of `imageId` or `image` should be set

#     url: docker.example.com/platform/example/postgres:latest

#     digest:␣
↪sha256:dummyhash1234567890abcdef1234567890abcdef1234567890abcdef1234567890

  postgresqlConfiguration: # Optional
    max_connections: 100
    max_locks_per_transaction: 64
  backupSchedule: 0 4 8  # Scheduled Backup Cron
  restoreAhaGroupRequest:
    #full: # for pointInTime restore mode use the pointInTime configuration as below␣
↪and comment out the full configuration
    #  postgresBackupId: 20241228T055553
     pointInTime:
       pointInTime: 2025-01-01T13:40:04.000Z
  instanceSize:
      cpuCores: 2 # CPU must be between 0.1 and 47.51 Cores
```

(次のページに続く)

(前のページからの続き)

```
      memoryGi: 2 # Memory must be between 0.1 and 178.51 Gi. For Aha clusters, the␣
↪recommended memory is 2 Gi or more.
  storage:
    databaseStorage:
      sizeGi: 20 # Size must be between 0 and Unlimited (0-Unlimited Gi). For Aha␣
↪clusters, the recommended size is 10 Gi or more.
      storageClass: ssd-csi # Supported storage classes can be fetched with command␣
↪`edbctl image list-image-tags` and checking `Supported Storage Classes` field
    walStorage:
      sizeGi: 20 # Size must be between 0 and Unlimited (0-Unlimited Gi)
      storageClass: standard-csi # Supported storage classes can be fetched with␣
↪command `edbctl image list-image-tags` and checking `Supported Storage Classes` field
  transparentDataEncryption: # Optional. Only one of passphrase or keyId should be␣
↪set. If TDE is enabled on source cluster, configuration must match source or␣
↪provide passphrase.
    passphrase: mypassphrase # Required if source cluster uses TDE with passphrase
    #keyId: 42c4f77e-3ebe-46ab-9749-4178887bded4 # Uses KMS key from source if source␣
↪cluster uses KMS
```

edbctl cluster switchover-primary

edbctl cluster switchover-primary コマンドは、データベースクラスター内で制御されたロールスワップを実行します。指定されたレプリカを新しいプライマリノードに昇格させますが、同時に現在のプライマリをレプリカロールに降格します。これは通常、計画的なメンテナンス、パフォーマンスの最適化、またはデータセンターの移行に使用されます。

```
edbctl cluster switchover-primary [options]
```

options は次のとおりです。

```
- i, --id string                The unique ID of the cluster to switchover␣
↪(required).
    --replica-id string        The unique ID of the replica node to promote to␣
↪Primary (required).
- P, --project string           The project context for the operation. Defaults to␣
↪`context_project`.
    --credential string        The credential to use for authentication. Defaults to␣
↪`context_credential`.
- y, --yes                      Auto-confirms all prompts and confirmations,␣
↪enabling non-interactive operation.
- h, --help                     Displays help information for the switchover-primary␣
↪command.
```

edbctl config

デフォルトのプロジェクト、資格情報、出力形式を含む edbctl 構成設定を表示または変更します。

コマンド

edbctl config コマンドには次のオプションがあります。

- *edbctl config get* — 指定された構成プロパティの値を印刷します。
- *edbctl config list* — すべての構成プロパティと値をリストします。
- *edbctl config set* — 指定されたプロパティの値で構成を更新します。

edbctl config get

指定された構成プロパティの値を表示します。

```
edbctl config get <key> [options]
```

options は次のとおりです。

```
-h, --help          Help for get.
-I, --interactive   Execute command interactively.
```

これらの例は、特定の構成プロパティの値を取得する方法を示しています。

```
$ edbctl config set output_mode json
$ edbctl config get output_mode
json
$ edbctl config get output_mode
table
$ edbctl config get confirm_enabled
true
```

edbctl config list

すべての構成プロパティとその値をリストします。

```
edbctl config list [options]
```

options は次のとおりです。

```
-h, --help            Help for list.
-o, --output string   table json yaml xml json-raw (default "table")
```

edbctl config set

指定された値で構成プロパティを更新します。

```
edbctl config set <key> <value> [options]
```

options は次のとおりです。

```
-h, --help          Help for set.
-I, --interactive   Execute command interactively.
```

これらの例は、特定の構成プロパティを設定する方法を示しています。

```
$ edbctl config set output_mode json
$ edbctl config set confirm_enabled true
$ edbctl config set prompt_enabled true
```

edbctl credential

このコマンドは、ユーザーがリソースへのアクセスを許可する認証資格情報を作成および管理します。ログイン資格情報として Access API キーをインポートしたり、環境変数 HCP_ACCESS_KEY を設定したりできます。

コマンド

edbctl credential コマンドには次のオプションがあります。

- *edbctl credential add-tde-key* - TDE キーを追加します。
- *edbctl* **資格情報リスト** - 資格情報を表示します。
- *edbctl credential list-tde-key* - プロジェクト内の TDE キーをリストします。
- *edbctl* **資格情報** *import-access-key* - 認証用に Access API キーをインポートします。
- *edbctl* **資格情報の削除** - 資格情報を削除します。

edbctl credential add-tde-key

edbctl credential add-tde-key コマンドは、透過的データ暗号化 TDE に使用されるマスターキーを外部キー管理サービス KMS から登録します。TDE は、保存時にデータベースファイルを暗号化することにより、追加のセキュリティレイヤーを提供し、EDB Postgres Advanced Server および EDB Postgres Extended Server バージョン 15 以降でサポートされています。

```
edbctl credential add-tde-key [options]
```

options は次のとおりです。

```
    --location-id string        Location ID (required). The location where the TDE␣
↪key will be stored.
    --kms-provider string       Key provider (required). The provider of the TDE key.␣
```

(次のページに続く)

(前のページからの続き)

```
↪Valid values: `aws`, `hashicorp-vault`.
    --hc-vault-address string   HashiCorp Vault address (required if key provider is␣
↪`hashicorp-vault`).
    --hc-vault-role string      HashiCorp Vault role (required if key provider is␣
↪`hashicorp-vault`).
    --hc-vault-key string       HashiCorp Vault key (required if key provider is␣
↪`hashicorp-vault`).
    --aws-kms-arn string        AWS KMS ARN (required if key provider is aws). The␣
↪Amazon Resource Name (ARN) of the AWS KMS key.
    --friendly-name string      Friendly name for the TDE key (optional). A human-
↪readable name for the TDE key.
- c, --credential string         The credential which you created via `credential␣
↪create` command, the default is fetched from `context_credential`.
- y, --yes                       Auto-confirm all confirmations.
- P, --project string            The project that groups your clusters and other␣
↪resources, the default is taken from `context_project`.
- h, --help                      help for `add-tde-key.`
```

この例は、AWS KMS から TDE キーを追加する方法を示しています。

```
edbctl credential add-tde-key \
  --location-id "loc_123" \
  --kms-provider aws \
  --aws-kms-arn "arn:aws:kms:us-east-1:123456789012:key/abcd1234-56ef-78gh-90ij" \
  --friendly-name "AWS-Prod-Key"
```

この例は、HashiCorp Vault から TDE キーを追加する方法を示しています。

```
edbctl credential add-tde-key \
  --location-id "loc_456" \
  --kms-provider hashicorp-vault \
  --hc-vault-address "https://vault.example.com" \
  --hc-vault-role "db-encryption-role" \
  --hc-vault-key "customer-data-key" \
  --friendly-name "Vault-Security-Key"
```

edbctl 資格情報リスト

このコマンドは、すべての資格情報を表示します。オプションで、 `--name` オプションを使用して、特定の資格情報をリストします。

```
edbctl credential list [options]
```

`options` は次のとおりです。

```
- h, --help            Help for list.
- n, --name string     Credential name (optional). Specify the name of the credential␣
↪to list.
- o, --output string   [table json yaml xml] \(default "table")
```

これらの例は、資格情報をリストする方法を示しています。

```
#  List all the credentials

edbctl credential list

#  List a specific credential

edbctl credential list -n <credential_name>
```

edbctl credential list-tde-key

edbctl credential list-tde-key コマンドを使用すると、プロジェクト内で登録されている透過的データ暗号化 TDE キーを表示および管理できます。キー登録を確認し、クラスター構成の特定のキー ID を取得し、特定の展開場所で使用可能なキーを監査する方法を提供します。

このコマンドは、特定の場所に関連付けられている TDE キーのリストを表示します。特定のキー ID を指定するか、プログラムで使用する出力形式を変更することにより、結果をさらに絞り込むことができます。

```
edbctl credential list-tde-key [options]
```

options は次のとおりです。

```
    --location-id string    Location ID (required). The location to filter TDE keys.
    --id string             Filter the list to display details for a specific TDE key.
- P, --project string        The project that groups your clusters and other␣
↪resources, the default is taken from `context_project`.
- o, --output string         Output format: `table` (default), `json`, `yaml`, `xml`,␣
↪`json-raw`.
- h, --help                  Displays help information for the list-tde-key command.
```

これらの例は、 list-tde-key コマンドの使用方法を示しています。

```
#  List all TDE keys for a specific location:

edbctl credential list-tde-key --location-id "loc_123"

#  List a specific TDE key by ID:

edbctl credential list-tde-key --location-id "loc_123" --id "key_456"

#  Export list to JSON:
```

(次のページに続く)

(前のページからの続き)

```
edbctl credential list-tde-key --location-id "loc_123" --output json
```

edbctl 資格情報 import-access-key

このコマンドは、Access API キーをインポートするために使用されます。サーバーポータルから Access API キーを取得して、認証できます。この資格情報を使用して、さらなる操作を行うこともできます。

```
edbctl credential import-access-key [--name --access-key --address] [options]
```

options は次のとおりです。

```
- n, --name string         Credential name (required). The unique name for the␣
↪credential.
- k, --access-key string   Access key (required). The API key used for authentication.
- a, --address string      Authenticator host address (required).
- I, --interactive         Execute command interactively.
- h, --help                Help for import-access-key.
```

これらの例は、 import-access-key コマンドの使用方法を示しています。

```
#  Import the Access API key:

edbctl credential import-access-key

#  Import the specific Access API key using the name, access-key, and address:

edbctl credential import-access-key --name cred1 --access-key <your-key-here> --
↪address <authenticator-host-address-here format: [http|https]://host:port example:␣
↪http://localhost:8080 and https://secure.portal.net:443>
>

edbctl credential import-access-key -n cred1 -k <your-key-here> -a <authenticator-
↪host-address-here format: [http|https]://host:port example: http://localhost:8080␣
↪and https://secure.portal.net:443>
```

edbctl 資格情報の削除

このコマンドは、名前を持つ特定の資格情報を削除します。

```
edbctl credential delete <credential_name> [options]
```

options は次のとおりです。

```
- h, --help          Help for delete.
- I, --interactive   Execute command interactively.
```

(次のページに続く)

(前のページからの続き)

```
- y, --yes           Auto-confirm all confirmations
```

この例は、資格情報を削除する方法を示しています。

```
#  Delete the credential with <credential_name> name:

edbctl credential delete <credential_name>
```

edbctl image-pull-secret

このコマンドは、ハイブリッドマネージャー HM のブートストラッププロセスで使用されるプルシークレットの管理に役立ちます。

コマンド

edbctl image-pull-secret コマンドには、次の使用可能なオプションがあります。

- *edbctl image-pull-secret create* — Kubernetes 名前空間と AssetPullSecret を作成する
- *edbctl image-pull-secret list* — Kubernetes 名前空間と AssetPullSecret をリストします

edbctl image-pull-secret create

提供された資格情報を使用して必要な Kubernetes 名前空間と AssetPullSecret を設定し、HM ブートストラップの準備をします。

```
edbctl image-pull-secret create --username <username> --password <password> --
↪registry <registry> [options]
```

options は次のとおりです。

```
    --username string   Registry username (required).
    --password string   Registry password (required).
    --registry string   Container Registry (required). The URL of the container␣
↪registry.
- h, --help              Help for create.
```

この例は、Kubernetes 名前空間と AssetPullSecret を作成する方法を示しています。

```
edbctl image-pull-secret create --username <username> --password <password> --
↪registry <registry>
```

edbctl image-pull-secret list

HM ブートストラップで使用される Kubernetes 名前空間と AssetPullSecret を取得およびリストします。

```
edbctl image-pull-secret list [options]
```

options は次のとおりです。

```
- h, --help help for list command
```

edbctl image

このコマンドは、ハイブリッドマネージャー HM のイメージの管理に役立ちます。このコマンドを使用すると、EDB のリポジトリからローカルレジストリにコンテナイメージを同期し、インターネットに切断された OpenShift 環境のアーティファクトを管理できます。

コマンド

edbctl image コマンドには次のオプションがあります。

- *edbctl image create-image-set-config-for-oc-mirror* —インターネットに切断された OpenShift 環境の AssetSetConfiguration YAML ファイルを作成します。
- *edbctl image list-image-tags* —プロビジョニングに使用可能なすべてのイメージのリストを表示します。
- *edbctl image sync-to-local-registry* — EDB Repos 2.0 からローカルレジストリにコンテナイメージを同期します。

edbctl image create-image-set-config-for-oc-mirror

インターネットアクセスのない OpenShift 環境の AssetSetConfiguration YAML ファイルを生成します。このファイルにより、oc Mirror は、指定された Hybrid Manager リリースのすべてのコンテナイメージを含むローカル tar ボールを作成できます。

```
edbctl image create-image-set-config-for-oc-mirror [options]
```

options は次のとおりです。

```
    --source-registry-username string   Source registry username (required). The␣
↪username for authenticating with the source registry.
    --source-registry-password string   Source registry password (required). The␣
↪password for authenticating with the source registry.
    --version string                    Hybrid Manager release version (required).␣
↪Specify the version of the release to sync.
    --output-file-path string           Output YAML file path (optional). Defaults to␣
↪ImageSetConfiguration.yaml in the current directory.
- h, --help                              Help for create-image-set-config-for-oc-
↪mirror.
```

この例は、OpenShift 環境の AssetSetConfiguration YAML ファイルを作成する方法を示しています。

```
edbctl image create-image-set-config-for-oc-mirror --source-registry-username
↪<username> --source-registry-password <password> --version <version> [--output-file-
↪path <output-file-path>]
```

edbctl image list-image-tags

プロビジョニングに使用されるすべての使用可能なイメージタグのリストを取得して表示します。出力には、各イメージタグの詳細なメタデータが含まれます。

- 画像タグ ID
- 名前
- SHA
- バージョン
- 圧縮サイズ
- アーキテクチャ
- 作成時間
- 摂取時間
- ロケーション
- 関連タグ

```
edbctl image list-image-tags [options]
```

options は次のとおりです。

```
    --location-id string        Location ID (required). Specify the location ID to␣
↪filter images.
    --pg-type string            Filter images by Postgres type. Examples: epas,␣
↪postgres, pgextended.
    --major-version string      Filter images by major version. Examples: 17, 16.
    --pg-architectures string   Filter images by cluster architectures. Examples: pgd,
 aha, psr, analytics.
- c, --credential string         The credential that you created via the `credential␣
↪create` command. The default is fetched from `context_credential`.
- o, --output string             [table json yaml xml] (default "table")
- h, --help                      Help for list-image-tags.
```

この例は、イメージをリストする方法を示しています。

```
edbctl image list-image-tags --location-id "managed-devspatcher"
```

edbctl image sync-to-local-registry

ローカル HM インストールを管理するために、EDB Repos 2.0 からローカルレジストリにコンテナイメージを同期します。

```
edbctl image sync-to-local-registry [options]
```

options は次のとおりです。

```
    --source-registry-username string        Source registry username (required).
    --source-registry-password string        Source registry password (required).
    --destination-registry-uri string        Destination registry URI (required).
    --destination-registry-username string   Destination registry username (required).
    --destination-registry-password string   Destination registry password (required).
    --version string                         Hybrid Manager release version␣
↪(required).
    --sign-by-sigstore-private-key string    Path to the Sigstore private key.
    --sign-passphrase-file string            Path to the file containing the␣
↪passphrase for the Sigstore private key.
- y, --yes                                    Auto-confirm all confirmations.
- h, --help                                   Help for sync-to-local-registry.
```

この例は、EDB Repo 2.0 をローカルレジストリに同期する方法を示しています。

```
edbctl image sync-to-local-registry --source-registry-username <username> --source-
↪registry-password <password> --destination-registry-uri <registry> --destination-
↪registry-username <username> --destination-registry-password <password> --version
↪<version>
```

edbctl operand

特にインターネットに接続されていないものを含む OpenShift 環境のコンテキストで、EDB Postgres AI オペランドを管理します。 `edbctl operand` は、次のようなオペランド管理を処理するコマンドのエントリポイントです。

- オペランドイメージの同期
- インターネットに切断されたエアギャップの OpenShift 環境のアーティファクトの管理

コマンド

- *edbctl* **オペランド** *sync-to-local-registry* オペランドイメージを EDB リポジトリからローカルレジストリに同期します。
- *edbctl operand list-images* EDB によって管理されているすべてのオペランドイメージの完全なリストを表示します。
- *edbctl* **オペランド** *download-image-list* 必要なオペランドイメージのリストを含むファイルをダウンロードするために使用されます。

edbctl オペランド sync-to-local-registry

オペランドイメージを EDB リポジトリからローカルレジストリに同期して、HM のローカルインストールを管理します。

```
edbctl operand sync-to-local-registry [options]
```

`options` は次のとおりです。

```
    --source-registry-username string        Source registry username (required).
    --source-registry-password string        Source registry password (required).
    --destination-registry-uri string        Destination registry URI (required).
    --destination-registry-username string   Destination registry username (required).
    --destination-registry-password string   Destination registry password (required).
    --version string                         Hybrid Manager release version␣
↪(required).
    --sign-by-sigstore-private-key string    Path to the Sigstore private key.
    --sign-passphrase-file string            Path to the file containing the␣
↪passphrase for the Sigstore private key.
-w, --workers integer                        Number of parallel workers to use for␣
↪syncing images (default is 3).
-y, --yes                                    Auto-confirm all confirmations.
-h, --help                                   Help for sync-to-local-registry.
```

この例は、EDB リポジトリからローカルレジストリにオペランドイメージを同期する方法を示しています。

```
edbctl image sync-to-local-registry --source-registry-username <username> --source-
↪registry-password <password> --destination-registry-uri <registry> --destination-
↪registry-username <username> --destination-registry-password <password> --version
↪<version> --workers 3 --cpu 3
```

edbctl operand list-images

このコマンドは、ハイブリッドマネージャーの特定のバージョンに関連付けられているすべてのコンテナイメージの包括的なリストを取得および表示するために使用されます。コンポーネントごとに、画像タブを含む重要な情報を提供します。

```
edbctl operand list-images [options]
```

options は次のとおりです。

```
    --source-registry-username string          Source registry username (required).␣
↪The username for accessing the repository where the images are stored.
    --source-registry-password string          Source registry password (required).␣
↪The password for accessing the repository where the images are stored.
    --version string                           Hybrid Manager release version␣
↪(required). Specifies which version of the operand to list images for.
-y, --yes                                      Auto-confirm all confirmations.
-h, --help                                     Displays help information for the list-
↪images command.
```

edbctl オペランド download-image-list

このコマンドを使用すると、ハイブリッドマネージャーの指定したバージョンに必要なすべてのオペランドイメージのリストを含むファイルをダウンロードできます。これは、エアギャップまたはインターネットに切断された環境でのインストールの準備をする場合に特に役立ちます。

```
edbctl operand download-image-list [options]
```

options は次のとおりです。

```
    --source-registry-username string       Source registry username (required). The␣
↪username for accessing the repository where the images are stored.
    --source-registry-password string       Source registry password (required). The␣
↪password for accessing the repository where the images are stored.
    --version string                        Hybrid Manager release version (required).
↪ Specifies which version of the operand to download the image list for.
    --output-filename string                Output file path (required). The path␣
↪where the downloaded image list file will be saved.
-y, --yes                                   Auto-confirm all confirmations.
-h, --help                                  Displays help information for the␣
↪download-image-list command.
```

この例は、特定のハイブリッドマネージャーバージョンのオペランドイメージリストをダウンロードする方法を示しています。

```
edbctl operand download-image-list --version "v1.0.0" --source-registry-username
↪"myuser" --source-registry-password "mypassword" --output-filename "/tmp/edb-images.
↪txt"
```

edbctl operator

EDB Postgres AI Operator コンポーネントとアーティファクトを管理します。edbctl operator コマンドは、Postgres AI Operator を管理するためのツールを提供します。

- オペレーター画像の同期
- インターネットに切断されたエアギャップの OpenShift 環境のアーティファクトの管理

コマンド

- *edbctl Operator sync-to-local-registry* オペレーターイメージを EDB リポジトリからローカルレジストリに同期します。

edbctl Operator sync-to-local-registry

オペレーターイメージを EDB リポジトリからローカルレジストリに同期して、ハイブリッドマネージャーのローカルインストールを管理します。

```
edbctl operator sync-to-local-registry [options]
```

options は次のとおりです。

```
    --source-registry-username string        Source registry username (required).
    --source-registry-password string        Source registry password (required).
    --destination-registry-uri string        Destination registry URI (required).
    --destination-registry-username string   Destination registry username (required).
    --destination-registry-password string   Destination registry password (required).
    --version string                         Hybrid Manager release version␣
↪(required).
    --sign-by-sigstore-private-key string    Path to the Sigstore private key.
    --sign-passphrase-file string            Path to the file containing the␣
↪passphrase for the Sigstore private key.
-y, --yes                                    Auto-confirm all confirmations.
-h, --help                                   Help for sync-to-local-registry.
```

この例は、EDB リポジトリをローカルレジストリに同期する方法を示しています。

```
edbctl operator sync-to-local-registry --source-registry-username <username> --source-
↪registry-password <password> --destination-registry-uri <registry> --destination-
↪registry-username <username> --destination-registry-password <password> --version
↪<version>
```

edbctl project

クラスターとリソースを編成するためのプロジェクトを管理します。このコマンドを使用してプロジェクトを作成、編集、表示、および削除すると、データベースクラスターの効率的な管理が簡単になります。

コマンド

- *edbctl* **プロジェクトリスト** — 使用可能なすべてのプロジェクトを表示します。
- *edbctl project create* — 新しいプロジェクトを作成します。
- *edbctl* **プロジェクトの編集** — プロジェクトを編集します。
- *edbctl project delete* — 指定されたプロジェクトを削除します。

edbctl プロジェクトリスト

使用可能なすべてのプロジェクトをリストし、プロジェクト ID、名前、ユーザー数、クラスター数、タグを含む詳細情報を表示します。プロジェクト名または ID で結果をフィルタリングすることもできます。

```
edbctl project list [options]
```

options は次のとおりです。

```
- n, --name string         Project name (optional). Filter the list by a specific␣
↪project name.
- i, --id string           Project ID (optional). Filter the list by a specific␣
↪project ID.
- c, --credential string   The credential that you created via the `credential␣
↪create` command. The default is fetched from `context_credential`.
- o, --output string       [table json yaml xml] \(default "table")
- h, --help                Help for list.
```

この例は、すべてのプロジェクトをリストする方法を示しています。

```
edbctl project list
```

edbctl project create

指定された名前とオプションのタグを使用して新しいプロジェクトを作成します。このコマンドを使用して、プロジェクトを効率的に定義および編成します。

```
edbctl project create [options]
```

options は次のとおりです。

```
- n, --name string         Project name (required). The unique name for the project.
    --tags strings        Assign tags to the project in the format name=color (color␣
↪is optional).
- c, --credential string   The credential that you created via the `credential␣
↪create` command. The default is fetched from `context_credential`.
- y, --yes                 Auto-confirm all confirmations.
- h, --help                Help for create.
```

これらの例は、さまざまなオプションを使用してプロジェクトを作成する方法を示しています。

```
edbctl project create "My Project"
edbctl project create --name "My Project" --tags name2=blue
edbctl project create --name "My Project" --tags name2=#FF0000
```

edbctl プロジェクトの編集

名前または ID を使用してプロジェクトを更新します。プロジェクトの名前を変更し、新しいタグを割り当てることができます。更新する前に、正しいプロジェクト名または ID を指定してください。

```
edbctl project edit {--id|--name} --new-name [options]
```

options は次のとおりです。

```
- i, --id string           Project ID (required if --name isnt provided). The␣
↪immutable unique identifier for a project.
```

(次のページに続く)

(前のページからの続き)

```
- n, --name string         Project name (required if --id isnt provided). The mutable␣
↪unique identifier for a project.
- N, --new-name string     New name for the specified project.
    --tags strings        Assign tags to the project in the format name=color (color␣
↪is optional).
- c, --credential string   The credential that you created via the `credential␣
↪create` command. The default is fetched from `context_credential`.
- y, --yes                 Auto-confirm all confirmations.
- I, --interactive         Execute command interactively.
- h, --help                Help for edit.
```

これらの例は、プロジェクトを更新する方法を示しています。

```
edbctl project edit --id prj_MFq4M4EWEoalBAL3 --new-name "New Name"
edbctl project edit --name "Old Name" --new-name "New Name"
edbctl project edit --name "Old Name" --tags name2=blue
```

edbctl project delete

名前または ID を使用して、指定されたプロジェクトを削除します。プロジェクトを削除する前に、そのすべてのクラスターを完全に削除する必要があります。このアクションは元に戻せないため、続行する前に正しいプロジェクト名または ID があることを確認してください。

```
edbctl project delete {--id|--name} [options]
```

options は次のとおりです。

```
- i, --id string           Project ID (required if --name isnt provided). The␣
↪immutable unique identifier for a project.
- n, --name string         Project name (required if --id isnt provided). The mutable␣
↪unique identifier for a project.
- c, --credential string   The credential that you created via the `credential␣
↪create` command. The default is fetched from `context_credential`.
- y, --yes                 Auto-confirm all confirmations.
- h, --help                Help for delete
```

例

```
edbctl project delete --id prj_MFq4M4EWEoalBAL3
edbctl project delete --name "My Project"
```

Using the API

システム管理者と開発者は、 EDB Postgres AI Hybrid ManagerHM API を使用して、管理タスクのために HM と直接統合できます。これらのタスクには、クラスターの管理プロビジョニング、プロビジョニング解除、およびスケーリング、管理操作の自動化、および移行の処理が含まれます。

詳細については、 EDB Postgres AI ポータルの API ドキュメントを参照してください。

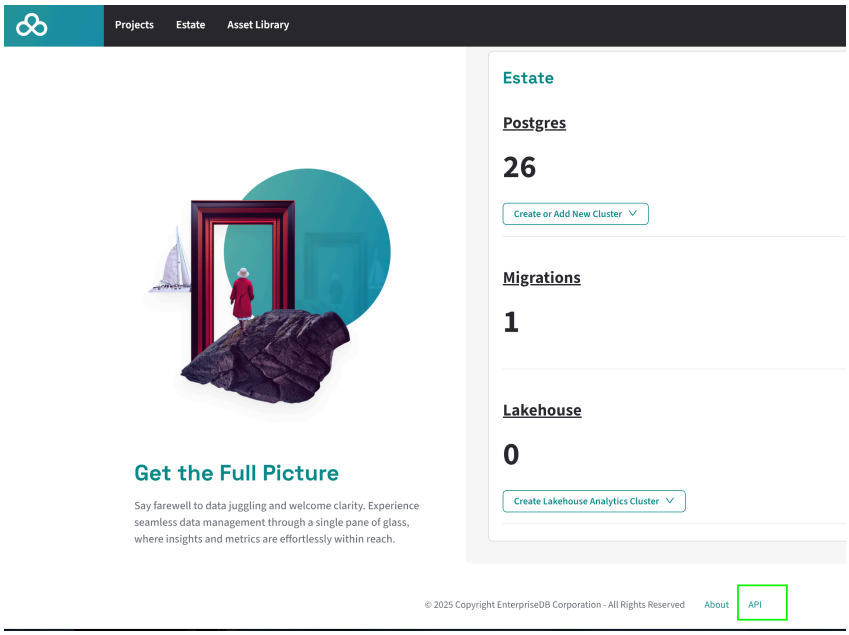


図 12: API Docs

1. Hybrid Manager コンソールの URL に移動し、資格情報を使用してログインします。
2. ページの下部で `API` リンクを探します。
3. `API` リンクを選択して、インタラクティブ API エクスプローラーを開きます。

使用可能なすべてのエンドポイントを表示し、必須パラメーターを確認し、ブラウザーから直接サンプル要求を実行することもできます。

アクセスキー

アクセスキーは、HM API への安全なプログラムによるアクセスを提供します。これにより、ユーザーは毎回対話型ログインを必要とせずに、HM サービスを認証および対話できます。

主な目的

- 認証 — API を呼び出すユーザーまたはシステムの ID を検証します。
- 自動化 —スクリプト、ツール、またはアプリケーションは、手動介入なしでクラスターのプロビジョニング、スケーリング、モニタリングなどのタスクを安全に実行できます。
- アクセス制御 —アクションを特定のユーザーまたはマシンに関連付け、監査証跡と権限の適用を有効にします。
- セキュリティ —自動化スクリプトにユーザー名とパスワードを埋め込む必要性を回避します。

詳細については、 *Create and manage Access key* を参照してください。

Access key

アクセスキーは、ユーザーがハイブリッドマネージャー HM API と直接対話するための認証を提供します。各アクセスキーは単一のユーザーにリンクされ、作成後に変更できません。

マシンユーザーまたは通常のユーザーのアクセスキーを作成できます。組織の所有者は、マシンユーザーのアクセスキーを管理できます。通常のユーザーは自分のアクセスキーを管理できます。

各キーには、1〜365 日の有効期限が作成時に設定される必要があります。アクセスキーの有効期限が切れたら、または紛失した場合、削除して新しいものを作成する必要があります。

User type	Quota - max keys	Keys created and managed
Machine user	2	Two access keys, created while *Creating a machine user* . Optionally, can be individually added from the **Access keys** tab on the machine user's details page using the **Create New Key** option.
Normal user	1	One access key, *1.Fernet* キーを生成しますまだ作成していない場合 and managed by the user from their home page.

アクセスキーはユーザーから取り消すことができます。キーを取り消しても、リンクされたユーザーのロールまたはその他の認証プロセスには影響しません。

個人用アクセスキーを作成します

1. HM コンソールにログインします。
2. 右上の組織名の横にあるメニューから、 **My Account** を選択します。
3. **My Account** ページから、 **Access Keys** タブを選択します。
4. [**Access Keys**] タブで、[**Create New Key**] を選択します。
5. **Access Key Name** の値を入力します。
6. **有効期限** の値を 1〜365 の範囲で入力します。
7. **Create Key** を選択します。

生成されたアクセスキーはユーザーに提供されます。

このアクセスキーをコピーし、安全な場所に保存します。アクセスキーは、作成した場合にのみ使用できます。アクセスキーを紛失した場合は、削除して新しいものを作成する必要があります。

アプリ内の受信ボックスまたはメール通知を有効にして、個人キーの有効期限が間もなくエキスパイアと通知されるようにできます。詳細については、*Managing notifications* を参照してください。

アクセスキーの管理

Users ページでマシンユーザーのアクセスキーを管理できます。

アクセスキーの詳細を表示するには

1. 右上の組織名の横にあるメニューから、 **User Management** を選択します。

2. アクセスキーの詳細を表示するユーザーの横にある **Edit** を選択します。
3. ユーザーのページにある **Access Keys** タブを表示します。各行は、アクセスキーの詳細を提供します。
    - **Name** — アクセスキーの名前。
    - **Access Key** — 半分暗号化された形式のアクセスキー。
    - **Expiry at** — アクセスキーの有効期限。
    - **作成日** — アクセスキーの作成日。

キーを削除するには、キーの横にある **Delete** を選択します。削除の確認を求められます。

キーを作成するには、 **Access Keys** タブの右上にある **Create New Key** を選択します。

アクセスキーを使用する

HM API は、呼び出し元が要求でアクセスキーを使用しているかどうかを検出します。呼び出し元がアクセスキーを使用する場合、API は以下をチェックします。

- アクセスキーの有効性。
- アクセスを要求しているユーザーの組織。
- ユーザーの権限。

次に、アクセスを承認します。詳細については、*Using the access key* を参照してください。

Using Access Keys to access GenAI Builder

GenAI Builder API と対話するには、アクセスキーを使用して要求を認証する必要があります。このキーは、ベアラートークンまたは `x-access-key` として要求ヘッダーに提供できます。

1. アクセスキーを生成する

UI または CLI を介したハイブリッドマネージャーの *Create an access key*

重要 キーは生成後に表示されないため、安全に保存してください。

2. API 要求でアクセスキーを使用する

GenAI Builder API エンドポイントのベース URL は、通常、`${BASE_URL}/launchpad/apps/genai-builder/api` として構造化されています。ここで `${BASE_URL}` は、Hybrid Manager インスタンスのドメインまたは IP アドレスを表します。

`curl` のようなツールを使用して、API アクセスをテストできます。アクセスキーは、2 つの方法のいずれかで HTTP ヘッダーに含めて送信する必要があります。

- **ベアラートークンとして** `Authorization: Bearer ${access_key}`
- **x-アクセスキーとして** `x-access-key: ${access_key}`

次に、 `x-access-key` ヘッダーを使用してユーザーのリストを取得する例を示します。

```
curl -H "x-access-key: ${access_key}" \
     -k "${BASE_URL}/launchpad/apps/genai-builder/api/users"
```

そこで、

- ${access_key} は、実際のアクセスキーです。
- ${BASE_URL} は、Hybrid Manager インスタンスのドメインまたは IP アドレスです。 **注** -k オプションを使用すると、curl は自己署名 TLS 証明書を操作できます。信頼できる証明書を使用している場合は、削除します。

API 応答の例 ユーザーの取得

上記の curl の例に示すように、ユーザーエンドポイントへの要求が成功すると、次のような JSON 応答を受け取る場合があります。


```
{
    "users": [
        {
            "user_id": "d9e03312-638d-52b6-8d6f-f5b99f5f2480",
            "created_at": "2025-06-25T19:27:32.642823Z",
            "updated_at": "2025-06-26T08:50:42.122521Z",
            "organizations": [
                "060e360b-0364-47b6-adc2-f62ac0feae41"
            ],
            "name": "owner@mycompany.com",
            "email": "owner@mycompany.com"
        }
    ]
}
```


Using the HM API

HM API を使用して、クラスターのプロビジョニング、プロビジョニング解除、スケーリングなどの管理アクティビティを行うために HM と直接統合します。

API リファレンスドキュメントは、 EDB Postgres AI ポータルから入手できます。

前提条件

API を呼び出す前に、アクセスキーを作成します。詳細については、*Access key* を参照してください。

API を呼び出す

HM API で認証するには、アプリケーションは、取得したアクセスキーを各 HTTP 要求の x-access-key ヘッダーに含める必要があります。例

```
curl --request GET \
  --url "https://portal.edb-hcp.com/api/v1/projects" \
  --header "x-access-key: $ACCESS_KEY"
```

ACCESS_KEY 変数は、コンソールからコピーしたアクセスキーのテキストです。

応答例

```
{
  "data": [
    {
      "projectId": "string",
      "projectName": "string",
      "userCount": 0,
      "clusterCount": 0,
      "tags": []
    }
  ]
}
```

6.3.4 For Kubernetes Administrators

Managing users in Hybrid Manager

Hybrid ManagerHM は、HM コンソールでのユーザー認証に Dex を使用し、HM とネイティブユーザーまたは LDAP や SAML のような外部ユーザー管理システム ID プロバイダーまたは IdP 間のプロキシとして機能します。

HM の初期ブートストラップ中に、管理者は HM コンソールアクセス用のネイティブスーパーユーザーを設定します。このネイティブスーパーユーザーを使用して、コンソールに初めてサインインします。

テストまたはデモ環境の場合、必要に応じてネイティブユーザーを追加することにより、操作を簡素化できます。

- 詳細については、 *Creating native users for testing* を参照してください。

システムロールをさまざまなユーザーに割り当てるために、HM はロールベースのアクセス制御 RBAC モデルを使用します。

- 詳細については、 *User roles and authorization* を参照してください。

運用展開の場合、 LDAP、SAML、OIDC、または別の supported connector のような外部 IdP を構成することを強くお勧めします。これにより、堅牢で安全なユーザー管理が保証されます。これは、インストール時に行う必要があります。選択した IdP を構成し、組織の新しい管理者を追加した後、ネイティブユーザーログインを無効にします。

- 詳細については、 *Configuring an IdP* を参照してください。

Managing native users for testing

テストまたはデモ環境の場合、ネイティブユーザーを使用して操作を簡素化できます。ネイティブユーザーのライフサイクルを管理するには、次のことができます。

- *Changing password for native users* 。
- *Creating native users for testing* 。

運用展開の場合、configure an external IdP を強くお勧めします。IdP を構成すると、次のことができます。

- *Disabling native user login* 。

Changing password for native users

環境への不正アクセスを回避するには、展開中または展開の直後にデフォルトのネイティブユーザーのパスワードを変更します。同じ方法で手動で追加したネイティブユーザーのパスワードを変更できます。

1. 新しい bcrypt ハッシュ化されたパスワードを作成します。この値を生成するには、次を使用します。

```
echo ${password} | htpasswd -BinC 10 admin | cut -d: -f2
```

新しいパスワードが${password} 変数の後ろに保存される場所。

出力には、新しいハッシュされたパスワードたとえば$2y$10$YDHMEnc8VZKnhvze538ePOLKol7dFH.S7IgpqC3.X3R/3P2Q9zRcG が印刷されます。

1. 既存のユーザーこの例では owner@mycompany.com のハッシュを置き換えます。使用するインストール方法に応じて、values.yaml ファイルまたは HybridControlPlane カスタムリソースのこの値を置き換えます。

```
portal:
  authentication:
    staticPasswords:
      - email: "owner@mycompany.com"
        hash: "$2y$10$YDHMEnc8VZKnhvze538ePOLKol7dFH.S7IgpqC3.X3R/    3P2Q9zRcG"
        username: "owner@mycompany.com"
        userID: "c5998173-a605-449a-a9a5-4a9c33e26df7"
```

1. Hybrid ManagerHM をインストールする場合、インストールの実行を続行します。既に展開されているインストールのパラメーターを変更する場合は、

apply the DEX configuration を使用して変更を入力します。

Creating native users for testing

テストまたはデモ環境の場合、ネイティブユーザーを追加することにより操作を簡素化できます。運用展開の場合、

configuring an external IdP を強くお勧めします。

現在、Hybrid ManagerHM コンソールで人間ユーザーを作成することはできません。これは、HM 構成ファイルを調整することによってのみできます。

ネイティブユーザーの追加

インストール中または後に、新しいユーザーを `portal.authentication.staticPasswords` セクションに追加します。使用するインストール方法に応じて、`values.yaml` ファイルまたは `HybridControlPlane` カスタムリソースのこの値を置き換えます。

```
portal:
  authentication:
    staticPasswords:
      - email: "owner@mycompany.com"
        hash: "$2y$10$STTzuGJa3PydqHvlzi2z6OgDU1G/JLTqiuGblH.RemIutWxkztN5m"
        username: "owner@mycompany.com"
        userID: "c5998173-a605-449a-a9a5-4a9c33e26df7"
     - email: "<userA@example.com>"
       hash: "<example_hashed_password>"
       username: "<userA@example.com>"
       userID: "<unique_user_id>"
```

Apply the DEX configuration を使用して変更を入力します。

パラメーター

これらの設定は、`portal.authentication.staticPasswords` エントリーのユーザー用です。各ユーザーには、`email` 、`hash` 、`username` 、および `userID` の値が必要です。

Parameter	Description
email	ユーザーのメールアドレス。 HM コンソールにアクセスするためのユーザーのログイン識別子としても機能します。
hash	パスワードストアの Bcrypt ハッシュ化されたユーザーパスワード。この値を生成するには、 `echo ${password} I htpasswd -BinC 10 userA I cut -d: -f2`を使用します。ここで、実際のパスワードは`${password}`変数の後ろに保存されます。 `userA`は、パスワードのハッシュプロセス中に使用されるユーザー名を表します。構成内の他の場所で使用されていないため、任意のテキストにすることができます。結果のハッシュのみが使用されます。
userID	静的パスワードで構成された各新しいユーザーには、ユーザーの一意の識別子 uuid が必要です。 UUID ジェネレータツールでこの値を生成するか、文字のランダムなシーケンスを手動で割り当てることを選択できます。
username	ユーザーの一意のユーザー名。これは、HM コンソールにログインするためのプライマリ識別子です。 `email`と同じでもかまいません。

Disabling native user login

ネイティブユーザーログインを無効にする前に、少なくとも 1 人の他の IdP ユーザーが組織所有者ロールがあることを確認してください。新しい組織の所有者をまだ割り当てていない場合は、割り当てます。

新しい組織の所有者を割り当てます

1. 組織の所有者になる IdP ユーザーとして Hybrid ManagerHM コンソールに少なくとも 1 回ログインします。これにより、HM コンソールは新しいユーザーのインデックスを作成します。次に、ログアウトします。
2. HM の現在のネイティブ所有者たとえば `owner@mycompany.com` としてログインします。
3. 右上のプロファイルメニューから、 **User Management** を選択します。新しい所有者になるユーザーがリストにあることを確認します。
4. ユーザーを編集し、所有者ロールを割り当てます。

ネイティブユーザーを無効にする

1. ネイティブユーザーログインを無効にするには、`portal.authentication.staticPasswords` を `[]` に設定して `staticPasswords` 構成の設定を解除します。

```
portal:
  authentication:
    staticPasswords: []
```

1. Apply the DEX configuration を再度実行して、変更を入力します。

Applying modifications to the user configuration

ユーザー認証設定を変更した後、Dex 構成を調整し、サービスを再起動して変更を有効にする必要があります。これらの更新を調整する方法は、ハイブリッドマネージャー HM をインストールした方法によって異なります。Helm チャートを介して、または EDB Operator を使用して。

Helm チャートのインストール

Helm チャート方法を使用してインストールされた HM インスタンスの場合

1. 更新された `values.yaml` で Helm チャートを再適用します。GCP インストールでこれを行う方法の例については、*Helm* **チャートを使用してハイブリッドマネージャーをインストールする** を参照してください。
2. Dex を再起動してリロードし、新しいユーザーを追加します。

```
kubectl rollout restart deploy upm-dex -n upm-dex
```

オペレーターのインストール

Operator 方法を使用してインストールされた Hybrid Manager インスタンスの場合

1. `HybridControlPlane` カスタムリソース CR を再適用します。RHOS インストールでこれを行う方法の例については、*Install Hybrid Manager using Operator* を参照してください。
2. Dex を再起動してリロードし、新しいユーザーを追加します。

```
kubectl rollout restart deploy upm-dex -n upm-dex
```

User roles and authorization

複数のユーザーロールをハイブリッドマネージャー HM ユーザーに割り当てることができます。参照

- *User role permissions by use case*
- *User role descriptions*

User role descriptions

Hybrid ManagerHM には、人間ユーザーネイティブユーザーまたは IDP ユーザーとマシンユーザーの 2 種類のユーザーがいます。

マシンユーザーのみがエステートインジェスターになれることを除き、両方のタイプのユーザーにデフォルトのロールを割り当てることができます。

ロールの 2 つの主要なカテゴリは、組織レベルとプロジェクトレベルです。

組織レベルのロール

HM コンソールのナビゲーションバーの右上にあるユーザープロファイルメニューの**ユーザー管理**オプションを使用して、これらのロールを人間またはマシンユーザーに割り当てることができます。

組織管理者 admin

このロールは、組織レベルでの読み取りアクセスを提供し、ユーザーは組織全体の情報と設定を表示できます。

組織の所有者 owner

このロールの目的は、高いレベルで組織を管理することです。組織の所有者はプロジェクトを作成し、組織レベルのロールを他のユーザーに割り当てることができます。

プラットフォーム管理者 platform admin

このロールは、HM の基になるプラットフォームコンポーネントにアクセスして管理する必要があるユーザーを対象としています。プラットフォームの管理および監視ツールにアクセスできます。

GenAI Builder ユーザー GenAI Builder ユーザー

このロールは、HM の GenAI Builder 機能を使用するユーザーを対象とており、その機能への完全なアクセスを付与します。

プロジェクトレベルのロール

これらのロールは、人間またはマシンユーザーに割り当てることができますエステートインジェスターを除きます。プロジェクトを表示するときに、左側のナビゲーションで **Users** を選択します。

プロジェクト所有者 owner

特定のプロジェクトでは、プロジェクト所有者が最高レベルの権限を持っています。彼らはそのプロジェクト内のすべてのアクションを実行でき、プロジェクトレベルのロールを他のユーザーに割り当てます。

プロジェクト編集者 Editor

このロールは、プロジェクト内のデータをアクティブに作業する必要があるユーザーを対象としています。プロジェクト編集者には、データの読み取りおよび書き込みアクセスがあります。

プロジェクトビューア viewer

プロジェクト内のデータを観察することだけが必要なユーザーに、プロジェクト閲覧者ロールはデータの読み取り専用アクセスを提供します。

エステートインジェスター estate ingester

このロールはマシンユーザー用であり、プロジェクトでエステートの取り込みを実行するためのアクセスを提供します。

カタログデータリーダー catalog data leader

このロールを持つユーザーは、プロジェクトスコープのカタログの Iceberg 名前空間/テーブル/ビューを読み取ることができます。

カタログデータライタ catalog data ライタ

このロールにより、ユーザーは、プロジェクト内のすべてのプロジェクトスコープのカタログの Iceberg 名前空間/テーブル/ビューの読み取り、書き込み、および削除ができます。

Migration Portal プロジェクトの所有者

このロールは、HM プロジェクトで移行ポータルプロジェクトを管理するユーザーを対象としています。これらのプロジェクトを作成、読み取り、更新、および削除できます。

Migration Portal プロジェクトエディター

このロールを持つユーザーは、HM プロジェクトの移行ポータルプロジェクトを読み取りおよび更新できます。

Migration Portal プロジェクトビューア

このロールは、HM プロジェクトの移行ポータルプロジェクトへの読み取り専用アクセスを提供します。

User role permissions by use case

これらのユーザーロールの承認は、特定のスコープ 1 つのプロジェクト内または 1 つのアカウント内のいずれかに制限が適用されるロールベースのアクセス制御 RBAC モデルに従います。

次のリストは、Postgres クラスターデータベースの承認をカバーしていません。

現在、カスタムロールを作成することはできません。これらの 11 の事前定義されたロールのみが使用できます。

- マシンユーザーのみが自己管理クラスターデータの取り込みに割り当てることができます。

Monitoring with Hybrid Manager

Hybrid ManagerHM を使用すると、さまざまなメトリックホストメトリック、Kubernetes メトリック、Postgres データベースデータ、クエリデータ、ロギングデータを収集できます。収集されたデータを HM コンソールまたは Grafana インスタンスで視覚化して、傾向の分析、アラートの構成、データベースの効率の向上、トラブルシューティングの実行などを行うことができます。

監視アーキテクチャの理解

HM には、モニタリング機能を提供する 3 つの主要な内部パイプラインがあります。

- Prometheus と Thanos はメトリックを収集します。
- Loki はログを収集します。
- Stats Collector はクエリデータを収集します。

これらのサービスはすべて、独自の専用ストアが付属しています。これは、収集されたデータが HM に保持されることを意味します。これらのサービスとストアはデフォルトで展開されます。

これらの収集されたメトリックは、2 つのインターフェイスを介して利用できるようになります。

- **Grafana** ダッシュボード。
- **HM console** たとえば、 **Estate** タブまたは **Cluster** view 。

データベースモニタリングの有効化

HM に付属する監視サービスは、監視するデータベースへのアクセスと、Stats Collector を有効にするための拡張機能が必要です。

- HM 管理クラスター HM で作成の場合、デフォルト設定は監視スタックへのアクセスを提供します。
- 外部自己管理クラスターの場合、データ収集を有効にするには、各データベースを registered with the HM agent にする必要があります。デフォルトでは、エージェントは Prometheus、Thanos、および Loki データを収集するための構成を提供します。さらに、エージェントは構成パラメーターを提供するため、Stats Collector クエリデータの収集に必要な拡張機能、つまり `edb_wait_states_data` 、`edb_stat_monitor` 、`edb_query_advisor` 、および `pg_stat_statements` を有効または無効にするかどうかを制御できます。
- AWS RDS などの外部クラウドサービスプロバイダー CSP 管理データベースの場合、HM は Kubernetes `secret` リソースを介してデータベースにアクセスして、データ収集を有効にする必要があります。詳細については、Monitoring CSP-managed databases を参照してください。 Stats Collector を使用したクエリデータは、CSP 管理データベースでは現在使用できません。

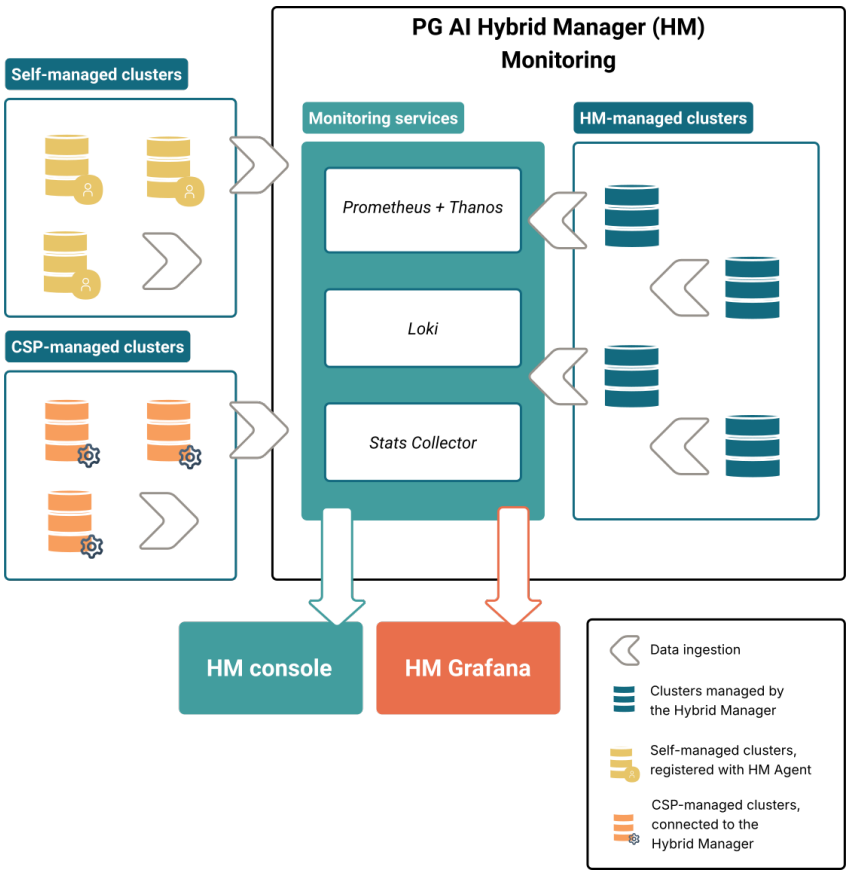


図 13: Monitoring infrastructure in Hybrid Manager

データの取り込みとスコープの監視

さまざまなクラスタータイプに対して HM のアクセスレベルが異なるため、データが取り込まれる方法、および収集されるデータの範囲は異なります。

- HM 管理クラスター HM で作成の場合、データベースはホストメトリック、Postgres メトリック、クラスターメトリック、クエリメトリック、レコメンデーションに必要なデータ、およびログを、対応する Prometheus、Thanos、Loki、および Stats Collector ストアにプッシュします。
- 外部自己管理データベースの場合、エージェントはホストメトリック、Postgres メトリック、クラスターメトリック、クエリメトリック、レコメンデーションに必要なデータ、およびログを構成可能な間隔で対応する Prometheus、Thanos、および Stats Collector ストアにプッシュします。
- AWS RDS などの外部 CSP 管理データベースの場合、HM は HM からデータベースへの接続を活用して、ホストおよび Postgres のメトリックを中央ストアにプルします。Stats Collector を使用したクエリデータは、CSP 管理データベースでは現在使用できません。

詳細については、 *Metrics support scope* を参照してください。

監視リフレッシュ頻度

HM によって収集されるメトリックの更新間隔は、メトリックとデータベースタイプによって異なります。この違いは、データ収集のさまざまな方法と、Prometheus / Thanos、Loki、および Stats Collector サービスの統合によるものです。

メトリックが更新される頻度と頻度をカスタマイズする方法については、*Metrics refresh intervals* を参照してください。

データを外部ツールに接続する

現在、EDB は、HM によって収集されたメトリックとログデータの外部ツールまたはエンタープライズシステムへのプラグインをサポートしていません。

関連トピック

- *Enabling monitoring* ☒エージェントをインストールするか、データベースを HM に接続して、外部データベースでのモニタリングを有効にします。
- *Metrics refresh intervals* ☒メトリックが更新される頻度と、頻度をカスタマイズする方法を理解します。
- Monitoring with Grafana ☒ Grafana を使用してデータベースをモニタします。
- Monitoring with HM console ☒ HM コンソールを使用してデータベースをモニタします。
- *Notifications* ☒ 通知を構成します。

Enabling monitoring

HM 管理のデータベースクラスター

Hybrid Manager (HM)-managed database clusters は HM と同じ環境で作成されているため、それらのリソースは既にアクセス可能です。したがって、これらのクラスターはデフォルトで監視され、展開が完了する

とすぐに監視が開始されます。

外部データベースクラスターの場合

External database clusters は、クラウドサービスプロバイダー CSP またはお客様によって管理されます。このため、データベースクラスターと HM インスタンス間の接続を作成するには、追加の手順が必要です。

モニタできる外部データベースには 2 種類があります。

- **自己管理展開のモニタリング**

enable monitoring on self-managed deployments には、エージェントをインストールし、外部自己管理データベースを登録する必要があります。

例ベアメタル上にある自己管理の Postgres データベース、または CSP によって管理されていないクラウドでホストされているデータベース EC2 インスタンスの Postgres データベースなど。

- **CSP 管理データベースの監視**

enable monitoring on CSP-managed databases 、データベースを HM に接続します。

例 Amazon RDS。

!!!note

データベースをエージェントに assess them for migration に登録することもできます。

Monitoring self-managed deployments

ベアメタルまたは仮想マシンに展開されている自己管理の外部 Postgres インスタンスをハイブリッドマネージャー HM からモニタするには、最初に Beacon エージェントをインストールする必要があります。エージェントは、モニタリングと可観測性メトリックを HM に提供します。

自己管理の Postgres インスタンスのモニタリング、クエリ診断、および推奨事項を行うには

1. *Create an OS user to run the agent.*
2. *Prepare your database.*
3. *Create a machine user.*
4. *Installation phase of Hybrid Manager on Rancher RKE2*
5. Configure the agent.
6. *Run the agent.*
7. *Configure the agent to run as a service.*

!!!note 可観測性機能を完全に活用するには、自己管理データベースクラスターを使用してホストマシンごとに 1 つのエージェントを展開することをお勧めします。次に、専用の DSN 文字列を使用して各データベースに接続するようにそのホスト上のエージェントを構成できます。DSN 構成の詳細は、 Configuring your agent で提供されています。

エージェントの構成に必要な管理タスクの詳細については、*Administrative tasks* を参照してください。

一般的なトラブルシューティングのヘルプについては 障害対応 を参照し、エージェントによってデータが保存される方法を理解するには *Data security and sovereignty* を参照してください。

Create an OS user to run the agent

エージェントを構成および実行するための専用のオペレーティングシステムユーザーを作成することをお勧めします。これは、機能する必要のないエージェントに権限を付与することを回避するため、セキュリティのベストプラクティスです。

ユーザーを作成するには

```
sudo useradd <agent_os_user>
```

エージェント OS ユーザーを Postgres ユーザーのグループに追加することをお勧めします。これにより、後でエージェントに Postgres ログファイルへのアクセスを許可することが簡単になります。

```
sudo usermod -aG postgres <agent_os_user>
```

Preparing your Postgres database for monitoring

監視ユーザー

エージェントを構成するときは、最初に監視する自己管理データベースに接続し、エージェントがデータベースへの接続に使用するユーザーを作成する必要があります。このユーザーは、pg_monitor ロールと pg_read_all_data ロールを持つ Postgres データベースユーザーで、ターゲットデータベースに対する CONNECT 特権があります。

この例では、ユーザー agent_user を作成します。

```
CREATE USER agent_user WITH PASSWORD <password>;
GRANT CONNECT ON DATABASE <target_database> TO agent_user;
GRANT pg_monitor, pg_read_all_data TO agent_user;
```

<target_database> は、モニタするデータベースの名前です。

Postgres 16 以前を使用している場合、次の権限を付与して、エージェントが Postgres ログファイルの場所を検出できるようにする必要があります。

```
GRANT EXECUTE ON FUNCTION pg_catalog.pg_current_logfile() TO agent_user;
```

エージェントユーザーが、接続する予定のデータベースにログインできることを確認する必要があります。たとえば、hr というデータベースに接続する予定がある場合

```
GRANT CONNECT ON DATABASE hr TO USER agent_user;
```

edb_internals スキーマと必要な拡張機能

Hybrid ManagerHM の完全なモニタリングと最適化機能を活用するには、Postgres にインストルメンテーションを追加する拡張機能を有効にする必要があります。これらの拡張機能は、**Query Diagnostics** と **Recommendations** タブ、および一部のメトリックを有効にするために必要です。

拡張機能をインストールするには

1. プラットフォームおよび Postgres ディストリビューションの正しいコマンドについては、拡張機能ごとのインストール手順を参照してください。

-

EDB Wait States

-

EDB Query Advisor

-

EDB Stat Monitor

さらに、パブリックスキーマで pg_stat_statements を有効にする必要があります。pg_stat_statements は Postgres とともにインストールされるため、追加のインストールは必要ありません。

例、RHEL 9 で EDB Advanced Server 17 をモニタするには

```
dnf install edb-as17-server-edb_wait_states \
            edb-as17-query-advisor \
            edb-as17-stat_monitor
```

1. データベースに接続しているときに、必要なプリロードモジュールを Postgres shared_preload_libraries に追加します。例、shared_preload_libraries の現在値が空の場合

```
ALTER SYSTEM SET
shared_preload_libraries="$libdir/pg_stat_statements","$libdir/edb_wait_states", "
↪$libdir/query_advisor","$libdir/edb_stat_monitor";
```

1. スーパーユーザーとして、エージェントが接続するデータベースにスキーマを作成し、pg_stat_statements 拡張機能と前の手順でインストールした拡張機能を有効にします。

```
CREATE SCHEMA edb_internals;
CREATE EXTENSION edb_wait_states WITH SCHEMA edb_internals;
CREATE EXTENSION query_advisor WITH SCHEMA edb_internals;
CREATE EXTENSION edb_stat_monitor WITH SCHEMA edb_internals;
CREATE EXTENSION pg_stat_statements;
```

1. edb_internals スキーマを検索パスに追加します。例、スーパーユーザーの場合

```
ALTER SYSTEM SET
search_path="$user", public, edb_internals;
```

1. エージェントユーザーにこのスキーマへのアクセスを付与します。

```
GRANT USAGE ON SCHEMA edb_internals TO agent_user;
GRANT SELECT ON ALL TABLES IN SCHEMA edb_internals TO agent_user;
GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA edb_internals TO agent_user;
```

Postgres ロギング構成

Postgres ログを収集するには、Postgres ロギングコレクターを有効にし、エージェントにログファイルを読み取る権限が必要です。これを行う最も簡単な方法は、Postgres ユーザーとしてエージェントを実行することです。ただし、セキュリティのベストプラクティスとして、 *Create an OS user to run the agent* を使用してエージェントを実行することをお勧めします。

エージェント OS ユーザーがログファイルを読めるようにするには、ログがデータディレクトリの外部にあり、適切な権限を持っている必要があります。たとえば、`postgresql.conf` で次のパラメーターを設定すると、ログが`/var/log/postgres-lc` に書き込まれ、所有者のグループのメンバーであるすべてのユーザーが読み取ることができます。

```
logging_collector=on
log_directory=/var/log/postgres-lc
log_file_mode=0640
log_destination = jsonlog
```

新しいロギングの場所は、Postgres ユーザーが所有し、そのユーザーのグループが読み取り可能である必要があります。エージェント OS ユーザーは同じグループのメンバーであるため、ログディレクトリへのアクセスも取得します。例

```
mkdir /var/log/postgres-lc
chown postgres:postgres /var/log/postgres-lc
chmod 0750 /var/log/postgres-lc
```

これらの変更が完了したら、Postgres を再起動します。

Creating a machine user

Hybrid ManagerHM コンソールで、エージェントがエステートにデータを取り込むために使用できるマシンユーザーを作成します。

マシンユーザーの作成

1. 右上隅にあるプロフィールアイコンを選択します。
2. **User Management > Add New User** を選択します。
3. **Add New User** ページで、新しいユーザーの名前を入力します。
4. オプションで、ユーザーの電子メールアドレスを入力します。
5. **Create Access Key** を選択します。フォームが展開され、**アクセスキー名** と **有効期限** 時間を入力できます。

6. **Add User** を選択します。
7. アクセスキーは安全に保存します。キーに再度アクセスすることはできません。**キーは安全に保存されました** を選択します。

ロールの割り当て

1. **Projects** タブを選択して、すべてのプロジェクトを表示します。
2. データベースをモニタするプロジェクトを選択します。
3. 左側のナビゲーションで **Users** オプションを選択し、作成したばかりのマシンユーザーを見つけます。
4. ユーザーエントリの編集ボタンペンアイコンを選択します。
5. **役割の割り当て** を選択します。
6. **Assign Project Roles** ウィンドウで、 **estate ingester** を選択します。
7. **Submit** を選択します。

新しいマシンユーザーは、エージェントからデータを取り込む準備が整います。

Installing the Agent

エージェントのインストール方法については、*Installation phase of Hybrid Manager on Rancher RKE2* を参照してください。

Configuring the HM agent

<beacon_server>:9443	Host URL that allows the agent to connect to the correct service endpoint. Obtain the host URL of your beacon server from an administrator, the port is fixed as 9443. See *Administrative tasks* for more information.
<project_id>	To obtain your project's ID, go to the HM console, select **Projects**, and select your project. Your browser displays a URL like *https://portal.example.com/projects/prj_5Wuqyl5JtpvxjiYC/clusters*. The project ID is the identifier starting with "prj_" in the URL of that page. In this example: *prj_5Wuqyl5JtpvxjiYC*.
N/A	Set to - *"onprem"*.
N/A	Set to - *"onprem-settings"* to enable recommendations for Postgres configurations.
N/A	Set to - *"onprem-schema"* to enable schema ingestion and migration assessment capabilities. Remove this section if you don't want to allow schema information to be collected.
N/A	Set to "" if your organization has configured the machine where HM agent runs to trust HM's TLS certificate. Otherwise, obtain the root certificate from an administrator and store it locally. Then, set this value to the path where you stored it. See *Administrative tasks* for more information.

次のページに続く

表 17 – 前のページからの続き

N/A	Set to *"info"* or *"trace"* to obtain logging information. Start with *"info"* and only use *"trace"* if required, as it is extremelly verbose.
N/A	Enables pushing metrics to the HM console. The *push_endpoint* is the same as *agent.beacon_server*. The *root_ca_path* is the same as *agent.root_ca_path*.
N/A	Enables storing usage data (and reporting usage data, *Troubleshooting AI Factory on Hybrid Manager*). Your *company_code* can be found on the EDB Support Portal .
<location_id>	A unique identifier used by the Hybrid Manager for this agent/location. Consider setting this to the same value as *provider.onprem.host.resource_id* for convenience.

これらの設定は、データベース構成に関連します。

YAML database settings	Placeholder	Guidelines
clusters.resource_id	*<cluster_resource_id>*	Assign a unique identifier to your database cluster. Encase in double quotes. You must begin identifiers with an alphanumeric character (*A-Z, a-z, 0-9*); in between you may also include hyphens (-), underscores (_) and periods (.). You must not use any other characters. Example: *An_I.D.-1*
clusters.name	*"<instance_name>"*	Assign a unique name to your database cluster.
clusters.manager	*"other"*	Collects statistics for cluster manager. Possible values are *"patroni"*, *"repmgr"*, *"efm"*, *"other"*. For self-managed instances, set to *"other"*.
clusters.nodes.resource_id	*<node_resource_id>*	Assign a unique identifier to your database node. Single-node instances require a single entry here. Multi-node instances require an entry per node. Encase in double quotes. Reuse the same id when configuring the DMS agent for migrations. This consistency is essential for the DMS to correctly correlate the resource across both the HM agent and DMS agent components. You must begin identifiers with an alphanumeric character (*A-Z, a-z, 0-9*); in between you may also include hyphens (-), underscores (_) and periods (.). You must not use any other characters. Example: *An_I.D.-1*
clusters.nodes.dsn	N/A	Connection string or DSN that provides access to your remote database. The DSN must follow the format: *<database_type>://<user>:<password>@<host>:<port>/<database_name>* The *database_type* can be postgresql or oracle.
clusters.nodes.disable_host_ass	N/A	Set to *true* to prevent this database being associated with the host on which the agent is running. For example, if the agent is running on a different server.

次のページに続く

表 18 – 前のページからの続き

YAML database settings	Placeholder	Guidelines
clusters.nodes.tags	N/A	Assign tags for resource labeling. Encase in double quotes and introduce each tag with a dash.
clusters.nodes.settings.enabled	N/A	Set to *true* to enable collecting database configuration data.
clusters.nodes.metrics.[...]	N/A	Controls 時系列メトリック *[*モニタリング*]* タブ the agent ingests, and how they are ingested.

これらの設定はホストデータに関連しています。

YAML database settings	Placeholder	Guidelines
host.metrics.disabled	N/A	Set to *true* to disable the collection of host metrics from the host on which the agent is located. For example, when the agent host is not the same as the database cluster host.
host.resource_id	*<host_id>*	Assign an identifier to the host on which the agent is running, by default this will be populated by the hostname detected during setup. This will link your host machine to an identifier. Encase in double quotes. Must be unique within the project.

スキーマの取り込みに関連する設定

YAML database schema settings	Placeholder/Value	Guidelines
clusters.nodes.schema.enabled	N/A	Set to *true* to enable schema ingestion for the database. Set to *false* to disable it.
clusters.nodes.schema.poll_inte	N/A	Defines how frequently the agent checks for schema changes. Provide a value in seconds or minutes (for example, 15 s or 1m).
clusters.nodes.schema.filter	N/A	Use this block to specify which schemas to include or exclude from ingestion. It contains the *mode* and *names* parameters.
clusters.nodes.schema.filter.mo	*include* or *exclude*	Specifies the filtering behavior. Use *include* to only ingest listed schemas, or *exclude* to ingest all schemas except those listed.
clusters.nodes.schema.filter.nar	*<schema_name>*	A list of strings representing the names of the schemas to be filtered. This list works in conjunction with the mode setting.

関連トピック

- 時系列メトリック [モニタリング] タブ ☒モニタリング機能をカスタマイズ、それらのサブセットを有効/無効にする方法を学びます。
- 複数のクラスターとノードの構成 ☒同じエージェント構成ファイルを使用して、複数のデータベースに接続し、データを取得する方法を学習します。
- インターネットに接続された環境の場合 ☒製品の使用状況に関する基本的な洞察を EDB に提供することにより、モニタリング取り組みを補完する方法を学びます。
- Administrative tasks ☒ HM インスタンスの管理者がルート証明書と `beacon_server` の値を取得する方法を学習します。
-

Use the agent to assess your Oracle database for migration ☒

HM エージェントを使用して Oracle データベースを評価し、移行ポータルで変換するためのスキーマを抽出します。

Fine-tuning monitoring data collection

Configuring the agent で提供されるテンプレートは、すべてのモニタリング機能を有効にします。有効になっている機能を微調整できます。

時系列メトリック [モニタリング] タブ

構成ファイルの `general.metrics.push` セクションは、時系列メトリックの Hybrid ManagerHM の Prometheus インスタンスへのプッシュを制御します。これらのメトリックは、[**Monitoring**] タブに入力されます。これを無効にするには、`general.metrics.push.enabled` を `false` に設定します。

Postgres ログ [Logs] タブ

`general.metrics.push.include_logs` を `true` に設定して、ログ収集を有効にします。この値を `false` に設定することにより、エージェントレベルで無効にします。さらに、Postgres ログを収集する `provider.onprem.databases` のエントリごとに、`logs.disabled` ～`false` を設定します。

デフォルトでは、エージェントは Postgres ファンクション `pg_current_logfile()` を呼び出すことにより、ログファイルの場所を検出しようとします。別の場所にログを保存するように Postgres インスタンスを構成した場合など、ログファイルの場所を明示的に指定する場合は、<logs_directory> をログファイルを保存するディレクトリへのパスに置き換えます。

たとえば、ファイルパス `/var/log/postgres-lc` から拡張子 `.log` を持つログを収集するには、パターン `/var/log/postgres-lc/*.log` を使用できます。

```
clusters:
  - resource_id: selfmanaged_postgres
    ...
    nodes:
```

(次のページに続く)

(前のページからの続き)

```
    - resource_id: selfmanaged_postgres_node1
      ...
      logs:
        disabled: false
        mode: file
        file:
          paths: <logs_directory>/*log
```

Diagnostics [クエリーの診断と推奨事項] タブ

`provider.onprem.clusters.nodes` の各エントリの下の `metrics.stats` セクションは、クエリ分析と最適化のためのより詳細な診断データの収集を制御します。

`metrics.stats.enabled` を `false` に設定することにより、このようなデータ収集をすべて無効にできます。または、適切な `enabled` パラメーターを設定することにより、各要素を有効または無効にすることができます。

- `metrics.stats.wait_states` は、EDB 待機状態からセッション状態をサンプリングすることに基づいて、クエリーデータの収集を制御します。
- `metrics.stats.query_text` は、 EDB Stat Monitor からのクエリ統計の収集を制御します。

これらのオプションを両方とも無効にすると、[**Query Diagnostics**] タブが無効になります。

- `metrics.stats.recommendations` は、EDB クエリアドバイザーからのインデックスと統計の推奨事項の収集を制御します。無効にすると、[**Recommendations**] タブが空白になります。
- `metrics.stats.query_text` は、 EDB 待機状態と EDB Stat Monitor の両方からクエリテキストの収集を制御します。無効にすると、クエリテキストは収集されず、**クエリ診断** にクエリ ID のみが表示されます。

Configuring multiple databases

可観測性機能を完全に活用するには、自己管理データベースクラスターを使用してホストマシンごとに 1 つのエージェントを展開することをお勧めします。次に、同じホスト上の複数のデータベースクラスターに接続するようにこの単一のエージェントを構成できます。

複数のクラスターとノードの構成

エージェントは `clusters` セクションを使用して、ホスト上のすべての監視対象データベースクラスターを定義します。

- **クラスターエントリ**クラスターリストの各項目は、単一のデータベースクラスターたとえば、プライマリデータベースとそのレプリカ、またはシングルノードインスタンスを表します。
- **ノードエントリ**各クラスター内のノードリストは、そのクラスターに属する個々のデータベースインスタンスを指定します。

シングルインスタンスデータベースの場合、ノードリストには 1 つのエントリが含まれます。

マルチノードクラスタープライマリ/レプリカ設定のような場合、ノードリストには、すべてのインスタンスのエントリを含めることができます。

```
clusters:
  - resource_id: selfmanaged_postgres_1
    name: "<cluster_name1>"
    manager: "other"
    nodes:
      - resource_id: <node_resource_id_1.1>
        disable_host_association: false
        dsn: "$DSN1.1"
        ...
      - resource_id: <node_resource_id_1.2>
        disable_host_association: false
        dsn: "$DSN1.2"
        ...
  - resource_id: selfmanaged_postgres_2
    name: "<cluster_name2>"
    manager: "other"
    nodes:
      - resource_id: <node_resource_id_2.1>
        disable_host_association: true
        dsn: "$DSN2.1"
    ...
```

単一のエージェントインスタンスは、異なるホストマシンにまたがるデータベースを追跡できますが、その設定は、エージェントと同じマシン上にないデータベースに限定的なホストレベルのメトリックを提供します。

注釈

現在、監視データは DSN で指定されたデータベースからのみ収集されます。たとえば、 *postgres* データベースを指定した場合、データベースサイズはこのデータベースのサイズのみを反映します。同じ Postgres インスタンス内の他のデータベースをモニタするには、複数のインスタンスを指定するのと同じ方法でそれらを個別に指定する必要があります。

Controlling schema ingestion

beacon-agent.yaml により、エージェントは、移行ユーザーが権限を持ち、少なくとも 1 つのテーブルを含むすべてのスキーマを取り込むことができます。エージェントにすべてのスキーマを取り込ませたくない場合は、`databases.schema.filter` オプションを使用してサブセットを定義できます。

スキーマフィルタを使用する

エージェントがデータベースから取り込むスキーマを制御するには、databases.schema 構成ブロック内に filter パラメーターを追加します。これにより、含めるまたは除外するスキーマのセットを指定できます。

Filter パラメーターには 2 つの設定が必要です。

- mode フィルタリング動作を定義します。 include または exclude のいずれかに設定する必要があります。 include に設定されている場合、エージェントは名前リストにリストされているスキーマのみを取り込みます。exclude に設定されている場合、エージェントは名前リストにリストされているものを除くすべてのスキーマを取り込みます。
- names フィルターを適用するスキーマ名のリスト。名前は大文字と小文字が区別されるため、スキーマがソースデータベースに保存される方法と同じ大文字と小文字を名前に使用してください。

!!!note エージェントには、Oracle と Postgres の両方のデフォルトの 除外されたシステムスキーマ があります。これらは、データベースの移行とは無関係であるため、取り込みから自動的に除外されます。exclude フィルターを使用する場合、このデフォルトリストは指定した追加スキーマと組み合わせられます。include フィルタの場合、エージェントは明示的に名前付けたスキーマを取り込みます。デフォルトで除外されるシステムスキーマを含みます。

インクルードの例

```
[...]
provider:
  onprem:
    clusters:
      - resource_id: "<cluster_resource_id>"
        [...]
        nodes:
          - resource_id: "<node_resource_id>"
            dsn: $DSN
            schema:
              enabled: true
              poll_interval: 15s
              filter:
                mode: "include"
                names:
                  - "<schema_name1>"
                  - "<schema_name2>"
            tags:
              - "<tag_names>"
```

Exclude の例

```
[...]
provider:
  onprem:
    clusters:
      - resource_id: "<cluster_resource_id>"
       [..]
       nodes:
         - resource_id: "<node_resource_id>"
           dsn: $DSN
           schema:
             enabled: true
             poll_interval: 15s
             filter:
               mode: "exclude"
               names:
                 - "<schema_name1>"
                 - "<schema_name2>"
           tags:
             - "<tag_names>"
```

除外されたシステムスキーマ

エージェントは、デフォルトで次のシステムスキーマを除外します。

Postgres および EDB Postgres Extended Server

- information_schema
- pg_catalog
- pg_toast

EDB Postgres Advanced Server

- sys

Oracle

- アノニマス
- APEX_PUBLIC_USER
- APEX_030200
- APEX_040000
- APEX_040200
- APPQOSYS
- AUDSYS

- CTXSYS
- DMSYS
- DBSNMP
- DBSFWUSER
- デモ
- DIP
- DVF
- DVSYS
- EXFSYS
- FLOWS_FILES
- FLOWS_020100
- フランク
- GGSYS
- GSMADMIN_INTERNAL
- GSMCATUSER
- GSMROOTUSER
- GSMUSER
- LBACSYS
- MDDATA
- MDSYS
- MGMT_VIEW
- OJVMSYS
- OLAPSYS
- ORDPLUGINS
- ORDSYS
- ORDDATA
- OUTLN
- ORACLE_OCM
- OWBSYS
- OWBYSS_AUDIT
- PDBADMIN

- RMAN
- REMOTE_SCHEDULER_AGENT
- SI_INFORMTN_SCHEMA
- SPATIAL_CSW_ADMIN_USR
- SPATIAL_WFS_ADMIN_USR
- SQLTXADMIN
- SQLTXPLAIN
- SYS
- SYSBACKUP
- SYSDG
- SYSKM
- SYSRAC
- システム
- SYSMAN
- TSMSYS
- WKPROXY
- WKSYS
- WK_TEST
- WMSYS
- XDB
- XS$NULL

Adding usage reporting for Hybrid Manager-monitored databases

すべて統合エージェント展開から製品の使用状況に関する貴重な洞察を EDB に提供することにより、モニタリング活動を補完できます。使用レポートが有効になっている場合、エージェントは基本的な使用データを収集して、EDB が顧客をより深く理解できるようにします。有効にした場合にのみ EDB にデータを送信します。

!!!info ハイブリッドマネージャー HM を使用していませんか？ HM を使用しておらず、スタンドアロンデータベースの使用状況レポートを構成する必要がある場合は、Enabling usage reporting for stand-alone databases を参照してください。

インターネットに接続された環境の場合

1. agent configuration example では、 `beacon_agent.yaml`

   ファイルに既に `general.metrics.usage.output.file.enabled` が `true` に設定されています。この設定により、エージェントは使用データのローカルコピーをファイル `beacon_usage.json` に保存できます。

2. このファイルを確認して、情報が正確で、EDB と共有したくないデータが含まれていないことを確認します。

3. `general.metrics.usage.output.http.enabled` を `true` に設定して、EDB へのレポートを有効にします。

エアギャップ環境

1. agent configuration example では、 `beacon_agent.yaml`

   ファイルに既に `general.metrics.usage.output.file.enabled` が `true` に設定されています。この設定により、エージェントは使用データのローカルコピーをファイル `beacon_usage.json` に保存できます。

2. このファイルを確認して、情報が正確で、EDB と共有したくないデータが含まれていないことを確認します。

3. `beacon_usage.json` レポートファイルを、インターネットにアクセスできるマシンにコピーします。

Running the agent

エージェントを実行します。

```
beacon-agent
```

!!!tip 他のエージェントモードとオプションについては、Agent CLI を参照してください。

エージェントはハイブリッドマネージャー HM へのデータの送信を開始します。ログに従って、データの取り込みプロセスを監視します。数分後、エージェントからのデータが HM コンソールに表示されます。

- **Estate** セクションの **Postgres** タブには、Postgres データベースと要約されたデータベース情報がリストされます。いずれかのデータベースを選択すると、より詳細なデータベースの概要と監視機能が提供されます。
- **Estate > Migrations** タブには、外部データベースたとえば Oracle と、要約されたデータベースと移行情報がリストされます。いずれかのデータベースを選択すると、より詳細な移行評価と移行機能が提供されます。

!!!重要

エージェントは、実行中にテーブルと列のスキーマ変更を検出します。ただし、ビューまたはプロシージャへの変更は、エージェントを再起動するまで検出されません。

本番使用の場合、エージェントを run as a service に構成します。

Running the agent as a service

継続的なメトリックストリーミングのために、データベースがハイブリッドマネージャー HM エステートで可視されたままであるように保証には、サービスとして実行するようにエージェントを構成します。サービスとして実行すると、システムの起動時に自動的に起動し、エラーが発生した後に再起動します。

エージェントをサービスとして設定しない場合、手動で実行した場合にのみメトリックがストリーミングされます。さらに、データベースは、エージェントがストリーミングを停止してから 72 時間後に HM 不動産から消えます。

サービスとして設定する前に、エージェントのインストールと構成を完了します。

!!!note

エージェントパッケージの将来のバージョンでは、これが自動的に設定される可能性があります。

この例では、RHEL 9 マシンで実行されるようにエージェントを構成します。ただし、値を置き換える時期がわかるように、オペレーティングシステム支援を提供します。

1. 選択したテキストエディタを使用してファイルを作成します。このコマンド例では、vi エディターを使用します。

```
sudo vi /usr/lib/systemd/system/beacon-agent.service
```

AWS EC2 インスタンスの他のデフォルトのシステムディレクトリは次のとおりです。

Operating system	Default system directory
RHEL	/usr/lib/systemd/system/beacon-agent.service
Rocky Linux	/usr/lib/systemd/system/beacon-agent.service
AlmaLinux	/usr/lib/systemd/system/beacon-agent.service
Ubuntu	/etc/systemd/system/beacon-agent.service
Debian	/lib/systemd/system/beacon-agent.service

1. 次のテンプレートを beacon-agent.service ファイルに貼り付けて、プレースホルダー < > の情報を置き換えます。

```
# TEMPLATE

[Unit]
Description=Agent

After=network.target

[Service]
Type=simple

User=<user>

WorkingDirectory=</home/user/>
```

(次のページに続く)

(前のページからの続き)

```
ExecStart=</usr/local/bin/beacon-agent>

Restart=on-failure
RestartSec=60

[Install]
WantedBy=multi-user.target
```

Service settings	Placeholder to replace	Guidelines
User=	*<user>*	Provide the name of the system user that will run the agent.
WorkingDirectory=	*</home/user/>*	Set the working directory to where you stored the *.beacon* directory. The path depends on the operating system.
ExecStart=	*</usr/local/bin/beacon-agent>*	Provide the path to the directory where the agent was installed. The path depends on the operating system.

1. 次のコマンドを実行して、新しいサービスをリロード、有効化、および開始します。これらのコマンドは、前述のすべてのオペレーティングシステムに適用されます。

```
sudo systemctl daemon-reload
sudo systemctl enable beacon-agent.service
sudo systemctl start beacon-agent.service
```

1. エージェントはサービスとして実行されています。ログを確認します。

```
journalctl -u beacon-agent.service
```

Troubleshooting

エージェントの構成に問題がある場合、または実行しようとしたときにエラーが発生した場合は、トラブルシューティングを行うことができます。隔離された方法でエージェントの機能をテストして、問題の原因を絞り込みます。

エージェントからハイブリッドマネージャーへの接続のテスト

エージェントとハイブリッドマネージャー HM の間の接続があることを確認するには、エージェントを実行します。

```
beacon-agent --config
```

他のエージェントモードとオプションについては、Agent CLI を参照してください。

出力にエラーが表示されない場合、HM への接続は正常に機能しています。問題は、データベースへの接続に関係している可能性があります。

出力にエラーが表示される場合、HM への接続は正常に機能していません。YAML ファイルの構文、アクセスキー、CA パスとコンテンツ、DSN 文字列、およびプロジェクト ID を次々に確認して、設定が正しいことを確認します。

エラーメッセージと解決策

`"fail fetching server version:  pq:  SSL is not enabled on the server"`

接続しているデータベースの SSL が無効になっている場合、暗号化されていない接続を使用するようにエージェントに明示的に指示する必要があります。これを行うには、設定`?sslmode=disable` を Agent's configuration で使用される DSN に追加します。例

```
export DSN="postgresql://postgres:password@localhost:5432/postgres?sslmode=disable"
```

Administrative tasks

エージェントを構成するには、ユーザーが自分自身では取得できない一部のリソースにアクセスする必要があります。ただし、管理者は、Hybrid ManagerHM のインストール中またはその後に証明書とホスト関連の設定を構成する権利と権限を持っているため、これらのリソースを利用できます。

`beacon_server` とは何ですか？

エージェントを構成するには、ユーザーが `beacon_agent.yaml` ファイルで `beacon_server` 値を設定する必要があります。これは、ホストとポートの 2 つの値で構成される文字列です。

ホストは、インストールまたは `values.yaml` に使用される Helm チャート構成ファイルで提供されている URL です。`upm-beacon.server_host` または `BEACON_SERVICE_DOMAIN_NAME` に設定された値を検索して、URL を取得します。ポートは 9443 に固定されています。

ルート認証局 CA とは何ですか?

エージェントは、TLS を使用して `beacon_server` URL への安全な接続を確立します。したがって、エージェントが実行されているマシンは、`beacon_server` が提供するサーバー証明書を信頼する必要があります。通常、この信頼は組織の既存のセキュリティインフラストラクチャによって既に管理されています。内部サーバー証明書の署名を担当する認証局 CA は、ネットワーク全体で信頼できる CA として構成する必要があります。これにより、エージェントを実行しているサーバーを含む環境内のサーバーが、この CA が発行した証明書を本質的に信頼することが保証されます。

これに当てはまらない場合、ユーザーは `beacon_agent.yaml` ファイルで `root_ca_path` 値を設定できます。これは、ユーザーが EDB Postgres AI Platform サーバー証明書を検証できるルート証明書を保存したディレクトリです。ただし、非管理ユーザーは証明書にアクセスできないため、それらに証明書を提供する必要があります。

ルート証明書を含む Kubernetes シークレットを表示するには

```
#  The certificate is base64 encoded as the data field "public.crt"

kubectl get secret -n upm-beacon beaconator-ca-bundle -o json
```

ルート証明書のみを印刷する読みやすい出力については、 jq および base64 をインストールして実行していることを確認します。

```
#  If `jq` and `base64` are available

kubectl get secret -n upm-beacon beaconator-ca-bundle -o json | jq -r .data."public.
↪crt" | base64 --decode
```

このコマンドは、証明書の内容を出力します。安全なチャネルを介してそれらを必要とするユーザーに送信します。ユーザーは crt 形式でローカルに保存する必要があります。

Data security and sovereignty

ハイブリッドマネージャー HM で外部データベースをモニタするエージェントを展開すると、セキュリティとコンプライアンス要件に合わせて、モニタリングと評価のために収集されたデータの制御を維持します。

限定的なデータ収集

エージェントは次のデータを収集します。

- **Estate** ビューにメトリックを表示するための一般的なデータベースメタデータ。

Postgres データベースの場合

- *HM* **コンソールと可観測性** および **チャットモデルクラスターメトリックが** *Grafana* **モデル概要ダッシュボードにありません** を介して可観測機能を提供するデータベースログとメトリック。

Oracle データベースの場合

- 移行の複雑さと作業レベルの自動評価を生成するデータベースデータは、[**移行**] タブの **エステート** セクションに表示されます。
- スキーマ変換にそのツールを使用する場合、移行ポータルによる最終的な取得と使用のためのスキーマ DDL 。

!!!note

現在、移行評価データとスキーマ DDL データは、Oracle データベースでのみ収集されます。新しいリリースでは、外部の Postgres およびクラウドサービスプロバイダー CSP データベースの移行評価とスキーマ DDL データも収集します。

当社は、これらの目的に厳密に必要な範囲を超えてデータを収集することはありません。

データの保存場所

収集されたデータは、選択した環境内に残ります。HM はお客様のデータ主権を尊重します。たとえば、HM を独自のプレミスに展開した場合、監視対象データベースから収集されたデータはすべてオンプレミスインフラストラクチャ内に残ります。HM インスタンスが EKS のようなクラウドプラットフォームに展開されている場合、収集されたデータは特定のクラウド環境に安全に存在します。

Monitoring cloud service provider-managed Postgres deployments

完全な不動産表現サポートを備えたクラウドサービスプロバイダー CSP によって管理される環境で、ハイブリッドマネージャー HM の外部で実行されている PostgreSQL データベースを簡単に監視できます。

現在、次の外部 CSP をモニタできます。

- *Monitoring AWS RDS deployments*

Monitoring AWS RDS deployments

ハイブリッドマネージャー HM を使用して、アマゾンウェブサービス AWS リレーショナルデータベースサービス RDS 展開をモニタできます。

前提条件

HM が存在する Kubernetes 環境にアクセスし、HM クラスター全体の設定を変更するための十分な管理者権限が必要です。

このアクセス要件は、必要な特権がより少ない Postgres の **自己管理クラスターのナレッジベースを照会できません** 展開など、HM を使用した他の外部データベースモニタリング状況とは異なります。

また、Kubernetes シークレット beacon-csp-credentials に保存されているユーザー資格情報に、監視対象の AWS RDS 環境に対する十分な権限があることを確認する必要があります。これらの権限がないと、システムは使用可能なすべてのメトリックを取得できません。

これを行うには、次の最小限の権限を持つ IAM ユーザーの AWS アクセスキーを取得する必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowInstanceAndLogDescriptions",
            "Effect": "Allow",
            "Action": [
                "rds:DescribeDBInstances",
                "rds:DescribeDBLogFiles"
            ],
            "Resource": [
                "arn:aws:rds: *:* :db:*"
            ]
        },
        {
            "Sid": "AllowMaintenanceDescriptions",
            "Effect": "Allow",
            "Action": [
                "rds:DescribePendingMaintenanceActions"
            ],
            "Resource": "*"
        },
```

(次のページに続く)

(前のページからの続き)

```
        {
            "Sid": "AllowGettingCloudWatchMetrics",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowRDSUsageDescriptions",
            "Effect": "Allow",
            "Action": [
                "rds:DescribeAccountAttributes"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowQuotaDescriptions",
            "Effect": "Allow",
            "Action": [
                "servicequotas:GetServiceQuota"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowInstanceTypeDescriptions",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeInstanceTypes"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowInstanceFilterByTags",
            "Effect": "Allow",
            "Action": [
                "tag:GetResources"
            ],
            "Resource": "*"
        }
    ]
}
```

HM での AWS RDS 展開をモニタするための HM エージェントの設定

HM ですべての AWS RDS 展開をモニタするようにエージェントを設定できます。

> **注釈**
>
> RDS リージョンを登録すると、システムはその RDS リージョン内にあるすべてのデータベースを追加します。これにより、複数のリソースが不動産に追加される場合があります。これらはすべて 1 つのプロジェクトの下にあるため、ユーザーとしてすべての RDS データベースにアクセスできるか、それらのいずれにもアクセスできません。

1. `beacon-csp-credentials` シークレットを追加します。

`beacon-csp-credentials` シークレットを HM Kubernetes クラスターの `upm-beacon` 名前空間に追加します。

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
name: beacon-csp-credentials
namespace: upm-beacon
stringData:
AWS_ACCESS_KEY_ID: replace_with_your_access_key_id
AWS_SECRET_ACCESS_KEY: replace_with_your_secret_access_key
AWS_REGION: example-us-east-1
BEACON_PROVIDER_AWS_CONSTANT_REGIONS: example-us-east-1
BEACON_PROVIDER_AWS_CONSTANT_PROJECT_ID: your_project_id
BEACON_AGENT_PROVIDERS: appliance,aws
BEACON_PROVIDER_AWS_MODE: constant
BEACON_PROVIDER_AWS_CONSTANT_SERVICES: rds
EOF
```

2. エージェントと外部メトリックレシーバーを再起動します。

```
kubectl -n upm-beacon rollout restart deployment upm-beacon-agent-k8s
kubectl -n upm-beacon rollout restart deployment upm-beacon-external-metrics-receiver
```

Estate ページに AWS RDS 展開が表示されます。データベースへの入力が開始されると、5 つのメトリックを表示できます。

- CPU
- ネットワークスループット
- 接続
- ディスク IOPS
- ディスクスループット

HM での AWS RDS モニタリングの無効化

AWS RDS 展開のモニタリングをオフにできます。

1. `beacon-csp-credentials` シークレットを削除します。

```
kubectl -n upm-beacon rollout restart deployment upm-beacon-agent-k8s
```

2. エージェントと外部メトリックレシーバーを再起動します。

```
kubectl -n upm-beacon rollout restart deployment upm-beacon-agent-k8s
kubectl -n upm-beacon rollout restart deployment upm-beacon-external-metrics-receiver
```

エステートで AWS RDS 展開は表示されなくなり、HM でモニタリングされなくなります。

Monitoring with the Hybrid Manager console

Hybrid ManagerHM コンソールを使用して、データベースメトリックの包括的なセットにアクセスできます。可用性は、メトリックとクラスタータイプによって異なります。

- *Support scope* クラスターでサポートされているメトリックの種類について説明します。
- *Metrics latency* メトリックが更新される頻度について学びます。

Metrics support

Hybrid ManagerHM は、幅広いメトリックを収集および表示して、データベースクラスターを包括的な可視性を提供します。

HM コンソールに表示できるメトリックの可用性は、メトリックとクラスタータイプによって異なります。

概要の概要

この概要には、データベースクラスタータイプごとに使用可能なメトリックが示されています。

	ホストメトリック	*Postgres* メトリック	チャットモデルクラスターメトリックが *Grafana* モデル概要ダッシュボードにありません	クエリメトリック	*GPU Recommendations for Default NIM Models*	メトリックとログへのアクセス
HM-managed cluster	Yes	Yes	Yes	Yes	Yes	Yes
External: **Self-managed cluster**	Yes	Yes	No	Yes	Yes	Yes

次のページに続く

表 23 – 前のページからの続き

	ホストメトリック	*Postgres* メトリック	チャットモデルクラスターメトリックが *Grafana* モデル概要ダッシュボードにありません	クエリメトリック	*GPU Recommendations for Default NIM Models*	メトリックとログへのアクセス
External: **CNPG database cluster**	Yes	Yes	Yes	Yes	Yes	Yes
External: **AWS RDS database**	Yes	partial support	No	No	No	No

!!!note 制限-現在、ハイブリッドマネージャーは EDB CloudNativePG クラスターのモニタリングをサポートしています。オープンソースの CloudNativePG クラスターのモニタリングは、将来のリリースでサポートされる予定です。- モニタリングは、外部 EDB Postgres 分散クラスターではサポートされていません。

詳細な概要

ホストメトリック

ホストメトリックには、データベースインスタンスをホストする基になるサーバーインフラストラクチャのリソース使用率が表示されます。それらは、**Estate** ビューと、クラスタービューの **Host** セクションの **Monitoring** タブに表示されます。

Metric	HM-managed cluster	External: **Self-managed cluster**	External: **CNPG database cluster**	External: **AWS RDS database**
Memory	Yes	Yes	Yes	Yes
CPU	Yes	Yes	Yes	Yes
Storage	Yes	Yes	Yes	Yes
Disk IOPS	Yes	Yes	Yes	Yes
Disk throughput	Yes	Yes	Yes	Yes
Network activity	Yes	Yes	Yes	Yes

!!!重要

メモリ、CPU、およびストレージメトリックは、ホスティングノード上の Postgres プロセスの平均使用率を示します。HM 管理対象クラスターでシステム全体の使用状況を監視するには、**チャットモデルクラスターメトリックが *Grafana* モデル概要ダッシュボードにありません** を使用します。システム全体の監視は、外部クラスターで現在サポートされていません。

Postgres メトリック

Postgres メトリックは、Postgres データベースサーバーに固有のパフォーマンスと操作統計を示します。それらは **Estate** ビューとクラスタービューの ***Monitoring*** タブに表示されます。

Metric	HM-managed cluster	External: **Self-managed cluster**	External: **CNPG database cluster**	External: **AWS RDS database**
Connections				
Connections	Yes	Yes	Yes	Yes
AAS by Wait Type	Yes	Yes	Yes	No
Nr. of blocked backends	Yes	Yes	Yes	No
Transactions				
Tuples in	Yes	Yes	Yes	Yes
Tuples out	Yes	Yes	Yes	Yes
Transactions rate	Yes	Yes	Yes	Yes
Buffer cache hit ration	Yes	Yes	Yes	Yes
Longest running transaction	Yes	Yes	Yes	No
Queries				
Query latency	Yes	Yes	Yes	No
Query rate	Yes	Yes	Yes	No
Storage				
Database size	Yes	Yes	Yes	No
Disk usage	Yes	Yes	Yes	No
WAL size	Yes	Yes	Yes	Yes
WAL usage	Yes	Yes	Yes	No
Live/Dead tuples	Yes	Yes	Yes	No
Index/table/temp estimated size	Yes	Yes	Yes	Yes
Internals				
Time since last autovacuum	Yes	Yes	Yes	Yes
Autovacuum Stats	Yes	Yes	Yes	Yes
Time since last checkpoint	Yes	Yes	Yes	Yes
Checkpoint stats	Yes	Yes	Yes	Yes
Time since last successful WAL archive	Yes	Yes	Yes	No
WAL archiving stats	Yes	Yes	Yes	Yes

クラスターメトリック

クラスターメトリックは、ホストおよび postgres メトリックであり、マルチプルのノード HA、AHA、および DHA トポロジにまたがるデータベース展開用に個別および集計されたものです。それらは、**Estate** ビューと、クラスタービューの ***Monitoring*** タブの **Topology** および **Replication** セクションに表示されます。クラスターメトリックは、単一ノード展開の場合には表示されません。

Metric	HM-managed cluster	External: **Self-managed cluster**	External: **CNPG database cluster**	External: **AWS RDS database**
RAFT status (AHA, and DHA)	Yes	No	No	No
Replication slot status	Yes	No	Yes	No
Clock skew	Yes	No	Yes	No
Node status	Yes	No	Yes	No
Transaction rate	Yes	No	Yes	No
Connections (total)	Yes	No	Yes	No
Connections (per node)	Yes	No	Yes	No
Memory (per node)	Yes	No	Yes	No
Storage (per node)	Yes	No	Yes	No
WAL size (per node)	Yes	No	Yes	No
Replication status	Yes	No	Yes	No

クエリメトリック

クエリメトリックは、実行されているデータベース Postgres クエリのパフォーマンスと特性に関する洞察を提供します。それらは、クラスタービューの ***Query diagnostics*** タブで確認できます。

Metric	HM-managed cluster	External: **Self-managed cluster**	External: **CNPG database cluster**	External: **AWS RDS database**
Query latency	Yes	Yes	Yes	No
Nr. of queries executed	Yes	Yes	Yes	No
Query latency histogram	Yes	Yes	Yes	No
Nr. of active sessions	Yes	Yes	Yes	No

推奨事項

レコメンデーションは、観察されたメトリックに基づいて、データベース環境を改善するためのアクションと最適化を提案しますインデックスの改善など。それらは、クラスタービューの**推奨事項** タブで確認できます。

Metric	HM-managed cluster	External: **Self-managed cluster**	External: **CNPG database cluster**	External: **AWS RDS database**
Indexes	Yes	Yes	Yes	No
Statistics	Yes	Yes	Yes	No
Configuration	Yes	Yes	Yes	No
Security	No	No	No	No
Workload	No	No	No	No

ログ

Logs は、トラブルシューティングと監査のためにデータベースシステムのさまざまなコンポーネントからの生および処理されたログデータを表示します。それらは、クラスタービューの ***Logs*** タブで確認できます。

Metric	HM-managed cluster	External: **Self-managed cluster**	External: **CNPG database cluster**	External: **AWS RDS database**
Logs	Yes	Yes	Yes	No

Metrics latency

Hybrid ManagerHM は、広範囲の可観測性データを収集および表示し、データベースクラスターの包括的な可視性を提供します。

これらのメトリックの更新間隔は、メトリックとデータベースタイプによって異なります。この違いは、データの収集と統合のさまざまな方法によるものです。

データレイテンシーの理解

HM コンソールでデータベースメトリックを表示すると、表示されるデータは 2 段階のプロセスの結果であり、それぞれに独自の更新間隔があります。両方の手順を理解することは、メトリックがどの程度最新であるかを理解するのに役立ちます。

バックエンドの更新これは最初のステップであり、HM の内部サービスがデータベースクラスターから更新されたメトリックとデータをアクティブに収集します。クラスターが HM で管理されているか外部データベースで管理されているかにかかわらず、バックエンドシステムは常に最新の情報を収集しています。このデータは、HM の中央データストアに保存されます。

HM コンソールの更新これは 2 番目のステップであり、HM コンソールが中央のデータストアを呼び出して、最後に収集されたデータを取得して表示します。

HM コンソールに表示されるデータの実際のレイテンシーは、これら 2 つの更新間隔の組み合わせです。画面上のデータは、これらの 2 つの間隔の合計より古くない情報を反映しています。多くの場合、それははるかに新鮮であり、場合によってはわずか数秒前です。

リフレッシュ間隔

バックエンドの更新 HM の内部サービスは次のようにデータストアを更新します。間隔は、データベースクラスタータイプとメトリックコンテキストによって異なります。

	HM-managed	External: self-managed	External: CSP-managed	External: EDB CNPG
Estate page ☒ Postgres table	20s	5m	5m	5m
Cluster view ☒ Monitoring tab	15s	15s	30s	15s
Cluster view ☒ Query Diagnostics tab	1m	1m	na	1m
Cluster view ☒ Recommendations tab	30m	30m	na	30m
Cluster view ☒ Logs tab			na	

これらの間隔は、自己管理および CSP 管理のクラスターで構成できます。自己管理クラスターのデータ収集の頻度をカスタマイズするには、HM agent configuration YAML で間隔を定義します。 CSP 管理対象クラスターの場合、*beacon-csp-credentials* を介してサポートされているメトリックの間隔を定義します。

HM コンソールの更新 HM コンソールは、次の頻度ですべてのクラスタータイプのデータを更新します。

	Console refresh interval
Estate page ☒ Postgres table	1m
Cluster view ☒ Monitoring tab	30s
Cluster view ☒ Query Diagnostics tab	Manual browser refresh
Cluster view ☒ Recommendations tab	Manual browser refresh
Cluster view ☒ Logs tab	10s

!!!note

データは、ブラウザページを明示的にリロードした場合にのみ更新されます。

Grafana

Grafana を使用すると、ハイブリッドマネージャー HM 展開、HM 管理対象データベースクラスター、および HM に接続されているその他の外部クラスターに関するメトリックを視覚化および分析できます。パフォーマンスの監視、問題のトラブルシューティング、およびリソースの使用率に関する洞察を得ることができます。

HM 管理対象クラスターの場合、Grafana はすぐにメトリックを収集します。外部クラスターの場合、Monitoring external databases で説明しているように、最初に HM エージェントをインストールする必要があり

ます。

Grafana のサポート

Grafana ダッシュボードは次のようにサポートされています。

- HM 管理クラスター すぐに使える完全なサポート。
- 自己管理クラスタークラスターが HM エージェントを介して接続された後の部分的なサポート。
- CNP クラスタークラスターが HM エージェントを介して接続された後の部分的なサポート。
- RDS クラスタークラスターが Kubernetes シークレットを介して接続された後の部分的なサポート。

Grafana へのアクセス

Launchpad を介してビルトイン Grafana インスタンスにアクセスできます。

結果のフィルタリング

HM によって管理されるすべてのクラスターには ID があり、名前空間とそのポッドまで追跡できます。クラスター ID 名前空間に相当するまたはポッド名で Grafana のメトリックをフィルタリングおよび分析し、トラブルシューティングとリソースプランニング操作を実行できます。

Kubernetes インフラストラクチャがクラスターにどのように関連するかを理解するには、*Tracing clusters back to their Kubernetes resources* を参照してください。より具体的には

- *ID and namespace for a cluster* の取得方法を理解します。
- クラスター/名前空間で *Loki* **ポッドのクラッシュ** を認識する方法を理解します。

関連トピック

- *Using Grafana dashboards*
- Grafana's Explore option

Using Grafana dashboards

Hybrid ManagerHM に同梱されている Grafana サービスには、さまざまな HM リソースをモニタするように設計されたデフォルトのダッシュボードが付属しています。または、独自のカスタムダッシュボードを作成して、組織の要件を満たす特定のデータを表示します。

デフォルトの Grafana ダッシュボード

HM に同梱されている Grafana サービスには、次のカテゴリにグループ化されたデフォルトのダッシュボードが付属しています。

- *Kubernetes dashboards*
- *Loki dashboards for logging monitoring*
- *Postgres dashboards*
- *Telemetry dashboards*

- *Other dashboards*

!!!note

ダッシュボードの説明は、HM の Grafana に含まれるデフォルトのダッシュボードの一般化および例を示した概要を提供します。特定のメトリックの可用性は、環境によって異なります。

カスタムダッシュボード

Grafana のドキュメント create your own dashboard を参照してください。

!!!警告

カスタマイズされた Grafana ダッシュボードのバックアップがあることを確認し、バージョン管理リポジトリに保存します。カスタマイズされた Grafana ダッシュボードは、HM アップグレード、ノードの Kubernetes バージョンのアップグレード、または一般的なノードの障害など、Grafana インスタンスが更新または再展開されるたびに削除されます。

ダッシュボードの作成、使用、および管理

ダッシュボードの作成、使用、および管理の方法については、Grafana documentation を参照してください。ここにいくつかの役立つリンクを示します。

-

Create a dashboard 。

-

Import dashboards 。

-

Modify dashboard settings 。

- Manage dashboards 。

-

Share dashboards 。

- Use Explore データの照会、収集、および分析。
- Use Alerting 特定のイベントを監視およびフラグを立てるようにシステムを設定します。

Kubernetes dashboards

これらのダッシュボードは、ハイブリッドマネージャー HM を強化する基になる Kubernetes インフラストラクチャの状態とパフォーマンスに関する洞察を提供します。

これらは、Grafana の公式ライブラリからの Kubernetes mixins と呼ばれる事前に構築されたダッシュボードです。

Kubernetes / API サーバー

!!!note

Grafana は API サーバーへのノードレベルのアクセスを必要とするため、このダッシュボードには、オンプレミスの HM 展開のデータのみが表示されます。

説明クラスターとのすべての相互作用のための重要なコンポーネントである Kubernetes API サーバーのパフォーマンスと可用性を監視します。

メトリックこれらのメトリックには、読み取りおよび書き込みの可用性、ワークキューのレイテンシー、読み取りおよび書き込みエラー、リソース使用率などが含まれます。

使用法これを使用して、API レイテンシー問題または可用性問題を特定します。

Kubernetes /コンピューティングリソース

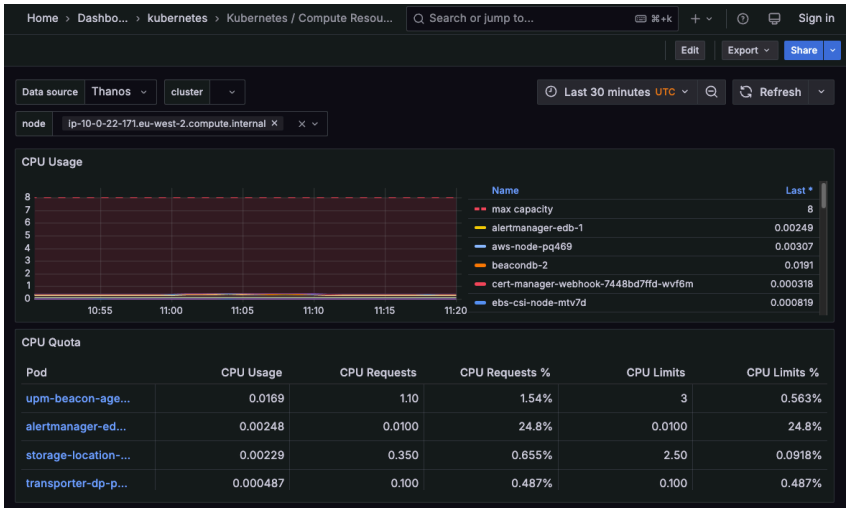


図 14: Compute Resources by Node and Pods

コンピューティングリソースは、さまざまなビューを提供する複数のダッシュボードに表示されます。**クラスター** 、 **名前空間ポッド** 、 **名前空間ワークロード** 、**ノードポッド** 、 **ポッド** 、または **ワークロード** 。

- **Cluster** には、Kubernetes クラスター全体のリソース使用率と、名前空間ごとの使用率が表示されます。
- **Namespace (Pods)** は、ポッド/サービスの消費に焦点を当てて、Kubernetes 名前空間でフィルタリングできるリソースの使用率を表示します。
- **Namespace (Workloads)** は、個々のワークロードの消費に焦点を当てて、Kubernetes 名前空間およびワークロードタイプ展開、デーモンセット、ステートフルセットなどによってフィルタリングできるリソース使用率を表示します。
- **Node (Pods)** は、個々の Kubernetes ノードでフィルタリングできるリソースの使用率を示し、それらで実行されているポッドごとに内訳。
- **Pod** は、特定の名前空間の個々の Kubernetes ポッドの詳細なリソース使用量を提供します。
- **Workload** は、特定の名前空間の特定の Kubernetes ワークロード展開ワークロードなどの詳細なリソース使用量を示します。

説明 Kubernetes クラスターレベルでのリソース使用率を表示します。各サブダッシュボードは異なる方法でデータを表示し、さまざまな Kubernetes レベルに焦点を当て、さまざまなフィルタリングオプションを提供します。

メトリックメトリックには、CPU、メモリ、帯域幅、パケットレート、IOPS、スループットなどが含まれます。

使用法 これらのダッシュボードは次の目的で使用します。

- すべてのワークロードのリソース消費を理解する
- リソースを大量に消費するアプリケーションまたはチームを特定する
- アプリケーション内の特定のワークロード、およびノード内の特定のポッドのリソースフットプリントを理解する
- 特定のアプリケーションインスタンスのパフォーマンス問題のトラブルシューティング
- 特定のアプリケーションコンポーネントのパフォーマンスを監視する

Kubernetes / Kubelet

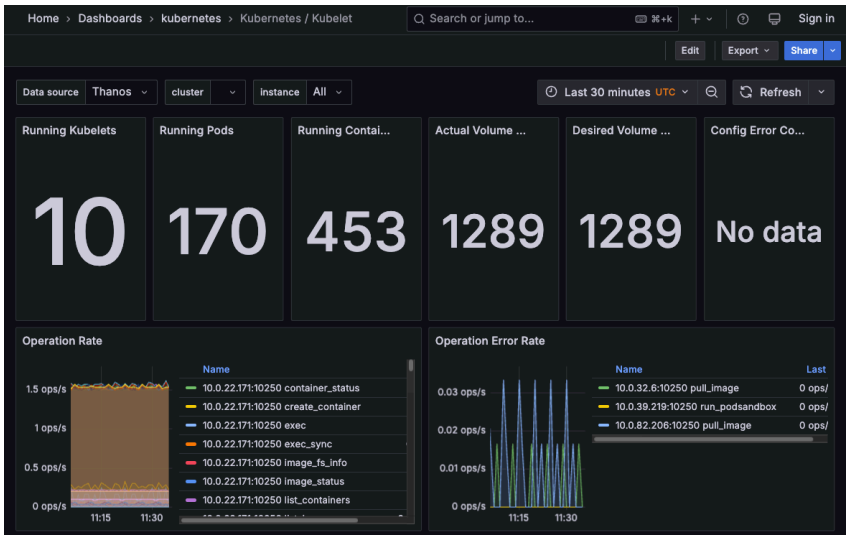


図 15: Kubelet dashboard

説明コンテナの管理を担当する、各 Kubernetes ノードで実行されている Kubelet サービスの状態とパフォーマンスを監視します。すべての kubelet の集計パフォーマンスを確認し、kubelet インスタンスでフィルタリングし、特定の kubelet 操作にドリルダウンできます。

メトリックメトリックには、kubelet 数、ポッド数、コンテナー数、操作率、操作期間、ポッドライフサイクル操作率、操作エラー率などが含まれます。

使用法これを使用して、ノードレベルのエージェントアクティビティ、操作動作、および遅延を特定し、特定のノードのパフォーマンス問題のトラブルシューティングを行います。

Kubernetes / ネットワーキング

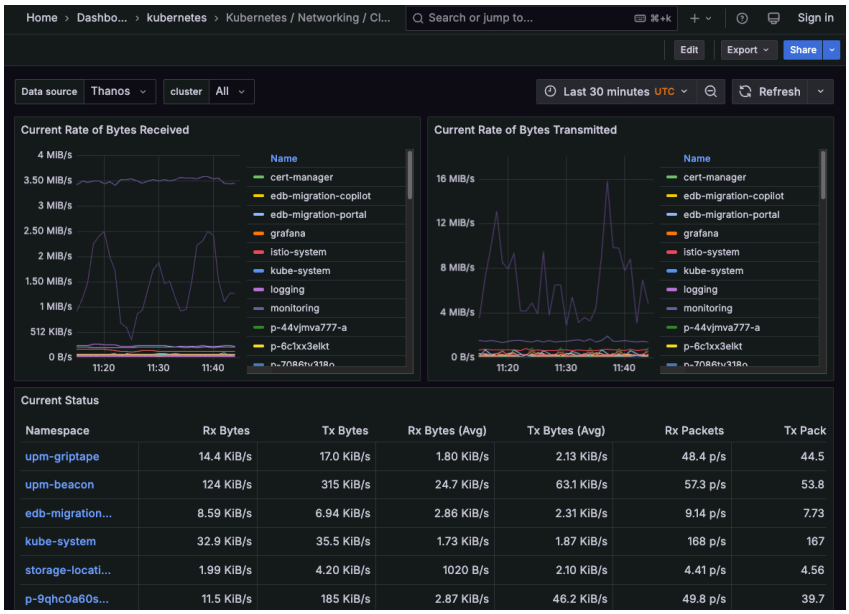


図 16: Networking overall cluster dashboard

ネットワークリソースは、さまざまなビューを提供する複数のダッシュボードに表示されます。

- **Cluster** には、Kubernetes クラスター全体および名前空間ごとのネットワーク使用率が表示されます。
- **Namespace (Pods)** は、Kubernetes 名前空間でフィルタリングできるネットワークトラフィックメトリックを、その名前空間のポッドに焦点を当てて表示します。
- **Namespace (Workloads)** は、個々のワークロードトラフィックに焦点を当てて、Kubernetes 名前空間およびワークロードタイプ展開、デーモンセット、ステートフルセットなどによってフィルタリングできるネットワークトラフィックメトリックを示します。
- **Pod** は、特定の名前空間内の個々の Kubernetes ポッドの詳細なネットワークトラフィックメトリックを提供します。
- **Workload** は、特定の Kubernetes ワークロード展開ワークロードなどの詳細なネットワークトラフィックメトリックを表示します。

説明 Kubernetes クラスターレベルでのネットワーク使用状況を表示します。各サブダッシュボードは異なる方法でデータを表示し、さまざまな Kubernetes レベルに焦点を当て、さまざまなフィルタリングオプションを提供します。

メトリックメトリックには、送受信されたバイト数、送受信されたパケットのレート、送受信帯域幅などが含まれます。

使用法 これらのダッシュボードは次の目的で使用します。

- 全体的なネットワーク通信パターン、トラフィック、スループット、および潜在的なボトルネックを理解する
- 特定のアプリケーションのネットワークパターンを分析する
- アプリケーション内の特定のワークロードのリソースフットプリントを理解する

- アプリケーション内の特定のポッドとのネットワーク接続問題のトラブルシューティング
- 特定のアプリケーションコンポーネントのネットワークパフォーマンスを監視する

Kubernetes / 永続ボリューム

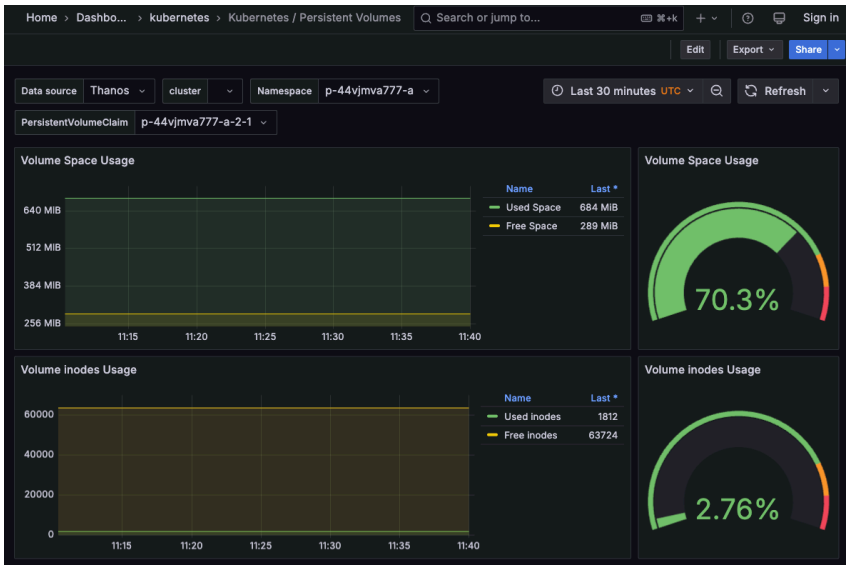


図 17: Persistent Volumes dashboard

説明 Kubernetes クラスター内の永続ボリューム PVs の使用状況とパフォーマンスを監視します。名前空間および永続ボリューム要求ごとに結果をフィルタリングできます

PVCs 。

メトリック メトリックには、空き領域と使用済み領域が含まれます。

使用法これを使用して、ストレージのボトルネックと容量の問題を特定し、ストレージ拡張の計画を立て、ストレージコストを監視します。

Kubernetes / StatefulSets

説明 ステートフルアプリケーションを管理する Kubernetes StatefulSets の特定のモニタリングを提供します。名前空間=アプリケーションとステートフルセット名で結果をフィルタリングできます。

メトリックメトリックには、CPU、メモリ、ネットワークの消費量、およびレプリカが含まれます。

使用法これを使用して、ステートフルアプリケーションの安定性、パフォーマンス、およびリソース消費をモニタし、ステートフルアプリケーションが目的のレプリカを満たしているかどうかを追跡します。

Kubernetes CoreDNS

説明 CoreDNS サービスのパフォーマンスと状態を監視します。これは、Kubernetes クラスターで DNS 解決を提供し、ポッドとサービス内の通信を有効にします。

メトリックメトリックには、プロトコル別の DNS 要求レート、クエリタイプ、DNS ゾーン、サイズ、応答コード、応答期間、応答サイズなどが含まれます。

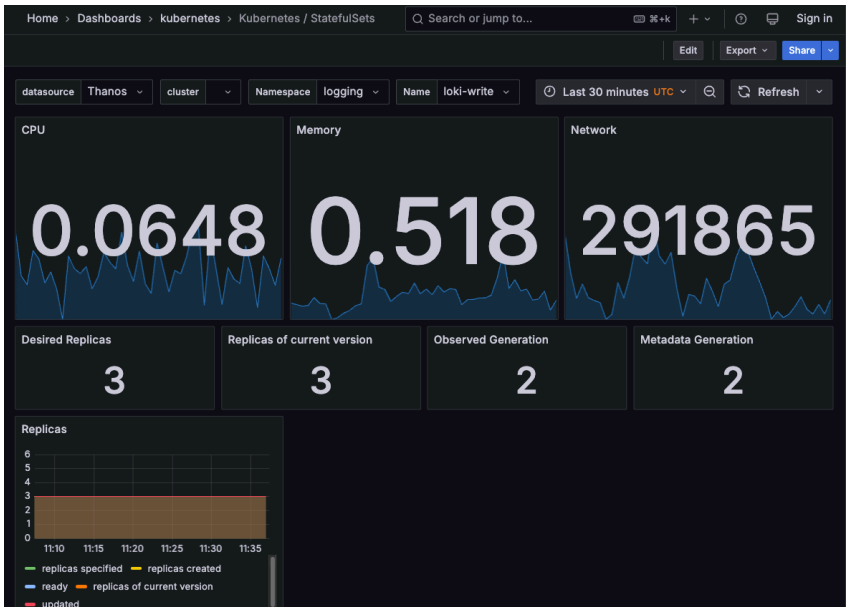


図 18: StatefulSets dashboard



図 19: CoreDNS dashboard

使用法これを使用して、CoreDNS コンポーネントの信頼性と効率をモニタし、サービス間の検出と通信の問題を特定します。

Loki dashboards for logging monitoring

これらのダッシュボードは、統合された Loki ログシステムに関連しており、ログを視覚化および分析できます。Loki が操作を実行する場所であるため、視覚化は `logging` 名前空間のみに制限されます。

Loki ダッシュボードは、Grafana の公式ライブラリからの Loki mixins と呼ばれる事前構築されたダッシュボードです。

ロキ / チャンク



図 20: Loki Chunks dashboard

説明 Loki のログデータチャンク単一のログストリームに属するログエントリのグループのストレージと管理を監視します。

メトリックメトリックには、アクティブなチャンク、パーセンタイル使用率、チャンクごとのログエントリ、チャンクライフサイクルイベントフラッシュ操作、チャンクサイズ、チャンク年齢、チャンクごとのバイトなどが含まれます。

使用法潜在的なストレージの非効率を特定し、チャンクのライフサイクルパターンを理解し、チャンク管理とクエリパフォーマンスに関連する問題のトラブルシューティングを支援します。

Loki / 削除

説明保持ポリシーに基づいて、Loki のログ削除プロセスのパフォーマンスとステータスを追跡します。デフォルトの Loki 保持期間は 30 日です。

メトリック削除期間、削除エラー、削除されたチャンク数、削除の影響を受けるシリーズの数、および削除キューの長さ。

使用法保持ポリシーが正しく適用されていることを確認し、削除プロセスの効率を監視し、ログの削除を妨げているエラーを特定できます。これを使用して、ログが予想通りにプルーニングされていることを確

認します。

Loki / Logs

説明 Loki によって取り込まれたログを表示およびフィルタリングできます。

メトリック ログ、取り込まれたログのレートなど。

使用法ログデータの視覚的な概要を提供し、傾向とエラーをすばやく識別できます。これは、リアルタイムログ分析とトラブルシューティングに使用します。

Loki / オペレーション



図 21: Loki Operational dashboard

説明 Loki システムの運用の状態とパフォーマンスに関する洞察を提供します。

メトリック 1 秒ごとのクエリーとプッシュ数、クエリーとプッシュのレイテンシー、テナントごとのログ行、テナントごとの MB 使用量、Loki コンポーネントの CPU 使用率インジェスター、ディストリビューター、クエリア、Loki コンポーネントのメモリ使用量、ストリーム、チャンク時間、その他。

使用法リソース消費と内部の状態を監視することにより、Loki ログシステムの安定性とパフォーマンスを保証します。Loki の潜在的なボトルネックまたはリソースの枯渇を特定します。

Loki / 読み取り

説明 Loki に対するログ読み取りクエリのパフォーマンスを監視します。

メトリック 1 秒ごとの Loki 読み取りクエリー、読み取りオペレーションごと、ポッドごとのレイテンシー。

使用法 ログクエリーに対する Loki の応答性を評価できます。

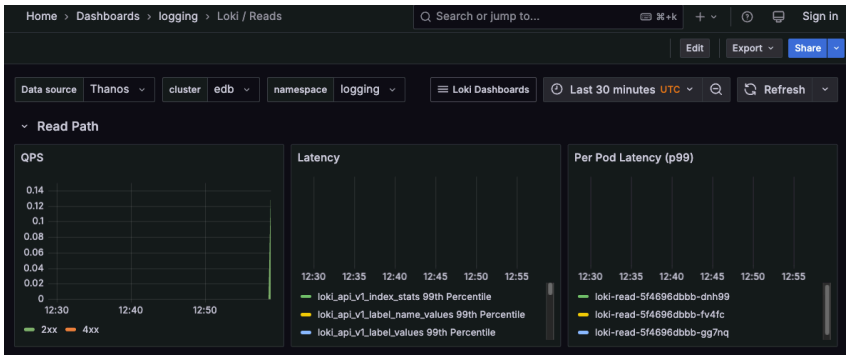


図 22: Loki Reads dashboard

Loki / Reads リソース

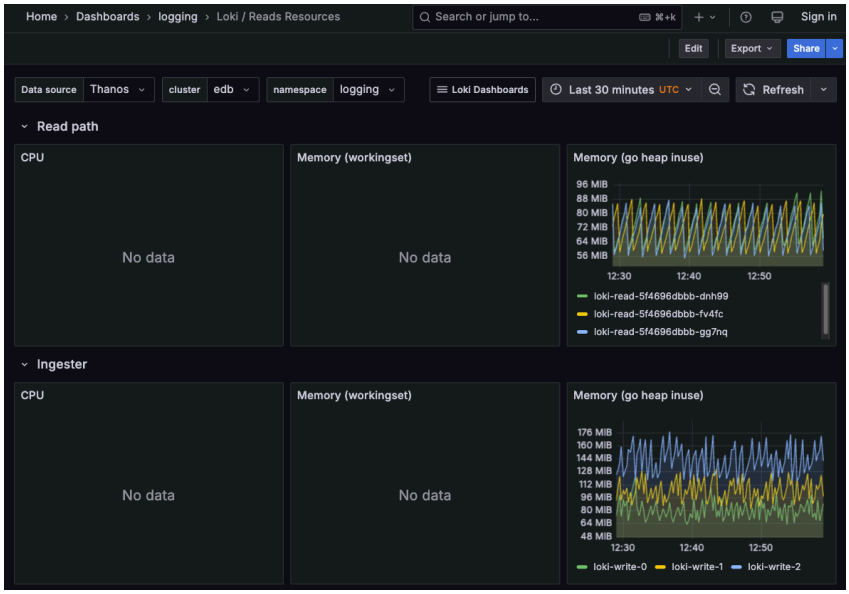


図 23: Loki Reads Resources dashboard

説明 Loki 読み取り操作によるリソース消費 CPU、メモリを表示します。

メトリック個々の Loki 読み取りポッドおよび Loki に取り込まれたポッドによって使用されているメモリ。

使用法ログ読み取りクエリのリソースオーバーヘッドを理解するのに役立ちます。

Loki / リテンション

説明 Loki のログ保持ポリシーとその適用に関する詳細なメトリックを提供します。

メトリック最後に成功した圧縮オペレーション、期間とステータス別の圧縮オペレーション、Loki バックエンドポッドが使用しているメモリ保持用。

使用法構成済みの保持ポリシーがアクティブであることを確認し、Loki のストレージ消費への影響を理解できます。

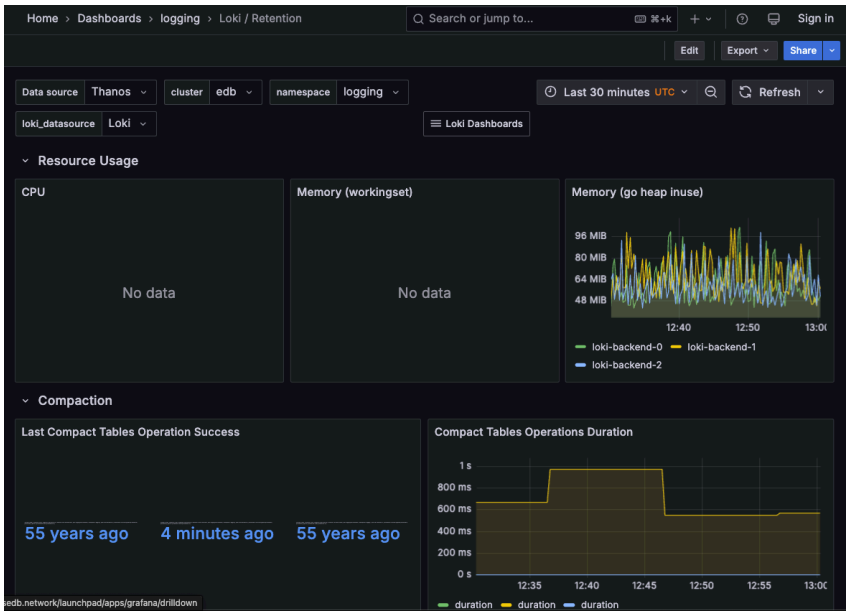


図 24: Loki Retention dashboard

Loki / Writes



図 25: Loki Writes dashboard

説明 Loki へのログ取り込み書き込みクエリのパフォーマンスを監視します。

メトリック 1 秒ごとの Loki 書き込みクエリー、書き込みオペレーションごと、ポッドごとのレイテンシー。

使用法遅い取り込み速度またはアプリケーションから Loki へのログ転送パイプラインでドロップされたログを特定するのに役立ちます。

Loki / Writes リソース

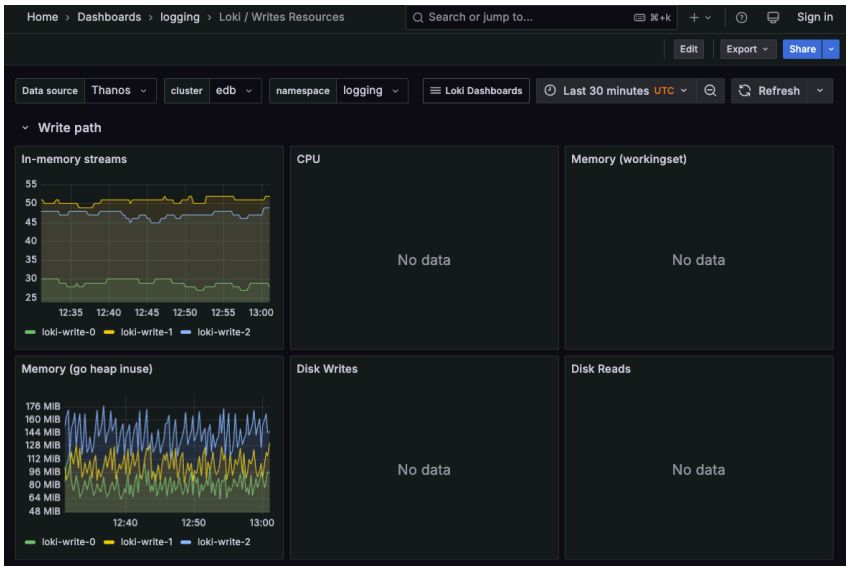


図 26: Loki Writes Resources dashboard

説明 Loki 書き込み操作によるリソース消費 CPU、メモリを表示します。

メトリック個々の Loki 書き込みポッドおよび Loki 取り込みポッドによって使用されているメモリ。

使用法ログ書き込みクエリのリソースオーバーヘッドを理解するのに役立ちます。

ロキデータ



図 27: Loki Data dashboard

説明これは、EDB によって設計されたダッシュボードで、Loki が管理しているログデータの量の高レベルのビューを提供します。

メトリック Loki の取り込みレートとバイト単位の合計保存チャンク。

使用法このダッシュボードは、キャパシティプランニング、データの増加の理解、およびロギング関連のストレージコストの管理に使用できます。

Postgres dashboards

これらのダッシュボードは、Hybrid ManagerHM で管理または監視する Postgres データベースのパフォーマンスと状態に関する洞察を提供するように EDB によって設計されました。

ハードウェア使用率

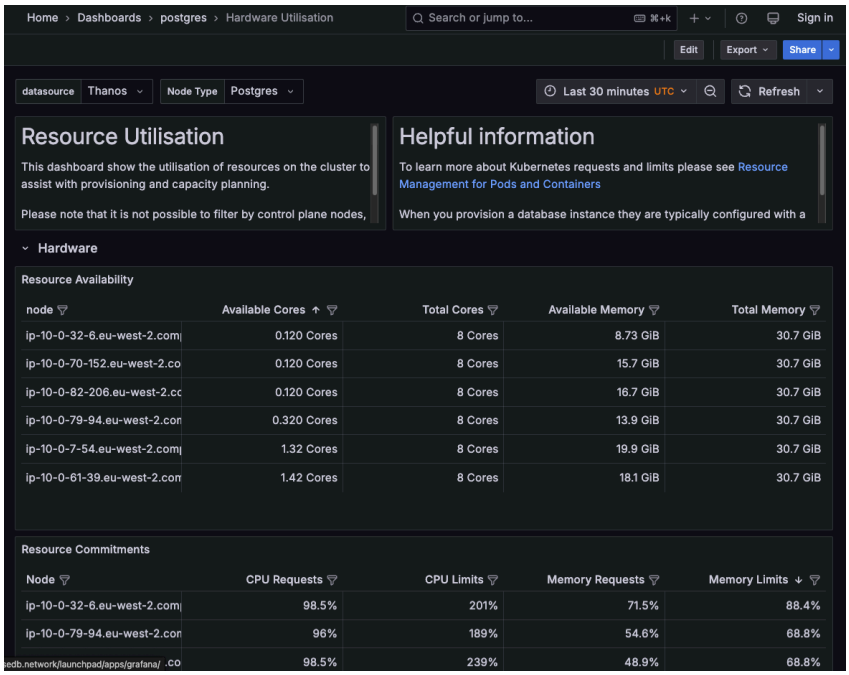


図 28: Postgres Hardware Utilisation dashboard

説明 Postgres インスタンスをホストするノードの基になるハードウェアリソース使用コア、メモリ、CPU を表示します。

メトリックノードごとの使用可能なコアと合計コア、ノードごとの使用可能なメモリと合計メモリ、ノードごとの CPU 要求と制限、ノードごとの Postgres ポッド数、ポッドステータスなど。

使用法これを使用して、キャパシティのボトルネックを特定し、オーバーサブスクライブしていないか、ノードがオーバースケジュールされているかどうかを確認します。

Kubernetes イベント

説明 Postgres のポッドと展開に関連する Kubernetes イベントを表示します。過去 1 時間のイベントが表示されます。したがって、ワークロードがスムーズに実行され、ポッドの復元に Kubernetes イベントが必要ない場合、指定された名前空間のイベントが表示されない可能性があります。

メトリック YAML 形式のイベントのタイプたとえば、ポッドの作成、削除、エラーなど。

使用法 Postgres インスタンスに影響を与える可能性のある Kubernetes 操作にコンテキストを提供します。失敗した展開、ポッドの再起動、またはデータベースの可用性に影響を与えるその他の Kubernetes レベル

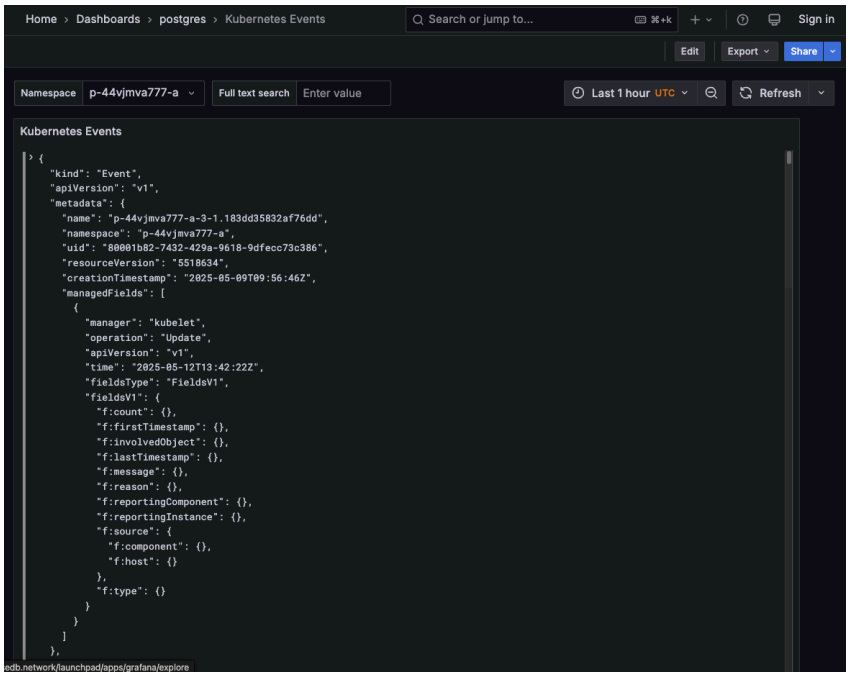


図 29: Postgres Kubernetes Events dashboard

の問題の診断に役立ちます。

PGD の概要

分散高可用性 DHA または PGD クラスターのみ。

説明レプリケーション機能に焦点を当てて、分散高可用性 PGD クラスターのパフォーマンスの概要を提供します。

メトリック PGD および BDR のバージョン、データノードのステータス、バージョン、Raft の状態とメトリック、BDR ワーカーの状態とメトリック、レプリケーションステータス、接続、スループットなど。

使用法クラスターの状態と同期ステータスを一元的にビューできます。遅延または異常なノードを特定し、分散セットアップの全体的なパフォーマンスを監視します。

Postgres エステート統計

説明 実行しているクラスターのフリート全体のビューを提供します。

メトリック実行中のすべてのクラスターのリストと数、その経過期間、および最新の復元スタンプ。

使用法 エステートの下のクラスターの高レベルの概要を提供します。

Postgres ログ

説明: Postgres サーバーログを表示できます。

メトリック テキスト形式のログ。

使用法 Postgres ログデータの視覚的な概要を提供し、傾向とエラーをすばやく識別できるようにします。これは、リアルタイムログ分析とトラブルシューティングに使用します。

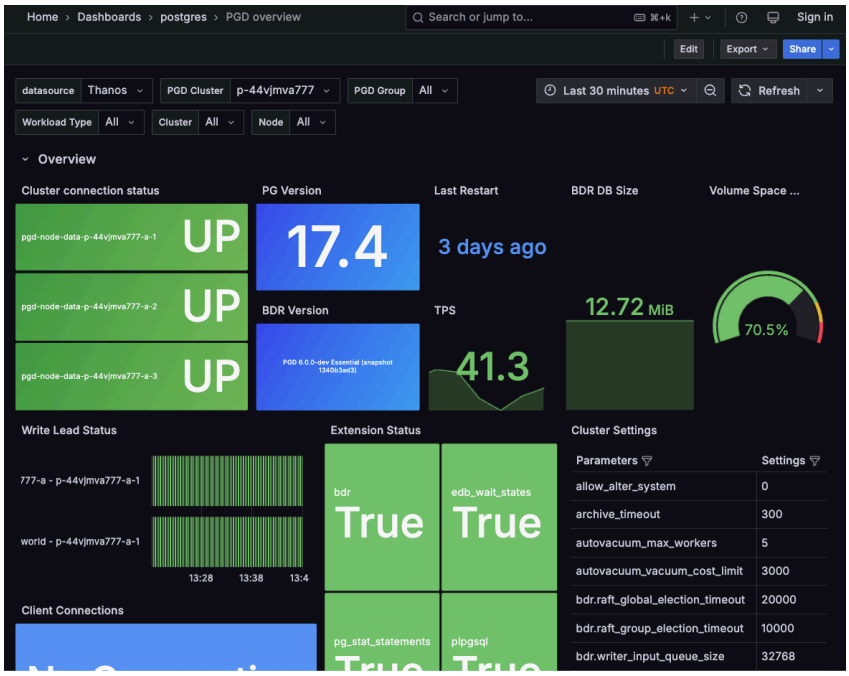


図 30: Postgres PGD Overview dashboard

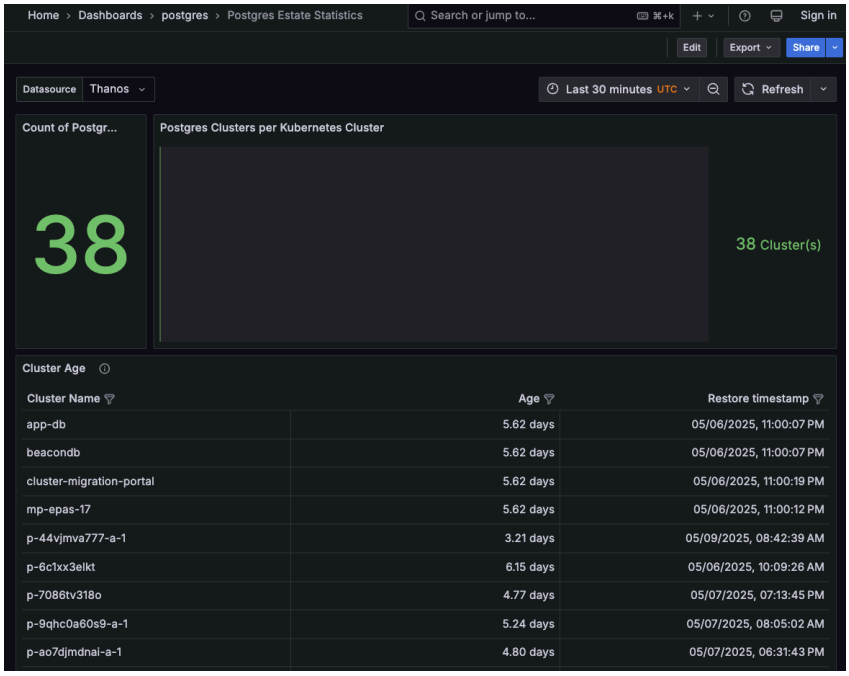


図 31: Postgres Estate Statistics dashboard

Postgres ステータス

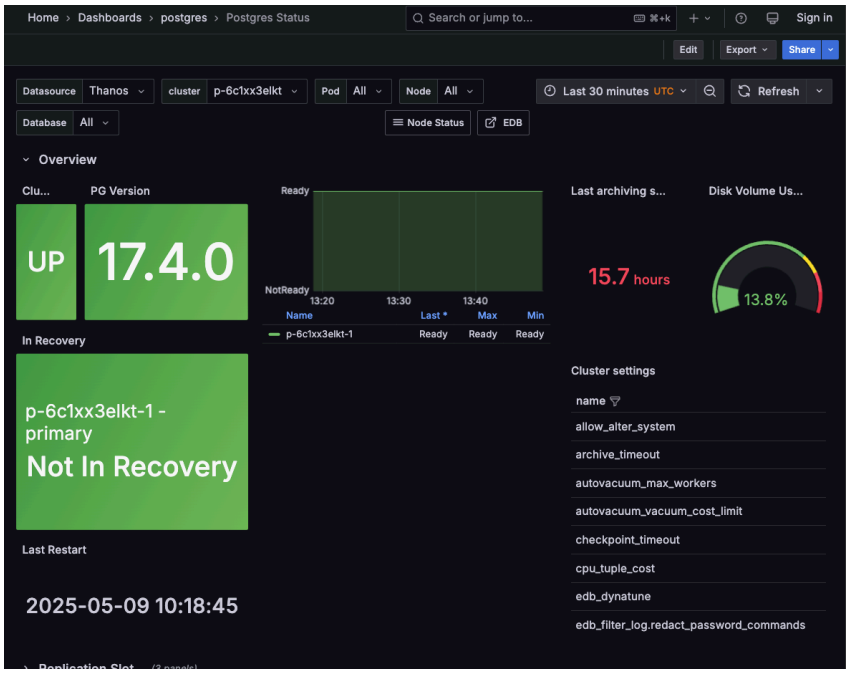


図 32: Postgres Status dashboard

すべてのクラスタータイプシングルノードおよびマルチプルノードの場合。

説明 個々の Postgres インスタンスのパフォーマンスの概要を提供します。

メトリッククラスターステータス、Postgres バージョン、ポッドステータス、レプリケーションステータスと遅延、接続、スループット、リソース使用率、WAL ファイリングステータスなど。

使用法個々の Postgres データベースの状態とアクティビティの一元的なビューを提供します。高負荷、接続の問題、レプリケーションラグ、または調査が必要なその他のパフォーマンス問題があるインスタンスを特定します。

Telemetry dashboards

これらのダッシュボードは、Hybrid ManagerHM プラットフォームとその基になるコンポーネントに関する洞察を提供します。

FluentBit

これは、Grafana の公式ライブラリからの事前ビルドされた Fluent Bit mixin です。

説明ノードからログを収集し、Loki に送信するために使用される FluentBit インスタンスに関するメトリックを提供し、Loki は Hybrid Manager に通知します。

メトリック入力および出力処理レート、入力および出力レコードレート、失敗およびエラー率など。

使用法 FluentBit でパフォーマンスのボトルネック、リソースの枯渇、または HM コンポーネントからログの収集と送信中にエラーが発生しているかどうかを特定するのに役立ちます。

Grafana の概要

説明さまざまなソースからメトリックを収集し、一箇所に表示するプラットフォームである Grafana インスタンスに関するメトリックを提供します。

メトリック アラートの発生、1 秒ごとの要求、レイテンシーなど。

使用法これらのダッシュボードの表示に使用している Grafana インスタンスの状態とパフォーマンスをモニタできます。Grafana が過負荷になっているか、問題が発生しているかどうかを特定します。

Memcached

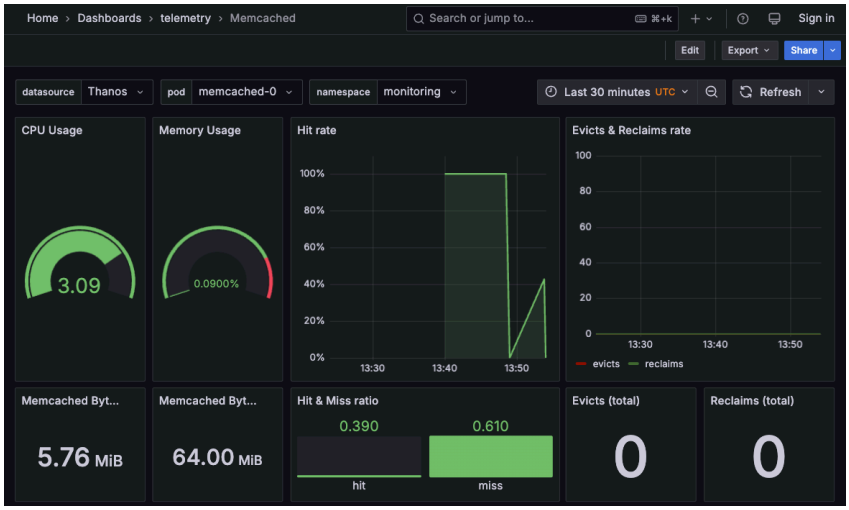


図 33: Telemetry Memcached dashboard

説明 HM インフラストラクチャの Memcached モニタリングサービスの状態とパフォーマンスを監視し、リソースの使用率、キャッシュ効率、およびエビクション率に関する洞察を提供します。

メトリック CPU とメモリの使用量、ヒット/ミス率、削除と再利用率、memcached バイト数など。

使用法このダッシュボードを使用して、Memcached インスタンスのリソース消費を理解し、キャッシュの有効性ヒット率を評価し、メモリプレッシャーのために項目がエビクトまたは再利用される頻度をモニタします。

ノードエクスポーター

これらは、Grafana の公式ライブラリから事前にビルドされた node mixins です。

ノードエクスポーターメトリックは、さまざまなビュー **ノード** 、 **USE Method / Cluster** 、および **USE Method / Node** を提供する複数のダッシュボードに表示されます。

- **ノード**は、特定のノードの集計されたリソース使用率を提供します。CPU 使用率、負荷、メモリの種類などに分類されます。
- **USE Method / Cluster** は、Kubernetes クラスター全体の集計されたリソース使用率をノードごとに色付けで表示します。
- **USE Method / Node** は、ノードごとの合計リソース使用率と、不揮発性メモリデバイスのいくつかのメトリックを表示します。

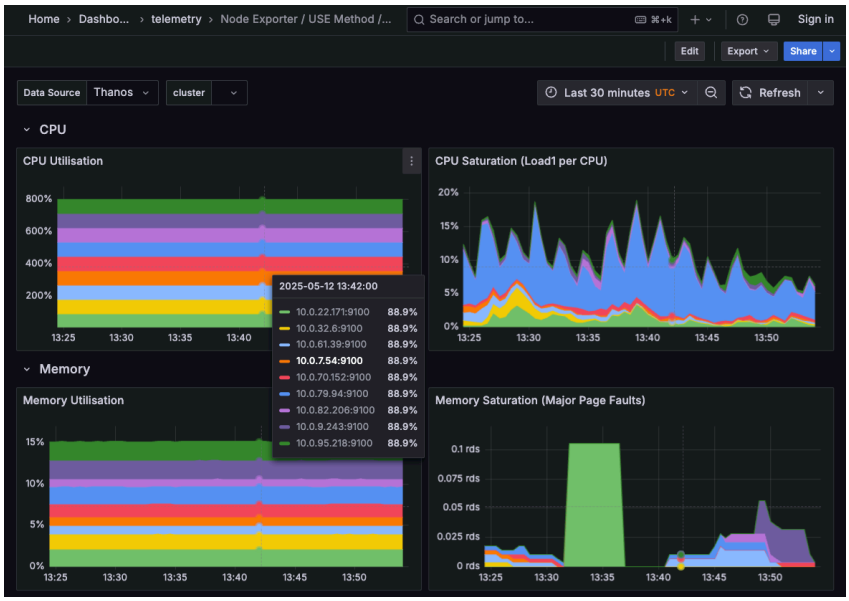


図 34: Telemetry Node Exporter dashboards

説明 HM コンポーネントを実行しているノードのハードウェアメトリックを表示します。

メトリック CPU 使用率、負荷平均、メモリ使用量、ディスク使用量など。

使用法 HM をホスティングするサーバーのハードウェアリソースのいくつかのビューを提供します。ノードレベルでリソースのボトルネックを特定するのに役立ちます。

Prometheus の概要

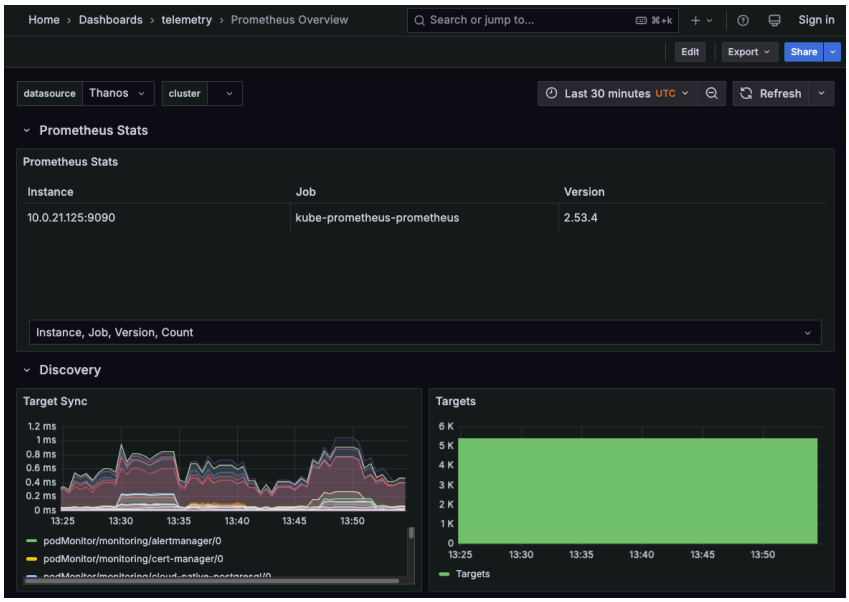


図 35: Telemetry Prometheus dashboard

これは、Grafana の公式ライブラリからの事前ビルドされた Prometheus mixin です。

説明 HM でメトリックを収集および出荷するために使用される Prometheus モニタリングシステムの状態とパフォーマンスを監視します。

メトリック Prometheus 統計、検出とターゲット、スクレイピング間隔と失敗、クエリレートなど。

使用法 Prometheus メトリック収集エンジンの状態を監視できます。Prometheus の問題により、監視データが不完全または遅延される可能性があります。

サノス

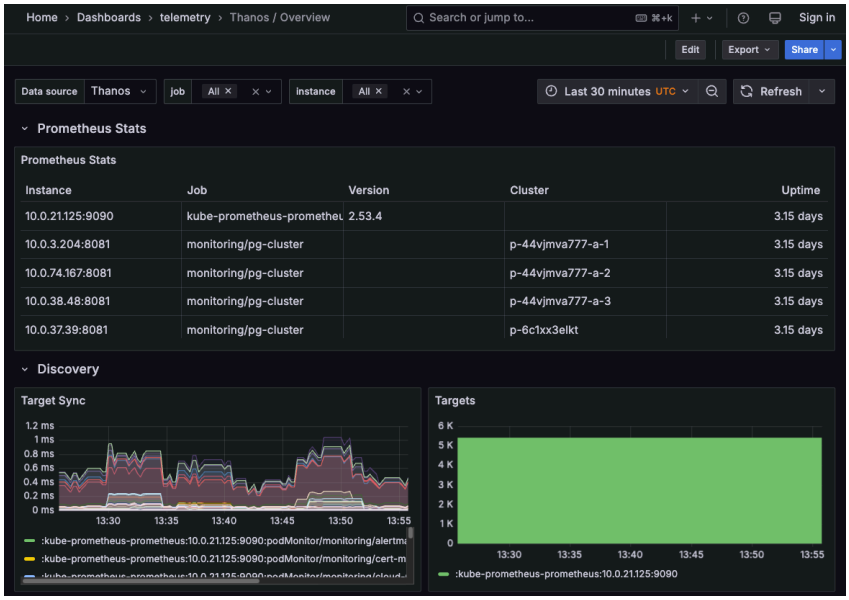


図 36: Telemetry Thanos dashboards

これらは、Grafana の公式ライブラリから事前にビルドされた Thanos mixins です。

Thanos メトリックは、 **Compact** 、 **Overview** 、 **Query** 、**Receive** 、および **Store** などのさまざまなビューを提供する複数のダッシュボードに表示されます。

- **Compact** は、Prometheus メトリックの長期保存とダウンサンプリングに使用されるコンポーネントである Thanos コンパクターをモニタします。
- **概要** は、HM での Thanos 展開の高レベルの概要を提供します。
- **Query** は Thanos クエリ API をモニタし、レートと期間メトリック、エラーなどを提供します。
- **Receive** は Thanos 着信要求をモニタし、レートと期間メトリック、エラーなどを提供します。
- **Store** は、Thanos ストレージジョブ、バケット、ブロック、キャッシュ操作、および関連するエラーをモニタします。

説明 Thanos の長期メトリックストレージシステムのパフォーマンスと状態を監視します。

メトリック Thanos ジョブの統計、スクレイピングインターバル期間、コンパクション、クエリー、着信要求とストアプロシージャー、プロシージャーのレイテンシー、エラー、CPU とメモリの使用量に関するメトリック。

使用法メトリックの長期ストレージが Thanos によって効率的に管理されていることを確認するのに役立ちます。コンパクターに問題があると、ストレージの非効率またはデータの損失が発生する可能性があります。Thanos のクエリと着信要求の処理が遅いと、Grafana ダッシュボードなどの応答性に影響を与える可能性があります。

Other dashboards

電力消費 電力使用量と二酸化炭素排出量

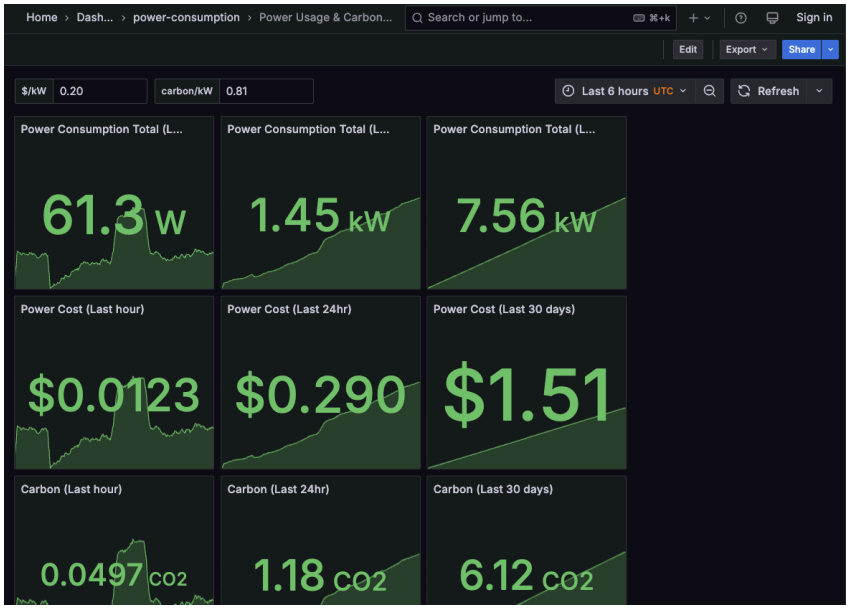


図 37: Power usage and carbon footprint dashboard

説明ハイブリッドマネージャー HM インフラのエネルギー消費と推定二酸化炭素フットプリントを監視します。これを使用して、環境への影響を追跡し、最適化する可能性があります。

このダッシュボードは、re:cinq の Cloud CPU Energy Consumption 計算方法と同様の手順に従って EDB によって設計され、ソースデータとして TEADS dataset から無料で利用可能な電力測定値を使用します。

メトリック過去 1 時間、24 時間、または 30 日間のグラフに分類された電力消費量ワットまたはキロワット単位、推定二酸化炭素排出量、および推定電力コスト。

使用法 HM インフラストラクチャの環境への影響を視覚化し、エネルギー使用の傾向を追跡し、コストと環境への影響を削減するために最適化する領域を特定できる可能性があります。

インフラストラクチャ:cert-manager

説明 TLS 証明書の管理と発行を自動化する Kubernetes アドオンである cert-manager の状態とパフォーマンスを監視します。これを使用して、HM での証明書管理が適切に機能することを確認します。

メトリック発行された証明書の数、証明書要求ステータス保留中、準備完了、失敗)、証明書の有効期限など。

使用法 HM の TLS 証明書が正しく管理され、それらに依存するアプリケーションが有効で最新の証明書を持つことを保証します。停止が発生する前に、証明書の発行または更新に関する潜在的な問題を特定します。

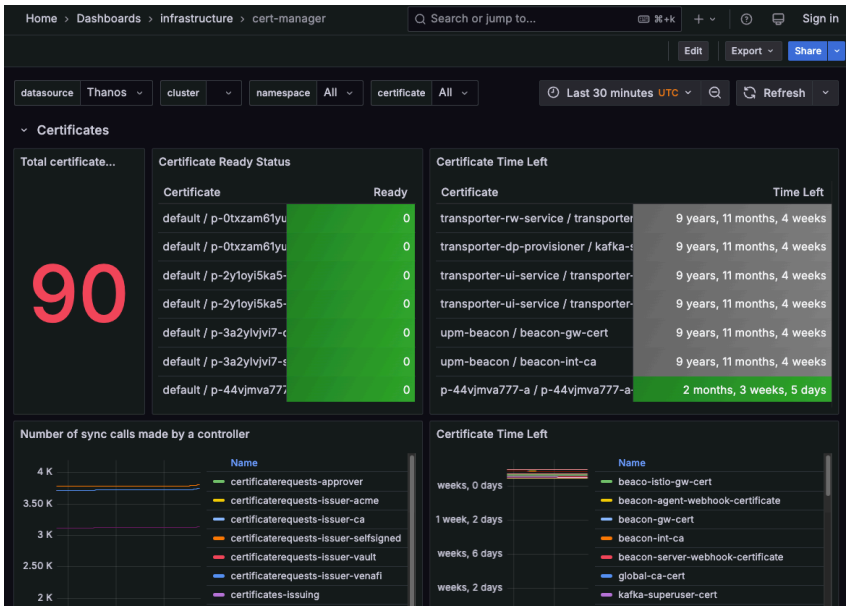


図 38: Cert-manager dashboard

インフラストラクチャ Velero の統計

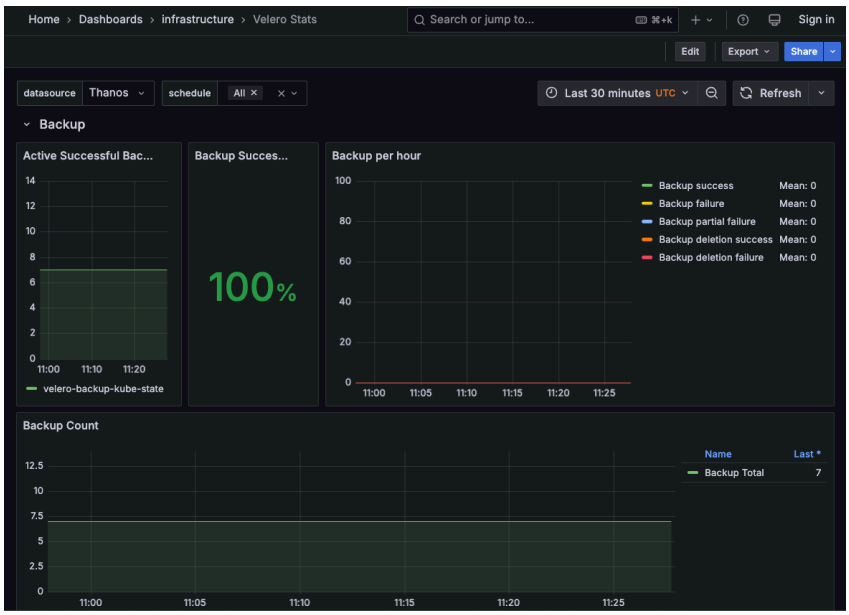


図 39: Velero dashboard

説明 Kubernetes リソースと永続ボリュームのバックアップおよび復元ツールである Velero の状態とパフォーマンスを監視します。これを使用して、HM でのバックアップのステータスと成功を追跡します。

メトリック作成されたバックアップの数、実行されたリストアの数、バックアップとリストア操作のステータス完了、失敗、進行中、バックアップサイズ、リストア期間、バックアップまたはリストア中に発生したエラー、スケジュールステータス。

使用法 HM でのバックアップおよび復元操作の可視性を提供します。バックアップの成功を確認し、復元プロセスを監視し、ディザスターリカバリー戦略の問題を特定できます。

Grafana’s Explore option

Grafana は、Loki によって収集されたログを照会できる **Explore** オプションを提供します。

特定のクラスターからログを照会するには

1. *cluster's ID and namespace* を取得します。 `cnp_instance_id` を選択します。
2. **Label filters** メニューで、 **cnp_instance_id** を選択します。
3. **Select value** メニューで、ポッドの ID を選択します。
4. そのインスタンスのログで特定のキーワードたとえば、「error」や「Exception」を検索するようにクエリーを作成します。

Troubleshooting in Grafana

Grafana ダッシュボードの `no healthy upstream` 警告は、サービスが目的のバックエンドに接続できないことを示しています。この問題は一時的なか、長期間持続する可能性があり、モニタリングおよびデバッグツールの使いやすさに影響を与えます。一般的な原因を理解することは、これらの問題の診断と解決に役立ちます。

潜在的な原因

ポッドのエビクションまたは移動

Karpenter のような動的なノードプロビジョニングとポッドスケジューリングツールは、ノード間でポッドを移動する場合があります。このプロセス中に、ポッドが一時的に使用できなくなる場合があり、再スケジュールされ、新しいノードで準備が整うまで、`no healthy upstream` エラーが発生します。

リソースのオーバースケジュールとメモリ不足 OOM エラー

サービス、特に複雑または大規模なクエリを処理するサービス Loki などは、大量のメモリを消費する可能性があります。通常、これらのツールはクエリを制約しません。彼らは、メモリ割り当てで成功しない可能性のあるクエリを特定してフラグを立てるのではなく、テラバイトのログまたはギガバイトのメトリックと OOM を返す可能性のあるクエリを実行しようとします。

クラスターノードがオーバーサブスクライブされている場合、またはポッドが適切なリソース制限で構成されていない場合、ホストのメモリ圧力が原因で Kubernetes はそれらを強制終了する場合があります。ポッドがすぐに回復した場合でも、OOM イベント中の短期間の使用不可により、`no healthy upstream` 警告がトリガーされる場合があります。

一時的なネットワークの問題または接続の終了

断続的なネットワーク問題、接続リセット、またはヘッダーを受信する前の突然の接続終了も `no healthy upstream` として発生する場合があります。

サービスの使用不可または構成の誤り

一時的な問題ではあまり一般的ではありませんが、 `no healthy upstream` メッセージは、バックエンドサービスが異常である、実行されていない、または誤って構成されていることを示し、Istio がトラフィックを正常にルーティングできないことを示している場合があります。

トラブルシューティングの手順

1. **クエリの最適化**これらの特定のバックエンドの OOM の最も一般的な直接の原因に対処します。Loki、Thanos、および Prometheus はクエリをガードレールしないため、これらのバックエンドが失敗し始めたときに最初に確認することは、広すぎるまたは複雑なクエリが OOM 問題を発生させているかどうかです。

時間ウィンドウを短縮するか、より少ないログメトリックのセットを選択するか、より具体的なフィルタリングストリームセレクター、ラインマッチング、JSON クエリー、正規表現式の回避を使用して、クエリーを絞り込んでみてください。詳細については、Query best practices を参照してください。

1. **リソースの使用率を監視する**監視ワークロード Loki、Thanos、Prometheus のリソース使用量を確認します。*resource utilization metrics for the affected pods* とその基になるノード CPU、メモリを確認します。これは、リソースのオーバースケジュールまたはキャパシティの不足が根本原因であるかどうかを特定するのに役立ちます。例、 *Loki dashboards for logging monitoring* を確認します。ポッドのリソース :ref:``upm-beacon-agent` 複雑な環境ではメモリ制限が不十分です <*upm-beacon-agent* 複雑な環境ではメモリ制限が不十分です>` や **リクエスト** など、構成済みの **リソースのコミットメント** を確認します。

リソースの使用率が高いように見え、OOM エラーが持続する場合は、ポッドのリソース要求/制限またはノードのキャパシティを調整することを検討します。ノードに十分なメモリ容量があるが、ポッドがアンダーサブスクライブされている場合、ポッドのメモリ制限を増やすことを検討します。ポッドがオーバーサブスクライブされ、ノードがサブスクライブされたポッドをホストするためにより多くのメモリ容量を必要とする場合、ノードのメモリを増やすことを検討します。

1. **Kubernetes イベントとログを確認します**ポッドの削除、スケジュール障害、またはノードの変更については、Kubernetes イベント cluster’s logs を確認します。ログは、ポッドの状態と特定の OOM イベントを確認するための良いソースです。これらは、どのポッドとプロセスに障害が発生しているかを理解するのに役立ちます。

2. **ネットワークの状態を確認します。**これはより一般的なチェックであり、OOM 問題の主な原因になる可能性は低いです。基になる *network infrastructure* の状態を確認します。

Notifications

Hybrid ManagerHM を使用すると、受信する通知の種類をカスタマイズできます。2 種類の通知チャネルは次のとおりです。

ユーザー通知電子メール、HM インボックス、またはその両方を介して通知を受信できます。これらの通知は、それらを有効にしているすべてのユーザー、および自分のロールで許可されている場合に対象となります。

システム通知電子メールメールグループ、Webhook、ポケベルデューティ、または任意の組み合わせを介して通知を受信できます。これらの通知は、主に組織内のタスクを自動化するために、イベントごとに 1 回トリガーされます。

HM でのユーザーのロールにより、通知設定へのアクセスの範囲が決まります。通知は次の設定セクションに分類されます。

- アカウント ユーザー通知チャンネル用。
- グローバル ユーザーおよびシステム通知チャネル用。
- プロジェクト ユーザーおよびシステム通知チャネル用。

[アカウント設定] セクションの通知は次のとおりです。

- あなたに割り当てられた新しいプロジェクトロール
- プロジェクトロールがあなたから割り当て解除されました
- 個人アクセスキーの有効期限が間もなくエキスパイア

グローバル設定セクションの通知は次のとおりです。

- 新しいプロジェクトが作成されました
- マシンアクセスキーの有効期限が間もなくエキスパイア

> **注釈**
>
> このセクションは、組織の所有者のみが表示されます。

[プロジェクト設定] セクションの通知は次のとおりです。

- アクティブなクラスターが一時停止
- 一時停止したクラスターが再アクティブ化されました

通知の設定

プロジェクトの所有者/編集者および組織の所有者/管理者は、自分に表示されるイベントの通知を有効/無効にできます。HM インボックス、電子メール、またはその両方で通知を受信できます。

ユーザーが行った通知設定は、そのユーザーにのみ適用されます。電子メール通知が有効になっている場合、電子メールはユーザーのログインに関連付けられた電子メールアドレスに送信されます。

システム通知チャネルは、組織の所有者がグローバルレベルで、またはプロジェクト所有者がプロジェクトレベルで構成できます。デフォルトでは、各組織のプロジェクトはグローバルレベルのシステム通知チャネルを継承します。ただし、プロジェクトレベルでシステム通知チャンネルを有効/無効または再構成して、デフォルト設定をオーバーライドすることができます。

グローバルレベルで通知を設定する

組織の所有者のみがグローバルレベルの通知を設定できます。

Webhook

1. HM コンソールで、プロフィールメニューから **Settings** を選択します。
2. **Notifications** タブを選択します。
3. **Webhook** を選択し、詳細を提供します。
4. Webhook のホスト URL POST または PUT
5. オプションで、次の認証パラメーターのいずれかを提供します。
   - 基本認証ユーザー名とパスワード
   - ベアラートークン
6. **Save** を選択します。

ポケベル当番

Pager Duty 通知チャネルを設定する前に

- 昇格した権限を持つ Pager Duty アカウントへのアクセスが必要です。 - Pager Duty Service は、Events API v2 統合を使用して作成および構成する必要があります。 - Pager Duty アカウントで、目的のサービスに移動し、その **統合**設定にアクセスします。 - Events API v2 統合を追加し、要件に従って構成します。 - 新しく作成した統合のために Pager Duty が提供した統合キールーティングキーをコピーします。

次に、チャネルを構成できます。

1. HM コンソールで、プロフィールメニューから **Settings** を選択します。
2. **Notifications** タブを選択します。
3. **Pager Duty** を選択し、統合キールーティングキーを設定します。
4. **Save** を選択します。

電子メール

1. HM コンソールで、プロフィールメニューから **Settings** を選択します。
2. **Notifications** タブを選択します。
3. **電子メール** を選択し、詳細を入力します。
4. 電子メールの宛先オプションで、システム電子メール通知の電子メールグループ
5. 電子メールの差出人 通知を送信している電子メール
6. Host : 電子メールサービスの URL
7. Port : ポート the email service.
8. Username メールサービスのユーザー名.
9. Password メールサービスのパスワード
10. **Save** を選択します。

プロジェクトレベルで通知を設定する

プロジェクトレベルの通知設定は、グローバル設定をオーバーライドできます。各プロジェクトは、一意の構成を持つことができます。

1. HM コンソールで、 **Projects** タブを選択します。
2. 任意のプロジェクトを選択します。
3. プロジェクトの概要ページに移動します。
4. 左側のナビゲーションで、 **Settings** を選択します。
5. **通知** を選択します。
6. *Webhook* 、 **ポケベル当番** 、または**電子メール** の説明に従って設定を指定します。

HM コンソールで通知を表示する

次のロールのユーザーは通知を表示できます。

- 組織の所有者はグローバル通知を表示できます。
- プロジェクトの所有者/編集者は、プロジェクトの通知を表示できます。
- すべてのユーザーが自分のアカウント通知を表示できます。

各通知は、HM で複数のロールが割り当てられているユーザーが属しているレベルとプロジェクトを示します。

通知を表示するには、HM コンソールの上部にあるベルアイコンを選択します。通知を読んだり、未読としてマークしたり、アーカイブしたりできます。

通知設定の管理

1. HM コンソールで、プロフィールメニューから **My Account** を選択します。
2. **Notifications** タブを選択します。通知は、アカウント、グローバル、および使用可能なプロジェクトごとにグループ化されています。
3. **Account** 、 **Global** 、または **Project** セクションまでスクロールして、管理するイベントを表示します。
4. エンティティの横にある右矢印を選択して、イベントのリストを展開します。
5. 通知ごとに、次のことができます。 - イベント名の前のトグルを使用して、特定のイベントの通知の受信をオプトインまたはオプトアウトします。- 電子メールとアプリ内用のチェックボックスを使用して、イベントの電子メールと HM コンソールアプリ内通知をカスタマイズします。

Alerts

アラートは、Postgres クラスターのパフォーマンスに影響を与える重要なイベントについて積極的に通知する Hybrid ManagerHM 可観測性エコシステムに不可欠です。重要なパフォーマンスメトリックを継続的に監視し、事前定義されたしきい値を超過するたびにアラートをトリガーします。この堅牢なアラートシステムは、HM エコシステムにシームレスに統合され、すべてのリージョンの Postgres クラスターとそのインフラストラクチャから収集されたリアルタイムデータを活用します。アラートは、HM で作成されたプロジェクトごとにデフォルトで有効になっています。

アラートシステムは、直感的な通知を介してタイムリーな通知を保証し、情報に基づいた迅速なアクションを可能にします。包括的なアラート構成オプションにより、特定のしきい値と間隔を構成して、アラートエクスペリエンスをカスタマイズできます。モニタリングとアラートのこのシームレスな統合により、完全な可視性が提供され、潜在的な問題への迅速な対応が可能になり、最適なクラスターパフォーマンスを維持できます。

アラートを表示する

クラスターの詳細ページの **モニタリング**タブにあるアラートにアクセスし、特定のクラスターのアラートの詳細を表示できます。

- HM クラスターの場合
    1. HM コンソールで、 **Projects** を選択します。
    2. リストからプロジェクトを選択します。
    3. 左側のナビゲーションで、 **Clusters** を選択します。
    4. 準備ができているクラスターを選択します。
    5. On クラスターの詳細ページで、 **Monitoring** タブを選択します。
- 外部 HM クラスターの場合
    1. HM コンソールで、 **Estate** を選択します。
    2. すべてのクラスターをリストしているテーブルに移動します。
    3. 管理が自己管理である準備ができているクラスターを選択します。
    4. クラスターの詳細ページで、 **モニタリング** タブ。

[**Monitoring**] タブには、アクティブなアラートリストが表に表示されます。このタブの上部には、アクティブなアラートの概要が表示され、重大度レベルが高、中、および低に設定されています。

次を使用して特定のアラートを検索できます。

- クエリベースの検索のための検索ボックス
- 特定の日付範囲ベースの検索の日付範囲セレクター
- 重大度レベルベースの検索のフィルタ
- 重大度、アラート開始時間、およびアラート名に基づいてアラートリストを並べ替える並べ替えユーティリティ
- アラートリストを更新するための更新ボタン
- 設定に従ってテーブルに表示する列を選択する列セレクター
- アラートリストを CSV 形式でダウンロードするための Download ボタン

アラートを手動で更新するには、 **Refresh** を選択します。最後の更新間隔は、アラートテーブルの下部に表示されます。

Node 列でノードを選択して、アラートがトリガーされたときの状態でチャートを表示します。ノードを選択すると、**Monitoring** タブのフィルタが適用され、関連するチャートが表示されます。

CSV のダウンロード — 右上隅の [Download] ボタンを使用して、アラートの詳細を CSV 形式でダウンロードできます。CSV ファイルには、次の列が含まれています。

- アラートの説明
- スタート時間
- アラートの重大度
- アラートをトリガーするメトリック

・ アラートが発生したノード

- アラートしきい値
- クラスター ID
- プロジェクト ID

アラートの構成

プロジェクトレベルでアラートを構成できます。プロジェクト管理者はアラートを構成できます。アラートが構成および適用されると、システムがアラートを完全に処理するまでに数分を必要とします。

アラートを構成するには

1. HM コンソールの上部で、 **Projects** を選択します。
2. リストからプロジェクトを選択します。
3. 左側のナビゲーションで、 **Settings** を選択します。
4. **Alerts** を選択して、アラート構成にアクセス/編集します。

アラートごとに、[**Edit**] ボタンを選択して次のことを行います。

- 有効/無効 —必要に応じて、特定のアラートをアクティブまたは非アクティブにします。
- Notify After — アラートがトリガーされるまでの時間を設定します。
- しきい値 —超過したときにアラートをトリガーするパーセンテージまたは値を指定します。

デフォルト構成のアラートのリスト

これらのアラートはデータベースレベルで設定できます。

- 接続の使用状況
- 最近の WAL アーカイブは成功しませんでした
- ノードダウン
- バイト単位のレプリケーションラグ
- 合計接続数
- 非アクティブなレプリケーションスロットの合計
- トランザクションのロールバック

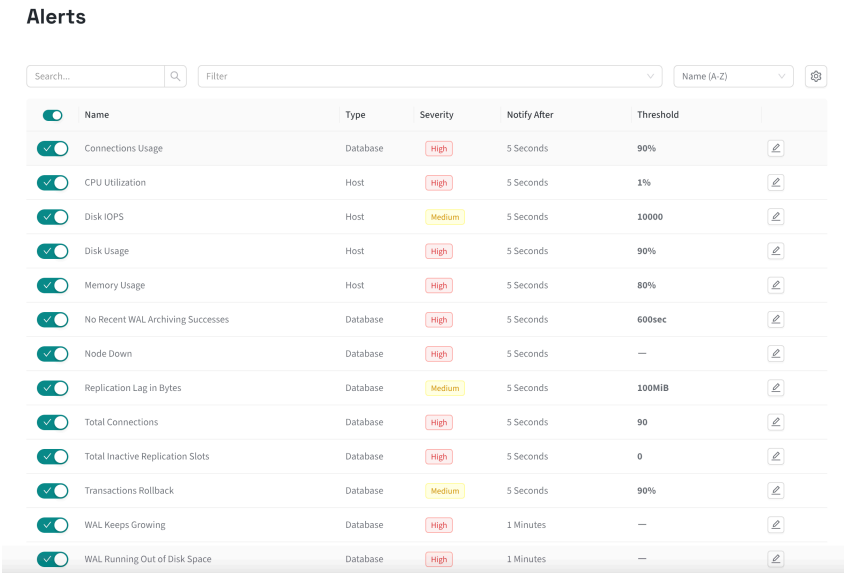


図 40: Alerts

- WAL は成長し続けます
- WAL のディスク領域が不足しています

これらのアラートはホストレベルで設定できます。

- CPU 使用率
- ディスク IOPS
- ディスクの使用量
- メモリ使用量

!!!note ノート

- アラート構成に加えられた変更は、自動的に保存されます。 - 技術的な制限のため、すべてのアラートは外部 HM クラスターでは使用できません。

Upgrading Hybrid Manager

ロール: インフラストラクチャエンジニア

前提条件

- Kubernetes クラスターへの管理アクセス。
- 現在のインストールに使用される `values.yaml` ファイルへのアクセス。
- **必要なツール** * `helm` 最新バージョンを推奨

1.2.3 → 1.3.0 からのアップグレード

このリリースでは、モニタリング用の基になるストレージ Thanos に大きな変更を導入し、Spire ID サービスを追加します。

システムレベルの変更を実行する EKS のみ

Amazon EKS で実行している場合、IAM ロールを更新して、NIM モデルが S3 にアクセスできるようにする必要があります。

- **アクション**サービスアカウントに関連付けられている IAM ロールを編集します。
- **更新** 条件"`system:serviceaccount:m- *:m-* `" を信頼関係ポリシーに追加します。

`values.yaml` を更新する

新しいパラメーターに対応するように構成ファイルを変更します。

1. **画像タグを更新します。**

```
bootstrapImageTag: v1.3.0
```

1. **監視用のストレージを増加します。**

このリリースでは、Thanos Compactor がより多くのストレージを必要とします。

```
parameters:
  upm-thanos:
    compactor_pvc_storage: 100Gi
```

1. **新しい必須のデフォルトを設定します。**

次の値は空にすることができないため、定義する必要があります。

```
parameters:
  upm-beaco-ff-base:
    aes_key_rotation_interval: "24h" # Example value
  upm-istio-gateway:
    enable_server_session: true
    session_duration_seconds: 3600 # Example value
```

1. **Spire の信頼ドメインを構成します。**

Spire がプラットフォームに含まれるようになりました。このインストールには、一意の信頼ドメインを設定する必要があります。

```
parameters:
  global:
    trust_domain: <your-unique-trust-domain>
```

アップグレードを実行する

```
helm repo update

helm upgrade --install edbpgai-bootstrap enterprisedb-edbpgai/edbpgai-bootstrap \
   -n edbpgai-bootstrap \
   -f ./values.yaml \
   --version 1.3.0
```

アップグレード後の予想

- **ポータルのダウンタイム**内部ビーコンデータベースがオンラインになるまでに最大 **30 分**かかる場合があります。この間、ポータルは利用できなくなる場合があります。
- **データベースの再起動**このアップグレード中に、ツリー内の barman-cloud からプラグインアーキテクチャに移行するときに、すべての顧客データベースが再起動します。
- **モデルホスティング** kubectl を介して以前に作成したモデルは、UI を使用して再インストールする必要があります。
- **移行ポータル** 1.3.0 より前に作成されたデータベースは、移行の宛先ターゲットとして使用できません。

Hybrid Manager Multi-DC deployment guide

概要

複数のデータセンターでハイブリッドマネージャーを実行する理由は何ですか?

マルチ DC は、 Postgres ワークロードと Hybrid ManagerHM コントロールプレーン CP の高可用性とディザスターリカバリーを提供します。

- サイト損失 DR の生き残りウォームなセカンダリサイトの準備をしておきます。プライマリ DC が利用できない場合は、セカンダリでレプリカをプロモートし、サービスを復元します。
- ダウンタイム HA を最小限に抑えるワークロードがもう一方のサイトで継続されているときに、一方のサイトでメンテナンスまたは移行を実行します。
- データの保護 RPO 2 番目の DC への継続的レプリケーションにより、単一サイトのバックアップのみと比較して、潜在的なデータ損失が削減されます。
- ブラスト半径を減らす一方の DC の障害、構成ミス、またはノイジーネイバーは、他方の DC をダウンさせません。
- コンプライアンス/主権を満たす制御を集中させながら、特定の地域または施設にコピーを保持します。
- 大規模な動作読み取りトラフィックを分割したり、アップグレードをステージしたり、DC 全体でブルー/グリーンのカットオーバーを実行したりします。

RTO/RPO の概要

- **RTO サービスの復元時間**通常は分、プロモーション/カットオーバーランブックおよび DNS/LB の変更によって駆動されます。

RPO データ損失ウィンドウ

- 非同期レプリケーション DC 全体で共通非常に低いですが、ゼロではありませんベストエフォート秒。
- 同期レプリケーション 遅延依存ゼロデータ損失に近づくことができますが、クロス DC 遅延が追加され、堅牢な低遅延リンクが必要です。

このガイドが役立つこと

- 同じプロバイダー/オンプレミスファミリー上の 2 つの HM クラスタープライマリ↔セカンダリを接続します。
- オブジェクトストレージ同一の edb-object-storage シークレットを調整して、バックアップ/アーティファクトを両方の DC で使用できるようにします。
- SPIRE フェデレーション 8444/TCP を介したトラストドメインを有効にして、プラットフォーム ID がクロスサイトで動作するようにします。
- プライマリがセカンダリを管理対象場所として登録し、そこでプロビジョニングできるようにエージェントビーコンを配線します 9445/TCP
- オプショナルクロスサイトメトリックとログのフェデレーションテレメトリ Thanos/Loki 。
- 一方の DC にプライマリ、もう一方の DC にレプリカを使用して Postgres トポロジを準備します。レプリカを昇格させることにより手動フェイルオーバーを実行します。

電流制限

- 2 つのサイトプライマリとセカンダリ。
- 手動フェイルオーバープライマリがダウンしている場合、セカンダリのレプリカを昇格させます。
- 同じクラウド/オンプレミスファミリークロス CSP マルチ DC はサポートされていません。

アーキテクチャの概要

- コントロールプレーン 2 つの HM クラスター、SPIRE を介してフェデレーション。プライマリは、ビーコンを介してロケーションとしてセカンダリを「管理」します。
- データノード DC-A の Postgres プライマリ、DC-B のレプリカデフォルトで非同期。
- ストレージバックアップ/アーティファクトのためのサイト全体での共有/一貫したオブジェクトストア構成。
- テレメトリオプショナルサイト全体でメトリック/ログを表示するための Thanos/Loki フェデレーション。

これは誰に向けたものですか?

これは、単一の DC が提供できるよりも高い復元力を必要とし、明確に定義された RTO / RPO 目標を使用して手動でリハーサルされたフェイルオーバープレイブックを実行することに慣れているチーム向けです。

前提条件

アーキテクチャの前提条件

- 2 つの HM クラスターが使用できますプライマリとセカンダリ Kubernetes コンテキスト構成済み。
- 各クラスターには、一意の SPIRE 信頼ドメインがあります。
- ネットワーク接続
  - 8444/TCP クラスター間でオープン SPIRE バンドルエンドポイント。
  - 9445/セカンダリ → プライマリからの TCP ビーコン gRPC.
  - ツール: kubectl、jq、および YAML をローカルで編集する場合は yq.
  - 同じプロバイダー/オン-prem ファミリークロスクラウドなし。
- ツール - `jq` 、`yq` 、AWS コマンドライン
- スクリプト - この手順全体で使用を参照するスクリプトが多数あります。スクリプトにアクセスするには、 EDB Professional Services または EDB Support Center に連絡してください。

必要な環境変数を検出して設定する

1. `kubectl` に、どのクラスターがプライマリとセカンダリであるかを指示します

後続のすべての検出コマンドで使用する kube コンテキストを設定します。

```
export KUBE_CONFIG_PRIMARY_CONTEXT="<primary-kube-ctx-or-arn>"
export KUBE_CONFIG_SECONDARY_CONTEXT="<secondary-kube-ctx-or-arn>"
```

1. プライマリポータルの FQDN を検出しますビーコン/テレメトリによって使用されます

プライマリクラスターでビーコンゲートウェイホストを照会し、`PRIMARY_PORTAL_URL` としてエクスポートします。

```
export PRIMARY_PORTAL_URL="$(
kubectl --context "$KUBE_CONFIG_PRIMARY_CONTEXT" \
    -n upm-beacon get gw beacon-server -o json \
| jq -r .spec.servers[1].hosts[0]
)"
```

1. (オプション) セカンダリポータルの FQDN を検出する

テレメトリもフェデレートする場合は、セカンダリポータルホストを `SECONDARY_PORTAL_URL` としてキャプチャし、それに応じて環境変数を設定します。

```
export SECONDARY_PORTAL_URL="$(
kubectl --context "$KUBE_CONFIG_SECONDARY_CONTEXT" \
    -n upm-beacon get gw beacon-server -o json \
| jq -r .spec.servers[1].hosts[0]
)"
```

1. プライマリのビーコン gRPC エンドポイントを導き出す

ビーコンは、プライマリポータルホストの:9445 でリッスンします。

```
export BEACON_SERVER_ENDPOINT_PRIMARY="${PRIMARY_PORTAL_URL}:9445"
```

1. プライマリの SPIRE 信頼ドメインを検出します

プライマリから SPIRE サーバー構成を読み取り、 TRUST_DOMAIN_PRIMARY をエクスポートします。

```
export TRUST_DOMAIN_PRIMARY="$(
kubectl --context "$KUBE_CONFIG_PRIMARY_CONTEXT" \
    -n spire-system get cm spire-server \
    -o jsonpath="{[data][server\.conf]}" \
| jq -r .server.trust_domain
)"
```

1. セカンダリの SPIRE 信頼ドメインを検出します

セカンダリでも同じことを行い、TRUST_DOMAIN_SECONDARY をエクスポートします。

```
export TRUST_DOMAIN_SECONDARY="$(
kubectl --context "$KUBE_CONFIG_SECONDARY_CONTEXT" \
    -n spire-system get cm spire-server \
    -o jsonpath="{[data][server\.conf]}" \
| jq -r .server.trust_domain
)"
```

1. 管理対象セカンダリの場所のラベルを選択します

任意の一意のラベルを選択しますプライマリで managed-<label> として表示されます。

```
export SECONDARY_LOCATION_NAME="secondary"
```

1. (EKS / IRSA ヘルパーのオプション) EKS 識別子を設定します

ヘルパーを使用して S3 信頼ポリシーを更新し、オブジェクトストアシークレットをコピーする場合にのみこれを行います。このステップには、PRIMARY_EKS 、SECONDARY_EKS 、および AWS_PROFILE が必要です。

```
export PRIMARY_EKS="<region>:<primary-eks-name>"
export SECONDARY_EKS="<region>:<secondary-eks-name>"
export AWS_PROFILE="<aws-profile>"
```

1. 続行する前のサニティチェック

キー変数が設定されていることを確認します欠品している場合はエラーになります。

```
: "${KUBE_CONFIG_PRIMARY_CONTEXT:?}"; : "${KUBE_CONFIG_SECONDARY_CONTEXT:?}"
: "${PRIMARY_PORTAL_URL:?}"; : "${BEACON_SERVER_ENDPOINT_PRIMARY:?}"
: "${TRUST_DOMAIN_PRIMARY:?}"; : "${TRUST_DOMAIN_SECONDARY:?}"
: "${SECONDARY_LOCATION_NAME:?}"
```

以下は、テレメトリーまたは EKS ヘルパーを実行する場合のオプションのチェックです。

```
: "${SECONDARY_PORTAL_URL:?Set if doing telemetry federation}" || true
: "${PRIMARY_EKS:?Set if using EKS helper}" || true
: "${SECONDARY_EKS:?Set if using EKS helper}" || true
: "${AWS_PROFILE:?Set if using EKS helper}" || true
```

注釈

後で新しいシェルを開くか、新しい CI ステップを実行する場合、これらのエクスポートを再実行しますまたはファイルに保存してソースします。

場所をまたがるオブジェクトストレージ

HM は、バックアップ、アーティファクト、WAL、および内部バンドルにオブジェクトストアを使用します。**マルチ DC では、両方のクラスターが同じオブジェクトストア構成を使用する必要があります。**

キー要件

各クラスターには、デフォルトの名前空間に `edb-object-storage` という名前の同一の Kubernetes シークレットが必要です。

注釈

セカンダリの場所に HM をインストールする前に、*edb-object-storage* を作成/同期します。

プライマリのインストール中にオブジェクトストレージを設定するときに、初期シークレットを作成します。これをセカンダリに複製します。

```
#  Clean slate on Secondary

kubectl delete secret \
- -context=$KUBE_CONFIG_SECONDARY_CONTEXT \
- n default edb-object-storage || true

#  Copy Primary → Secondary

kubectl get secret \
- -context=$KUBE_CONFIG_PRIMARY_CONTEXT \
```

(次のページに続く)

(前のページからの続き)

```
- n default edb-object-storage -o yaml | \
kubectl apply \
- -context=$KUBE_CONFIG_SECONDARY_CONTEXT \
- n default -f -
```

EKS IRSA トラストポリシー

サービスアカウント IRSA の S3 + IAM ロールを使用する場合、ロールは両方のクラスターの OIDC プロバイダーを信頼する必要があります。

1. 信頼ポリシーを手動で更新して両方の OIDC 発行者を含める、または
2. ヘルパー `object-storage-on-multi-dc.sh` ヘルパースクリプトを取得します。
3. ヘルパースクリプトを使用してシークレットをコピーし、OIDC プロバイダーをロールの信頼ポリシーに自動的に追加します。

```
./object-storage-on-multi-dc.sh \
-p <region>:<primary-eks-name> \
-s <region>:<secondary-eks-name>[,<region>:<another-eks>] \
-a <aws-profile>
```

`object-storage-on-multi-dc.sh` ヘルパースクリプトは、プライマリの edb-object-storage シークレットから IAM ロール ARN を読み取り、同じシークレットを各セカンダリにコピーし、セカンダリクラスターの OIDC プロバイダーがロール信頼ポリシーにない場合は追加します。

1. 検証チェックリスト
   - シークレットは同一です .data のみを比較します。
   - 両方のクラスターはバケットをリスト/書き込むことができます簡単なポッド/ジョブテスト。
   - IRSA ロールの信頼には、両方の OIDC プロバイダー EKS の場合が含まれます。

セットアップオプション

オプション A ワンショットまたはオプション B ステップバイステップを選択します。同じ最終状態に到達します。

オプション A — クイックスタートマスタースクリプト

マスタースクリプトは、オブジェクトストレージ同期、SPIRE フェデレーション、ビーコン配線、およびオプションのテレメトリを 1 つのパスで実行します。

1. `master-install.sh` スクリプトを取得します。
2. 次のスクリプトを実行します。

```
cd scripts/multi-dc
./master-install.sh
```

1. 便利なフラグ

```
./master-install.sh --dry-run \
--skip-object-store --skip-federation --skip-beacon --skip-telemetry
```

1. 確認します。

- SPIRE フェデレーションがリストされていることを確認します。

```
kubectl -n spire-system exec svc/spire-server -c spire-server -- \
/opt/spire/bin/spire-server federation list
```

- プライマリに登録されたセカンダリの場所を確認する

```
kubectl get location
```

オプション B — 手動セットアップ 詳細/カスタマイズ可能

必要なスクリプトを取得します

マルチ DC を手動でセットアップするには、これらのスクリプトを取得する必要があります。

- update_objectstore_secrets.sh
- apply-federated-domain.sh
- configure-beacon-primary.sh
- configure-beacon-secondary.sh
- install.sh テレメトリーフェデレーションを設定する場合

スクリプトを実行してマルチ DC をセットアップする

これらのスクリプトを次の順序で実行します。

- update_objectstore_secrets.sh
- apply-federated-domain.sh
- configure-beacon-primary.sh
- configure-beacon-secondary.sh
- install.sh テレメトリーフェデレーションを設定する場合

1. オブジェクトストレージの同期

```
./update_objectstore_secrets.sh
# (EKS only, if IRSA)
./eks-object-storage-on-multi-dc.sh -p $PRIMARY_EKS -s $SECONDARY_EKS -a $AWS_PROFILE
```

1. SPIRE フェデレーション

```
./apply-federated-domain.sh $KUBE_CONFIG_PRIMARY_CONTEXT $KUBE_CONFIG_SECONDARY_
↪CONTEXT
```

1. 検証します。

```
kubectl -n spire-system exec svc/spire-server -c spire-server -- \
/opt/spire/bin/spire-server federation list
```

1. ビーコンの配線

```
./configure-beacon-primary.sh   $TRUST_DOMAIN_SECONDARY
./configure-beacon-secondary.sh $BEACON_SERVER_ENDPOINT_PRIMARY $TRUST_DOMAIN_PRIMARY
↪$SECONDARY_LOCATION_NAME
```

1. 検証します。

```
kubectl get location
```

1. オプションのテレメトリーフェデレーション

```
cd thanos && ./install.sh -l secondary -p $PRIMARY_PORTAL_URL -s $SECONDARY_PORTAL_URL
./install.sh -l primary  -p $PRIMARY_PORTAL_URL -s $SECONDARY_PORTAL_URL

cd ../fluent-bit && ./install.sh -l primary -p $PRIMARY_PORTAL_URL -s $SECONDARY_
↪PORTAL_URL
./install.sh -l secondary -p $PRIMARY_PORTAL_URL -s $SECONDARY_PORTAL_URL
```

SPIRE フェデレーションの詳細 what/why/how

SPIRE フェデレーションにより、各 SPIRE サーバーはピア信頼ドメインを信頼し、そのバンドルを継続的に更新できます 8444/TCP が必要です。CRD または spire-server フェデレーション CLI を介して構成できます。

一般的なフロー CRD ベース

- 各クラスターから ClusterFederatedTrustDomain マニフェストを生成しますヘルパースクリプト。
- それらをクロス適用します A→B、B→A。
- spire-server フェデレーションリストで検証します。

フェデレーション後サイトを越える必要があるワークロード ID には、`federatesWith: "<peer-trust-domain>"` を持つ `ClusterSPIFFEID` が必要です。

ビーコン構成プライマリ&セカンダリ

ビーコンにより、プライマリ HM はセカンダリを管理対象「ロケーション」として登録し、そこでプロビジョニングできます。

プライマリ: セカンダリの信頼ドメインを許可します

1. 現在の値をエクスポートして編集します。

```
kubectl get configmap -n edbpgai-bootstrap -l app=edbpgai-bootstrap -o yaml \
| yq .items[0].data["values.yaml"] > /tmp/primary-boot-values.yaml
```

1. /tmp/primary-boot-values.yaml を編集します。

```
beaconServer:
additionalTrustDomains:
    - "<secondary-location-trust-domain>"
```

1. エージェントビーコンインストールスクリプト install-dev.sh を取得します。
2. ビーコンを再インストールします。

```
./install-dev.sh -f <provider> -a install -c upm-beacon -v <beacon-version> -p /tmp/
↪primary-boot-values.yaml
```

セカンダリ: エージェントをプライマリビーコンにポイントします

1. セカンダリから

```
kubectl get configmap -n edbpgai-bootstrap -l app=edbpgai-bootstrap -o yaml \
| yq .items[0].data["values.yaml"] > /tmp/secondary-boot-values.yaml
```

1. /tmp/secondary-boot-values.yaml を編集します。

```
parameters:
upm-beacon:
    beacon_location_id: "secondary"   # unique label
beaconAgent:
beaconServerAddress: "<primary-portal-fqdn>:9445"
beaconServerTrustDomain: "<primary-trust-domain>"
plaintext: false
tlsInsecure: false
inCluster: false
```

1. エージェントビーコンインストールスクリプト install-dev.sh を取得します。
2. ビーコンを再インストールします。

```
./install-dev.sh -f <provider> -a install -c upm-beacon -v <beacon-version> -p /tmp/
↪secondary-boot-values.yaml
```

フェデレートエージェントビーコン SPIFFE ID 必須

1. federate-beacon-spiffe-ids.sh スクリプトを取得します。

2. セカンダリから実行プライマリ信頼ドメインを含む

```
./federate-beacon-spiffe-ids.sh "$TRUST_DOMAIN_PRIMARY"
kubectl rollout restart -n upm-beacon deploy/upm-beacon-server
kubectl rollout restart -n upm-beacon deploy/upm-beacon-agent-k8s
```

1. プライマリから実行しますセカンダリ信頼ドメインを含む

```
./federate-beacon-spiffe-ids.sh "$TRUST_DOMAIN_SECONDARY"
kubectl rollout restart -n upm-beacon deploy/upm-beacon-server
kubectl rollout restart -n upm-beacon deploy/upm-beacon-agent-k8s
```

1. 登録の検証プライマリ

```
kubectl get location
# Expect: managed-<SECONDARY_LOCATION_NAME> with recent LASTHEARTBEAT
```

テレメトリーフェデレーションオプショナル

クロスサイトのメトリック/ログが必要な場合

Thanos メトリック

1. Thanos インストールスクリプト install.sh を取得します。

2. セカンダリ

```
./install.sh -l secondary -p $PRIMARY_PORTAL_URL -s $SECONDARY_PORTAL_URL
```

1. プライマリ:。

```
./install.sh -l primary -p $PRIMARY_PORTAL_URL -s $SECONDARY_PORTAL_URL
```

1. 検証しますポートフォワード thanos-query を検証し、thanos-query-federated を含むエントリの/api/v1/stores をヒットします。

Fluent Bit/Loki ログ

1. Loki インストールスクリプト install.sh を取得します。

2. プライマリ

```
./install.sh -l primary -p $PRIMARY_PORTAL_URL -s $SECONDARY_PORTAL_URL
```

1. セカンダリ

```
./install.sh -l secondary -p $PRIMARY_PORTAL_URL -s $SECONDARY_PORTAL_URL
```

1. ポートフォワード loki-read を検証し、{app="fluent-forward"} を照会します。

> **注釈**
>
> メトリックとログの値に個別の接頭辞を使用します。

- “global.metrics_storage_prefix”
- “upm-loki.logs_storage_prefix”

目標ワイヤビーコンにより、プライマリ HM はセカンダリを管理対象として扱うことができ、DC 全体に PG プライマリ/レプリカをプロビジョニングできます。

前提条件の概要

- SPIRE フェデレーションが既に構成されている 2 つの HM Kubernetes クラスター同じプロバイダー/オンプレミスファミリー。
- 共有オブジェクトストア edb-object-storage シークレットは、両方のクラスターに存在および同一です。
- オープンネットワーク 8444/TCP SPIRE バンドルエンドポイント、9445/TCP Beacon gRPC、および Postgres レプリケーションポート。
- インストールされるツール jq 、yq 。

重要な値を検出します特記事項にある各クラスターに対して実行します。

1. ビーコンが使用するプライマリポータルホストを取得する 追加 9445

```
kubectl get gw beacon-server -n upm-beacon -o json \
| jq -r .spec.servers[1].hosts[0]
```

1. 信頼ドメインの取得各クラスターで実行

```
kubectl get cm spire-server -n spire-system -o jsonpath="{[data][server\.conf]}" \
| jq -r .server.trust_domain
```

再利用する環境変数をエクスポートします。

```
export KUBE_CONFIG_PRIMARY_CONTEXT="<primary-kube-ctx-or-arn>"
export KUBE_CONFIG_SECONDARY_CONTEXT="<secondary-kube-ctx-or-arn>"

export TRUST_DOMAIN_PRIMARY="<primary-trust-domain>"
export TRUST_DOMAIN_SECONDARY="<secondary-trust-domain>"

export BEACON_SERVER_ENDPOINT_PRIMARY="<primary-portal-fqdn>:9445"
export SECONDARY_LOCATION_NAME="secondary"     # any unique label
```

Multi-DC Beacon Helm のインストールプライマリ

1. プライマリの現在の values.yaml を抽出します。

```
kubectl get configmap -n edbpgai-bootstrap -l app=edbpgai-bootstrap -o yaml \
| yq eval .items.0.data["values.yaml"] > /tmp/primary-boot-values.yaml
```

1. /tmp/primary-boot-values.yaml を編集して、セカンダリ信頼ドメインを許可します。

```
beaconServer:
# ...
additionalTrustDomains:
    - "<secondary-location-trust-domain>"
```

1. プライマリへのビーコンのインストール/再インストール例

```
./install-dev.sh -f <provider> -a install -c upm-beacon -v <upm-beacon-version> -p /
↪tmp/primary-boot-values.yaml
```

マルチ DC ビーコン Helm インストールセカンダリ

1. 現在の values.yaml for セカンダリを抽出します。

```
kubectl get configmap -n edbpgai-bootstrap -l app=edbpgai-bootstrap -o yaml \
| yq eval .items.0.data["values.yaml"] > /tmp/secondary-boot-values.yaml
```

1. /tmp/secondary-boot-values.yaml を編集して、このクラスターを管理対象場所として登録し、エージェントをプライマリにポイントします。

```
parameters:
upm-beacon:
    # name can be anything, just not the same as Primary
    beacon_location_id: "secondary"

beaconAgent:
beaconServerAddress: "<primary-portal-fqdn>:9445"
beaconServerTrustDomain: "<primary-trust-domain>"
plaintext: false
tlsInsecure: false
inCluster: false
```

1. セカンダリにビーコンをインストール/再インストールします。

```
./install-dev.sh -f <provider> -a install -c upm-beacon -v <upm-beacon-version> -p /
↪tmp/secondary-boot-values.yaml
```

ビーコンクラスター SPIFFEID フェデレーション両方向

DC を横断する各ビーコン SPIFFE ID には、federatesWith にピア信頼ドメインを含める必要があります。

両方のクラスターでヘルパースクリプトを使用します。

1. セカンダリからプライマリ信頼ドメインを追加します。

```
./federate-beacon-spiffe-ids.sh "$TRUST_DOMAIN_PRIMARY"

kubectl rollout restart -n upm-beacon deploy/upm-beacon-server
kubectl rollout restart -n upm-beacon deploy/upm-beacon-agent-k8s
```

1. プライマリからセカンダリ信頼ドメインを追加します。

```
./federate-beacon-spiffe-ids.sh "$TRUST_DOMAIN_SECONDARY"

kubectl rollout restart -n upm-beacon deploy/upm-beacon-server
kubectl rollout restart -n upm-beacon deploy/upm-beacon-agent-k8s
```

スクリプトは、ビーコンクラスター SPIFFEID オブジェクトを見つけ、欠落している場合は federatesWith リストを作成し、ピア信頼ドメインが存在しない場合は追加します。

配線の検証

1. プライマリ上 管理対象セカンダリの場所をリストする必要があります

```
kubectl get location
```

1. 各クラスターに存在する SPIRE フェデレーションを検証する

```
kubectl -n spire-system exec svc/spire-server -c spire-server -- \
/opt/spire/bin/spire-server federation list
```

1. 予想されること

- kubectl get location は、 managed-<SECONDARY_LOCATION_NAME> と最近の LASTHEARTBEAT を表示します。
- federation list は、 bundle endpoint profile: https_spiffe およびピアの:8444 URL との 1 つの関係ピア信頼ドメインを示します。

クロス DC Postgres トポロジを作成する

この時点で、HM はセカンダリの場所にプロビジョニングできます。実際の DB トポロジを選択して作成することもできます。

一般的なフロー

1. プライマリ HM から、プライマリ DC に Postgres プライマリを作成します。
2. プライマリ HM から、セカンダリ DC にレプリカクラスターを作成します管理対象セカンダリの場所を選択します。
3. レプリケーションモード同期/非同期を確認し、レプリケーションラグが SLO を満たしていることを監視します。
4. バックアップが両方の DC から共有オブジェクトストアに書き込まれていることを確認し、リストアをテストします。

操作上の注意事項

- DB TLS は、SPIRE / Beacon プラットフォーム ID とは別です。ポリシーごとに PG TLS を構成します。
- 各 DC のストレージクラスが PG IOPS/レイテンシーを満たしていることを確認します。
- サイト間でレプリケーションポートを開きます。

検証エンドツーエンド

1. フェデレーション関係を検証する

```
kubectl -n spire-system exec svc/spire-server -c spire-server -- \
/opt/spire/bin/spire-server federation list
```

1. セカンダリの場所が登録されていることを確認するプライマリ

```
kubectl get location
```

1. セカンダリへのプロビジョニングの検証

- プライマリ HM から、小規模なテストワークロードをセカンダリの場所に展開します。
  - テレメトリオプショナル Thanos ストアはフェデレーションピアを表示します。Loki クエリーは、Secondary.
  - オブジェクトストレージからタグ付けされたログを返します。両方のクラスターはバケットの読み取り/書き込みができます。シークレットは同じです。

手動フェイルオーバーランブック

プライマリからセカンダリへの手動フェイルオーバー手順

1. プライマリへの書き込みを静止しますメンテナンスモード/LB カットオーバー。
2. セカンダリのレプリカをプライマリにプロモート HM ワークフロー/スクリプトごとに。
3. クライアント DNS / LB をセカンダリにリダイレクトします。
4. 観察します書き込みが成功したことを確認します。レプリケーションロールが更新されました。

5. 元のプライマリが戻ったら、新しいプライマリのレプリカとして再シードします。オプションで、後の削減を計画します。

オペレーターのヒント

- カットオーバーのために DNS TTL を十分に低く保ちます。
- ダウンタイムを追跡して RTO を測定します。
- プロモーション後のバックアップを検証します。

障害対応

- 問題 **フェデレーション関係がありません**
  - 再生成およびクロス適用 `ClusterFederatedTrustDomain` CRs.
  - 8444 / TCP の到達可能性を確認します。
- 問題 **``kubectl get location`` にリストされていないセカンダリ**
  - 両側のビーコン値を再確認します。 beacon server/agent.
  - を再起動しますプライマリポータルへの 9445/TCP 到達可能性を確認します。トラストドメインが正しいです。
- 問題 **セカンダリ**
  - でオブジェクトストアアクセスが失敗する 再同期 `edb-object-storage` .
  - EKS / IRSA の場合セカンダリ OIDC がロールの信頼ポリシーに含まれていることを確認します。
- 問題 **テレメトリフェデレーションがありません**
  - 正しい-l プライマリ|セカンダリフラグと一意のプレフィックスで再インストールします。
  - Thanos `/api/v1/stores` および Loki 読み取り API を確認します。
- 問題 **レプリカの遅延/接続** - ネットワーク ACL / SG、TLS 証明書、およびストレージパフォーマンスを確認します。

付録 A — CLI を介した SPIRE フェデレーションオプショナル

CRD の代わりに spire-server フェデレーション create、list、update、delete、show、fresh を使用してフェデレーションを管理できます。直接サーバー制御を希望する場合、またはデバッグのためにこれを使用します。

付録 B — 簡単な日常チェック

- プライマリの `kubectl get location` は、セカンダリの準備ができていることを示しています。
- Thanos / Loki フェデレーションは正常な場合有効な場合。
- オブジェクトストアの書き込みは両方の DC から成功します。
- SLO 内のレプリケーションラグ。

PGAIHM High Availability & Disaster Recovery (HA/DR)

ハイブリッドマネージャー HM クラスターの HA / DR、ベストプラクティスなどについて学びます。

- *HA/DR planning and best practices*
- *Hybrid Manager backup and recovery*

HA/DR planning and best practices

概要

EDB PostgreSQL 分散 PGD 展開を使用してハイブリッドマネージャー HM を使用してマルチデータセントポロジを構成し、高可用性およびディザスターリカバリー HA/DR 要件を満たすことができます。このトポロジには、プライマリ HM と、複数の Kubernetes クラスターにわたる 1 つ以上のデータ専用 HM が含まれます。

この HA 環境では、プライマリ HM が標準の展開です。データ専用 HM は基本的に同じ HM 展開となりますが、プライマリ HM に接続し、誤ってローカルプロビジョニングを防ぐためのバックエンド調整が含まれます。この接続を促進するキーコンポーネントは、HM 間の通信と証明書の受け渡しを処理するエージェントとビーコンサーバーです。

1 つのクラスターがダウンした場合、アプリケーションは同期されたデータを含む別のクラスターを指すことができ、継続的なオペレーションを保証します。マルチデータセンター設定は、データセンターが停止した場合の冗長性を提供します。中央の HM は、複数のリモート Kubernetes クラスターにわたってデータベースを管理でき、データベース管理を簡素化します。

トポロジ

次のトポロジを使用してマルチプル HM セットアップを展開できます。

- **プライマリおよびデータのみの HM** これは、1 つのプライマリ HM と少なくとも 1 つのデータ専用 HM を使用した基本的なセットアップです。これにより、2 つ以上の Kubernetes クラスターにまたがる PGD 展開が可能になります。
- **複数のデータ専用 HM** システムは複数のデータ専用 HM をサポートできるため、単一のプライマリ HM で多くの Kubernetes クラスターにわたってデータベースを管理できます。これは、それぞれが独自のクラスターを持つ多数のアプリケーションチームを持つ組織に役立ちます。
- **PGD 用の 3 つの HM** 最適な PGD 展開、特に 2 つのデータグループと 1 つの監視グループを使用する場合、3-HM トポロジをお勧めします。この構成により、適切な分散コンセンサスのための奇数クォーラムが保証されます。

マルチデータセンター展開には、メイン HM インスタンスを含むプライマリデータセンターと、データ専用の HM または他の Kubernetes クラスターを含む可能性のあるセカンダリデータセンター、およびマルチプルのデータセンターからアクセスできるグローバルオブジェクトストアが含まれます。

> **注釈**
>
> HM バージョン 1.3 は、異なる場所での複数の HM の展開をサポートしていますが、2 つの場所に制限されています。将来のバージョンでは、他の高度なトポロジが計画されています。

> **注釈**
>
> クロスデータセンター HA/DR データセンターまたはリージョンをまたがるコントロールプレーンの高可用性とディザスターリカバリーは、現在サポートされていません。バックアップとリストアを介したディザスターリカバリーが唯一のオプションです。

要件

HA トポロジを構成するときは、次の要件が適用されます。

- 2 つ以上の既存の Kubernetes クラスターたとえば、Amazon EKS または RedHat OpenShift を使用します。
- Kubernetes クラスターのいずれかに完全に展開されたプライマリ HM。
- 提供された構成用スクリプトを使用して、他の Kubernetes クラスターにインストールされたデータ専用の HM。
- データベースを複数のクラスターにまたがる場合、PGD を使用する必要があります。
- PGD の Witness ノードには、少なくとも 10GB のディスク領域が必要です。
- クラウドおよびマルチリージョン展開の高性能ネットワーキングレイヤー。EKS は、AWS エコシステムとの統合により、これを簡素化します。OpenShift では、より複雑なネットワーク構成が必要になる場合があります。複雑なネットワーク構成、特に OpenShift では、クラスター間の接続の問題が発生する可能性があります。適切なネットワーク設定を確認して、PGD ノード間の通信を許可します。Raft 設定が適切に調整されていない場合、ネットワーク遅延が高いと PGD クラスターが不安定になる可能性があります。症状には、障害の検出と復旧の遅延が含まれる場合があります。Raft 関連の設定を調整して、ネットワーク遅延の増加に対応します。

ネットワーク遅延

PGD がコンセンサス問題にどれだけ早く反応できるかと、存在するネットワーク遅延の量の間には固有のトレードオフがあります。レイテンシーが長くなると、より緩やかなコンセンサス設定が必要になり、障害の検出と対応に時間がかかる可能性があります。一部の Raft 設定は、HM コンソールから直接構成できない場合があります。

リージョンをまたがるクラウド環境と比較して、オンプレミスの密結合環境では、遅延はあまり問題になりません。高レイテンシーシナリオに対応するように、よりリラックスした Raft 設定を構成できます。

pg CLI を使用して、クラスターと対話し、コンソールに公開されていない構成設定、特に Raft 設定を調整できます。`bdr run on nodes` ファンクションを使用して、他のノードでコマンドを実行します。

Kubernetes とフェイルオーバー

Kubernetes は、HM 内でフェイルオーバーと冗長性を処理します。次の構成がサポートされています。

- **単一クラスター、単一ゾーン**複数のノードとポッドを含む 1 つのクラスターと 1 つのゾーンを含む基本的な Kubernetes セットアップ。
- **単一クラスター、複数のゾーン**複数のゾーンに分割された Kubernetes クラスター。ゾーンは別個のラックまたはデータセンターを表すことができます。これは**ストレッチクラスター** と呼ばれます

- **マルチクラスター、マルチロケーション**複数の Kubernetes クラスターは、異なるデータセンターにあり、地理的に離れている可能性があります。

これらの構成は、冗長性とフェイルオーバーメカニズムを介してクラスター内またはゾーン全体で継続的なサービスの可用性を保証することにより、HA ポスチャを実現します。DR は、データセンターの停止などの大規模な障害が発生した場合に、さまざまな場所にあるバックアップとデータレプリケーションを使用してサービスとデータを復旧することにより実現されます。

推奨事項

Kubernetes を使用する場合、次の推奨事項が適用されます。

- HM は HA / DR の簡素化を目指していますが、Kubernetes の概念の基本的な理解は有益です。HA / DR 計画には、ノード、ポッド、ゾーン、クラスターなどの Kubernetes の概念を理解することが重要です。
- ノードに障害が発生した場合にポッドを再スケジュールするには、Kubernetes クラスター内の適切なリソースが必要です。
- HM および PostgreSQL ワークロードを別のワーカーノードに分離します。
- ストレージの種類ローカル、共有、オブジェクト、その場所、冗長性、バックアップ/復元手順を含むストレージについては、注意深く計画することが重要です。ディザスターリカバリーシナリオ、特にデータセンターの障害には、オブジェクトストレージを強くお勧めします。
- ストレッチクラスターの場合、ネットワーク遅延は許容範囲内であり、監視ノードの配置を考慮する必要があります。たとえば、OpenShift は 5 ミリ秒未満を推奨しています。

考慮事項

Kubernetes を使用するときは、次の考慮事項に注意してください。

- ワーカーノードがローカルストレージを使用する場合、ノードに障害が発生してもデータは自動的にフェールオーバーされません。リカバリーにはバックアップからの復元が必要です。
- データの永続性と復旧には、共有ストレージ Ceph や SAN などやオブジェクトストレージなどの適切なストレージソリューションを利用することが重要です。
- HM サービスのデータが特定のゾーンに関連付けられている場合、マルチゾーンシナリオで自動的にフェイルオーバーしない場合があります。
- フェールオーバーシナリオでのデータの可用性は、ストレージの場所とレプリケーションによって異なります。
- Kubernetes は、ノードに障害が発生した場合にポッドの再スケジュールを処理できますが、データの復旧はストレージ構成に依存します。
- 別のゾーンまたはデータセンターへのフェールオーバーには、データレプリケーションとアプリケーションのリダイレクトについて注意深く計画を立てる必要があります。
- 共有または複製されたストレージが不足していると、リカバリが妨げられる可能性があります。
- HM は HA のために Kubernetes 機能を活用しますが、HM 固有の構成を検討することが重要です。

HM バックアップと DR

HM マルチデータセンター環境は、以下を使用して、堅牢なデータ保護戦略を実現します。

- **ボリュームスナップショット**ディスクベースのバックアップ、通常はデータセンターにローカル。
- **Barman:** オブジェクトストレージベースのバックアップ。グローバルオブジェクトストアに保存できます。

推奨事項

HM バックアップに関する次の推奨事項に注意してください。

- Barman バックアップを保存するためのグローバルオブジェクトストレージソリューションをお勧めします。このアプローチでは、データセンター全体でバックアップの可用性を保証できます。
- ボリュームスナップショットと Barman バックアップの両方を使用して、3-2-1 バックアップ戦略 3 つのデータのコピー、2 つの異なるメディア、1 つのオフサイトコピーに従います。
- Kubernetes クラスター全体でバックアップメタデータをレプリケートして、ディザスターリカバリーシナリオでのリストアを簡素化することを検討します。
- Barman とボリュームスナップショットの両方のバックアップを自動化する機能を実装します。
- PostgreSQL データベースに加えて、HM のディザスターリカバリー計画を開発します。
- オンプレミス展開で作業する場合、複数の Red Hat OpenShift インスタンスを接続するためのネットワーク構成を検討します。

考慮事項

HM バックアップに関する次の考慮事項に注意してください。

- セカンダリデータセンターで HM が利用できない場合にバックアップを復元することは簡単ではありません。
- バックアップのメタデータは、バックアップが開始される Kubernetes クラスター内に保存されます。そのクラスターが失われると、メタデータへのアクセスも失われ、復元が複雑になります。
- プライマリ HM なしでセカンダリデータセンターでバックアップを復元するには、手動の手順が必要になる場合があります。
- バックアップとリストアを管理するには、プライマリ HM インスタンスが必要です。
- 自動バックアップは 1 つの場所にのみ送信できます。システムは、場所を超えてバックアップ定義を自動的にレプリケートしません。
- 自動バックアップ用に Barman またはボリュームスナップショットを設定できますが、HM コンソールを介して両方を同時に設定することはできません。
- プライマリ HM が利用できない場合、バックアップの復元には CLI の相互作用と手動構成が必要になる場合があります。
- セカンダリデータセンターでデータを復旧するには、プライマリ HM がなくても、バックアップデータへのアクセスと復元を開始する機能が必要です。

- バックアップのメタデータは各 Kubernetes クラスターに対してローカルであるため、あるクラスターで取得されたバックアップは別のクラスターで自動的に可視されません。

使用する

HM コンソールは、バックアップとリストアを構成および管理するためのツールを提供します。API を使用して、特定の場所へのバックアップを開始することもできます。kubectl を使用して、Kubernetes クラスター内のバックアップを表示できます。

HM クラスターのバックアップとリストアを構成する方法については、*HM backup and recovery* を参照してください。

Hybrid Manager HA/DR recovery

災害はハイブリッドマネージャー HM に影響を与え、使用できなくなる可能性があります。たとえば、EKS ベースのアプライアンスの CSP リージョンの使用不可、またはハードウェアアプライアンスを使用不可にするデータセンターの停止が発生する可能性があります。ディザスターリカバリー DR オプションを使用すると、使用可能なバックアップからポイントインタイムにデータベースを復元できます。

HM バックアップは Velero で処理されます。

HM を回復するには 2 つのシナリオが考えられます。

- **HM を元の場所に復元します** 2 つのデータセンター DC1、DC2 があり、HM は DC1 で実行されています。HM をオブジェクトストレージから DC1 に復元する必要があります。
- **HM を代替の場所に復元します** 2 つのデータセンター DC1、DC2 があり、HM は DC1 で実行されています。HM をオブジェクトストレージから DC2 に復元する必要があります。

DR スコープ

DR 手順は次のことに対応します。

- アプライアンスで作成した Postgres クラスター。
- 関連する s3 互換ストレージ領域のアプライアンスで内部定義されたカスタム管理ストレージの場所。

> **注釈**
>
> DR プロシージャは移行コンポーネントをカバーしていませんが、これらを使用して、元のアプライアンスの `transporter-db` および `migration-db` データベースを復元することはできます。

RTO と RPO

復元を実行する機能、関連する目標復旧時間 RTO、および目標復旧ポイント RPO は、バックアップの頻度とサイズによって異なります。

これらの要素は、環境に割り当てられたクリティカル度とデータの性質によって大きく異なるため、RTO および RPO 値を事前に知ることはできません。環境を適切に準備し、定期的なディザスターリカバリー演習を実行して、RTO および RPO 要件を満たせるようにすることをお勧めします。

バックアップの準備状況

各アプライアンスには、以下を保存するリンクされた s3 互換ストレージがあります。

- 内部バックアップ HM アプライアンスデータ
- Postgres のバックアップ Postgres データベースのバックアップ

プラットフォームで使用する同じバケットにカスタムの保存場所を定義することもできます。

このデータはすべて、災害後に利用できる必要があります。データの重要性と復旧できる災害のレベルによっては、アプライアンスが存在する CSP リージョンまたは物理データセンターの外部にこのデータをレプリケートする必要があります。

Tip

AWS S3 バケットを使用する場合、cross-region replication を使用してレプリケーションを実現できます。

Postgres データベースはデフォルトで継続的バックアップを使用するため、任意の時点で復元できます。これらはバックアップライフサイクルポリシーによってのみ制限されます。

Postgres クラスターの定義などの重要なアプライアンスデータは、Kubernetes オブジェクトとして保存され、Velero バックアップに含まれます。デフォルトでは、このバックアップは、デフォルトのスケジュール `velero-backup-kube-state` で定義されているように、毎日 23 時に発生します。

RPO でより頻繁なバックアップが必要な場合、新しいバックアップスケジュールを定義できます。

危険

アプライアンスソフトウェアの更新によって上書きされる可能性があるため、デフォルトのスケジュールは変更しないでください。

次の例は、必要なリソースを 1 時間ごとにバックアップするカスタムスケジュールを示しています。

```
apiVersion: velero.io/v1
kind: Schedule
metadata:
  name: custom-velero-backup-kube-state
  namespace: velero
spec:
  schedule: 0
  skipImmediately: false
  template:
    includedNamespaces:
    \-
    includedResources:
    \- storagelocations.biganimal.enterprisedb.com
    \- clusterwrappers.beacon.enterprisedb.com
```

(次のページに続く)

(前のページからの続き)

```
  \- backupwrappers.beacon.enterprisedb.com
  snapshotVolumes: false
  ttl: 168h
```

DR プロシージャ

DR 手順は、新しいアプライアンスの展開から、通常の復元手順を使用して Postgres クラスターを復元可能になるまでに実行する必要がある一連の手動手順として定義されます。

> **警告**
>
> この手順は、アプライアンスの 1.0 リリースに基づいており、機能セットの変更に応じて継続的に変更される可能性があります。有効なままにするには、常にテストと更新を行う必要があります。

1. バックアップの可用性を確認する

最初の手順では、使用できないアプライアンスのバックアップ別名「古いバックアップ」が、新しいアプライアンスから到達可能であることを確認します。

これは複数の方法で実現できます。

- 新しいアプライアンスの s3 互換のリンクバケットとしてレプリケートされたバケットを使用するため、古いバックアップを新しいアプライアンスで直接利用できます。
- 破損したアプライアンスのバックアップを、新しいアプライアンスのリンクストレージにコピーする。次の項目をコピーする必要があります。
  - フォーマットの内部 EDB バックアップフォルダー `edb-internal-backups/\<random-string\>`
  - Postgres クラスターバックアップフォルダー `customer-pg-backups`
  - 定義されたカスタム保存場所に対応するフォルダー

> **注釈**
>
> 新しいアプライアンスに定義された内部バックアップフォルダーは、異なる `<random-string>` を持つため、古いものとは異なります。

2. 準備手順

Velero のリカバリバックアップ保存場所を定義する

利用可能なバックアップがある場合は、Velero の新しい保存場所を定義して、破損したアプライアンスのバックアップからリソースを復元できるようにします。これは、これらのバックアップの上書きまたは削除を防ぐための読み取り専用の場所です。

新しい保存場所を定義するには、次の Kubernetes マニフェストを使用します。

```
apiVersion: velero.io/v1
kind: BackupStorageLocation
metadata:
  annotations:
    appliance.enterprisedb.com/s3-prefixes: edb-internal-backups/\<old-backups-ramdom-
↪string\>/velero
  labels:
    appliance.enterprisedb.com/s3-credentials: bound
  name: recovery
  namespace: velero
spec:
  accessMode: ReadOnly
  config:
    insecureSkipTLSVerify: "false"
    region: \<region-of-attached-bucket\>
    s3ForcePathStyle: "true"
  default: false
  objectStorage:
    bucket: \<linked-bucket-name\>
    prefix: edb-internal-backups/\<old-backups-random-string\>/velero
  provider: aws
```

`velero get backup-locations` コマンドを使用して確認します。`Available` として表示される必要があります。ステータスが `Available` でない場合は、Velero ポッドログで s3 バケットの権限エラーを確認します。

リカバリーのための Velero バックアップの選択

古い内部 Velero バックアップが復旧保存場所で利用できるようになると、次のコマンドを使用してリストできます。

```
velero get backups \--selector velero.io/storage-location=recovery
```

通常、復旧には使用可能な最新の完全なバックアップを選択します。復元には両方が必要であるため、Velero バックアップ名と日付と時刻 UTC をメモします。

例

```
NAME                                        STATUS      ERRORS   WARNINGS   CREATED
                    EXPIRES   STORAGE LOCATION   SELECTOR
velero-backup-kube-state- **20241216154403**    Completed   0        0          2024-
↪12-16 16:44:03 0100 CET   5d        recovery           \<none\>
```

注釈

タイムスタンプ値は、以降の手順では*リカバリ日*と呼ばれます。

追加要件

リカバリ手順には、次の要件が適用されます。

- 新しいアプライアンスは、古いものと同じバージョンの Postgres AI ソフトウェア展開を実行している必要があります。
- 古いアプライアンスで使用されていたのと同じ場所 `locations.beacon.enterprisedb.com` カスタムリソースは、新しいアプライアンスで使用できます。 `Locations` は、現在インストール中に作成される内部リソースであり、コンソールでは使用できません。`managed-devspatcher` はデフォルト値です。
- 古いアプライアンスでクラスターを構築するために使用されるコンテナイメージは、新しいアプライアンスで利用できます。

3. 復旧手順

EDB 内部データベース app-db および beacon-db の復元

古いバックアップが利用できるようになると、EDB 内部データベースを復元できます。内部データベースごとに

1. クラスターマニフェストを yaml ファイル `kubectl get \<cluster-name\> \-o yaml \>\ <cluster-name\>.yaml` に保存します。
2. yaml ファイルのクラスター仕様を編集して、バックアップからクラスターが作成されるようにします。

- ブートストラップの **init** セクションを **recovery** セクションに置き換えます。

```
recovery:
  database: \<database name as in the init section\>
  owner: \<owner name as in the init section\>
  source: \<pg-cluster-name\>
  secret:
    name: \<secret name as in the init section\>
  recoveryTarget:
    targetTime: "\<recovery date in YYYY-MM-DD HH:MM:SS+00 format\>"
```

- 次のセクションを追加します。

```
externalClusters:
\- barmanObjectStore:
    destinationPath:  S3://\<linked-bucket-name\>/edb-internal-backups/\<old-backups-
↪random-string\>/databases
    s3Credentials:
      inheritFromIAMRole: true
    wal:
      maxParallel: 8
  name: \<pg-cluster-name\>
```

- 次の接頭辞を inheritedMetdata セクションの appliance.enterprisedb.com/s3-prefixes アノテーションに追加しますリストはコンマ区切りです。

```
edb-internal-backups/\<old-backups-random-string\>/databases/\<db-name\>
```

3. クラスターを削除します。

```
kubectl delete cluster \<cluster-name\>)
```

4. クラスターのバックアップ領域をクリーンアップします。

```
aws s3 rm s3://\<linked-bucket-name\>/edb-internal-backups/\<new-backups-random-
↪string\>/databases/\<pg-cluster-name\>  \--recursive
```

5. 再作成するクラスターの yaml ファイルを適用します kubectl apply \-f \<cluster-name\>.yaml
6. クラスターが正常に復元され、正常な状態になったら、名前空間 upm-beaco-ff-base の accm-server を再起動します。

この時点で、新しいクラスターのポータルが再び利用できるようになります。

Velero プラグインの構成

プラグインは、Kubernetes リソースを正しい状態に復元するのに役立つため、カスタム管理ストレージの場所のみが復元されます。Postgres クラスターリソースは削除されたとおりに復元されるため、後で必要に応じてデータを復元できます。

プラグイン構成は ConfigMap を介して作成されるため、このマニフェストを適用する必要があります。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: velero-plugin-for-edbpgai
  namespace: velero
  labels:
    velero.io/plugin-config: ""
    enterprisedb.io/edbpgai-plugin: RestoreItemAction
data:
  Number of configure disaster recovery mode, so restored items are transformed as␣
↪needed
  drMode: "true"
  Number of configure a date corresponding to the velero backup date. Note the format\
↪!
  drDate: "\<recovery date in YYYY-MM-DDTHH:MM:SSZ format\>"
  Number of old and new buckets for internal custom storage locations
  oldBucket: \<old-appliance-bucket-name\>
  newBucket: \<new-appliance-bucket-name\>
```

カスタム管理ストレージの場所を復元する

次の Velero 復元リソースマニフェストを構成および適用します。

```
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: restore-1-storagelocations
  namespace: velero
spec:
  Number of Change the backup name to a custom backup name as required
  backupName: \<velero-backup-name\>
  includedResources:
  \- storagelocations.biganimal.enterprisedb.com
  includeClusterResources: true
  labelSelector:
    matchLabels:
      biganimal.enterprisedb.io/reserved-by-biganimal: "false"
```

クラスターラッパーを復元する

次の Velero 復元リソースマニフェストを構成および適用します。

```
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: restore-2-clusterwrappers
  namespace: velero
spec:
  Number of Change the backup name to a custom backup name as required
  backupName: \<velero-backup-name\>
  includedResources:
  \- clusterwrappers.beacon.enterprisedb.com
  restoreStatus:
    includedResources:
    \- clusterwrappers.beacon.enterprisedb.com
```

バックアップラッパーを復元する

次の Velero 復元リソースマニフェストを構成および適用します。

```
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: restore-3-backupwrappers
  namespace: velero
spec:
```

(次のページに続く)

(前のページからの続き)

```
Number of Change the backup name to a custom backup name as required
backupName: \<velero-backup-name\>
includedResources:
\- backupwrappers.beacon.enterprisedb.com
restoreStatus:
  includedResources:
  \- backupwrappers.beacon.enterprisedb.com
```

Enabling the Migration Portal AI Copilot

Hybrid ManagerHM を展開した後、外部データベーススキーマを移行するための Launchpad ツール Migration Portal は、すぐに機能します。

ただし、移行ポータル内のスキーマ移行の互換性問題の解決に役立つチャットボットツール AI Copilot はデフォルトで有効になっていないため、有効にするには追加の構成が必要です。

OpenAI、Azure OpenAI、または代替 AI ベンダーを使用して AI コパイロットを構成できます。または、EDB AI Factory 機能を使用して HM で以前に作成したセルフホストモデルと連携するように AI コパイロットを構成できます。これにより、サードパーティベンダーのすぐに使用できるモデルを使用したり、データと環境を完全に制御したりできる柔軟性が得られます。

図 41: Model options for the Migration Portal AI Copilot

サードパーティモデルクラウド

ワークフロー移行ポータルの使用中に、移行ポータル AI コパイロットにプロンプトを送信して、スキーマ変換に関する情報とヘルプを取得できます。AI コパイロットは、インターネットを介して選択した外部のサードパーティクラウドベンダー OpenAI などと通信します。サードパーティベンダーは、独自のモデルを使用してプロンプトを処理し、回答を生成します。

モデルサードパーティベンダーは、チャット完了モデルや埋め込みモデルを含むコア AI 機能を提供します。Azure OpenAI の場合、引き続き埋め込みモデルを展開する必要があります。

利点このアプローチでは、AI インフラストラクチャを管理する必要がなく、非常に強力で継続的に更新されるモデルへのアクセスを提供します。

考慮事項このオプションには、中断のないインターネット接続が必要です。ライセンスによっては、データはベンダーに保存される場合があります。また、チャット完了モデルと埋め込みモデルが必要とする定期的なヘルスチェック要求によって発生するコストとクエリごとの料金を検討します。

セルフホストモデルオンプレミス

ワークフロー移行ポータルの使用中に、移行ポータル AI コパイロットにプロンプトを送信して、スキーマ変換に関する情報とヘルプを取得できます。AI コパイロットは、HM 環境でローカルにホストされているモデルと通信します。これらのモデルは、EDB AI Factory 機能を使用して組織が以前に作成したものです。自己ホストモデルは、プロンプトを処理し、回答を生成します。

モデル EDB AI Factory を使用してこれらのモデルを作成および管理します。これらは、NVIDIA NIM マイクロサービスを使用して提供される Llama 3 のようなモデルです。

利点このアプローチは、安全性の高いエアギャップ環境を必要とする組織に適しています。すべてのデータ処理がインフラストラクチャ内に残り、厳格なコンプライアンスとプライバシー要件を満たします。

考慮事項 AI インフラストラクチャは自分自身が管理します。つまり、組織に AI の専門知識が必要です。また、セルフホストモデルには、サードパーティベンダーが提供するビルトインのコンテンツフィルタリングと安全対策がありません。モデルによって生成された安全でないまたは有害なコンテンツに対する責任および潜在的な法的責任は、ユーザーまたは組織にあります。また、AI ワークロードの実行に必要な高価な GPU ノードのリソース消費も考慮する必要があります。

Enabling a third-party model for the Migration Portal AI Copilot

AI Copilot のインタラクションにサードパーティのクラウドベンダーを活用するために、これらの例では OpenAI および Azure OpenAI を使用します。

前提条件

ベンダーの AI ソリューションを使用して移行ポータル AI コパイロットを展開するために環境が必要なリソースを準備します。

- 特定の AI ベンダーのサービスへのアクセス資格情報が必要です。
    - OpenAI を使用している場合、組織の ID と API キーが必要です。
    - Azure OpenAI を使用している場合、組織のエンドポイント URL と API キーが必要です。また、Azure OpenAI インスタンスで、次の展開を作成します。
        - gpt-4o 展開:
            - 展開名: `gpt-4o`
            - 推奨バージョン: `2024-11-20`
        - text-embedding-3-small 展開:
            - 展開名: `text-embedding-3-small`
            - Suggestedb
            -

AI コパイロットの有効化

Hybrid ManagerHM をインストールする前または後に、AI Copilot を有効にできます。

1. `edb-migration-copilot` が存在するかどうかを確認します。

```
kubectl get namespaces edb-migration-copilot
```

名前空間は、HM のインストール中に作成されます。HM をインストールする前に AI Copilot を有効にする場合、事前に名前空間を作成する必要があります。

2. `edb-migration-copilot` 名前空間がまだ存在しない場合は、作成します。

```
kubectl create ns edb-migration-copilot
```

3. 次の環境変数を設定して、AI コパイロットを既存の AI アカウントにリンクします。プレースホルダーを AI ベンダーのアカウント情報に置き換えます。

```
export OPENAI_ORG_ID=<openai_org_id>   # organisation id for the openai account
export OPENAI_API_KEY=<openai_api_key> # api key for accessing the openai account
```

```
export AZURE_OPENAI_ENDPOINT=<azure_openai_endpoint> # endpoint for the azure openai␣
↪instance
export AZURE_OPENAI_API_KEY=<azure_openai_api_key>   # api key for accessing the␣
↪azure openai instance
```

4. edb-migration-copilot 名前空間にシークレットを追加して、ベンダーの資格情報を構成します。

```
kubectl create secret generic ai-vendor-secrets \
  --namespace=edb-migration-copilot \
  --type=opaque \
  --from-literal=AI_VENDOR=OPENAI \
  --from-literal=OPENAI_ORG_ID="${OPENAI_ORG_ID}" \
  --from-literal=OPENAI_API_KEY="${OPENAI_API_KEY}"
```

```
kubectl create secret generic ai-vendor-secrets \
  --namespace=edb-migration-copilot \
  --type=opaque \
  --from-literal=AI_VENDOR=AZURE_OPENAI \
  --from-literal=AZURE_OPENAI_ENDPOINT="${AZURE_OPENAI_ENDPOINT}" \
  --from-literal=AZURE_OPENAI_API_KEY="${AZURE_OPENAI_API_KEY}"
```

5. 次の Helm 値で migration-portal-values.yaml というファイルを作成して、デフォルトの AI ベンダーシークレットを前の手順で作成したシークレットでオーバーライドします。

```
parameters:
  edb-migration-copilot:
    ai_vendor_secrets: ai-vendor-secrets
```

- ai_vendor_secrets で参照されるシークレットは、以前に作成したシークレットの名前と一致する必要があります。

```
parameters:
  edb-migration-copilot:
    ai_vendor_secrets: ai-vendor-secrets
    chat_model: gpt-4o
```

- ai_vendor_secrets で参照されるシークレットは、以前に作成したシークレットの名前と一致する必要があります。- chat_model で指定された LLM バージョンは、AI コパイロットの推奨バージョンであるため、デフォルトで gpt-4o です。

6. **インストールとセットアップ** 中に helm upgrade コマンドを実行するときにこのファイルを参照するため、移行ポータルのカスタマイズされた値が使用されます。後続の helm upgrade を呼び出す

たびに、このオプションを使用します。それ以外の場合、値はベンダーのデフォルトでオーバーライドされます。

```
helm upgrade \
  -n edbpgai-bootstrap \
  --install \
  ...
  -f migration-portal-values.yaml \
  ...
```

Enabling a self-hosted model for the Migration Portal AI Copilot

セルフホスト AI ファクトリーモデルを使用して、AI コパイロットにサービスを提供できます。この例では、NVIDIA NIM を使用して要求を処理し、Llama 3 を使用して処理し、回答を生成します。

- チャット完了 nvidia/llama-3_3-nemotron-super-49b-v1
- 埋め込み nvidia/llama-3.2-nv-embedqa-1b-v2

!!!警告移行ポータル AI コパイロットでセルフホストモデルを使用する場合、考慮すべき安全性の重大な影響があります。OpenAI amd Azure OpenAI などのサードパーティベンダーが提供するモデルには、モデルが安全でないコンテンツに応答、生成、または寄与するリスクを削減するように設計されたコンテンツフィルタリングおよびその他の安全対策が含まれています。セルフホストモデルを使用する場合、これらの保護は存在しません。

さらに、モデルをホストしているため、不安全な動作に関連するリスクと潜在的な責任に対する責任を負います。

前提条件

セルフホストソリューションを使用して移行ポータル AI コパイロットを展開するために環境が必要なリソースを準備します。

- HM 環境への管理アクセスがあります。
- 組織は、HM *AI Factory Architecture on Hybrid Manager* で *chat completion and a text embeddings model* を作成しました。彼らは、環境変数として設定できる各モデルのエンドポイントを提供しています。

```
export COMPLETIONS_SVC=llama-3-3-nemotron-super-49b-v1
export EMBEDDINGS_SVC=llama-3-2-nv-embedqa-1b-v2

export COMPLETIONS_ENDPOINT=$(kubectl get inferenceservice $COMPLETIONS_SVC -o␣
↪jsonpath={.status.url})
export EMBEDDINGS_ENDPOINT=$(kubectl get inferenceservice $EMBEDDINGS_SVC -o␣
↪jsonpath={.status.url})
```

AI コパイロットの有効化

1. edb-migration-copilot 名前空間が存在するかどうかを確認します。

```
kubectl get namespaces edb-migration-copilot
```

名前空間は、HM のインストール中に作成されます。HM をインストールする前に AI Copilot を有効にする場合、事前に名前空間を作成する必要があります。

1. edb-migration-copilot 名前空間がまだ存在しない場合は、作成します。

```
kubectl create ns edb-migration-copilot
```

1. 次の環境変数を設定して、シークレットをモデルエンドポイントにリンクします。

```
export OPENAI_API_BASE=${COMPLETIONS_ENDPOINT}/v1
export OPENAI_EMBEDDINGS_API_BASE=${EMBEDDINGS_ENDPOINT}/v1
export OPENAI_API_KEY=<openai api key> # set to a placeholder value like `noop` if␣
↪models are deployed in a way that no key is required
```

!!!note AI コパイロットは、OpenAI 互換 API を使用して、セルフホストモデルを含むすべてのモデルと通信します。そのため、クエリを提供するために別のモデルを使用している場合でも、一部の構成パラメーターの名前に openai が含まれます。

1. ai-vendor-secrets シークレットを作成し、モデルのエンドポイントを指すように構成します。

```
kubectl create secret generic ai-vendor-secrets \
    --namespace=edb-migration-copilot \
    --type=opaque \
    --from-literal=AI_VENDOR=NIM \
    --from-literal=RAGCHEW_OPENAI_API_BASE="${OPENAI_API_BASE}" \
    --from-literal=RAGCHEW_OPENAI_EMBEDDINGS_API_BASE="$    {OPENAI_EMBEDDINGS_API_
↪BASE}" \
    --from-literal=OPENAI_API_KEY="${OPENAI_API_KEY}"
```

1. 次の Helm 値で migration-portal-values.yaml というファイルを作成して、デフォルトの AI ベンダーシークレットを前の手順で作成したシークレットでオーバーライドします。

```
parameters:
  edb-migration-copilot:
    ai_vendor_secrets: ai-vendor-secrets
```

1. HM インストールファイルを更新して、AI Copilot 構成を含めます。インストールに使用した YAML 値を更新するか、AI Copilot 構成パラメーターを使用して helm upgrade コマンドを実行します。

この構成ブロックを、Helm で HM のインストールに使用した values.yaml ファイル、または operator で HM のインストールに使用した CRD に追加します。

```
[...]
parameters:
  edb-migration-copilot:
    ai_vendor_secrets: ai-vendor-secrets                    # Allows the default AI␣
↪vendor secrets to be overridden by the secret created above. This  *must*  match␣
↪the name of the above secret.
    chat_model: nvidia/llama-3.3-nemotron-super-49b-v1      # This must match the␣
↪model name as listed by `${COMPLETIONS_ENDPOINT}/v1/models`.
    embeddings_model: nvidia/llama-3.2-nv-embedqa-1b-v2     # This must match the␣
↪model name as listed by `${EMBEDDINGS_ENDPOINT}/v1/models`.
    chat_model_profile: llama3                              # Enables prompt rules␣
↪and other parameters which can help improve the quality of responses from chat␣
↪completion models of the llama3 family.
    embeddings_dimension: "2048"                           # This must match the size␣
↪of the vectors generated by the embeddings model.
    tokenizer: huggingface                                  # Causes tokenizers from␣
↪the Hugging Face Transformers library to be used for token counting.
    tokenizer_model: nvidia/llama-3.3-nemotron-super-49b-v1 # Sets the specific␣
↪tokenizer to use, ensuring it corresponds to a pretrained model on Hugging Face Hub.
    chat_request_max_tokens: "1024"                        # Limits the total number␣
↪of requested tokens to prevent excessively-verbose responses.
    default_similarity_limit: "10"                         # Limits the number of␣
↪contextual chunks included in queries to prevent overwhelming the model with input.
    stream_no_stop_sequence: "true"                        # Prevents OpenAI-specific␣
↪stop sequences being sent to the model. This should always be set when using NIM␣
↪models in order to avoid duplicated final chunks in response streams.
[...]
```

次の構成パラメーターを含めて helm upgrade コマンドを実行します。後続の helm upgrade を呼び出すたびに、これらのパラメーターを含めることを忘れないでください。それ以外の場合、値はデフォルトでオーバーライドされます。

```
helm upgrade \
    -n edbpgai-bootstrap \
    --install \
    [...]
    --set parameters.edb-migration-copilot.ai_vendor_secrets=ai-vendor-secrets \
    --set parameters.edb-migration-copilot.chat_model=nvidia/llama-3.3-nemotron-super-
↪49b-v1 \
    --set parameters.edb-migration-copilot.embeddings_model=nvidia/llama-3.2-nv-
↪embedqa-1b-v2 \
    --set parameters.edb-migration-copilot.chat_model_profile=llama3 \
    --set parameters.edb-migration-copilot.embeddings_dimension="2048" \
    --set parameters.edb-migration-copilot.tokenizer=huggingface \
    --set parameters.edb-migration-copilot.tokenizer_model=nvidia/llama-3.3-nemotron-
↪super-49b-v1 \
```

(次のページに続く)

(前のページからの続き)

```
    --set parameters.edb-migration-copilot.chat_request_max_tokens="1024" \
    --set parameters.edb-migration-copilot.default_similarity_limit="10" \
    --set parameters.edb-migration-copilot.stream_no_stop_sequence="true" \
   [...]
```

1. edb-migration-copilot サービスを再起動して、システムとの新しい値の調整をトリガーします。

```
kubectl rollout restart edb-migration-copilot -n edb-migration-copilot
```

エアギャップインストールの追加構成実験

エアギャップ環境で実行されている場合、移行ポータル AI コパイロットは、Hugging Face Hub から事前トレーニングされたトークナイザーデータを取得しようとすると失敗します。

代わりにトークナイザーデータのローカルスナップショットを使用するように次のパラメーターを設定します。

次の airgapped_mode パラメーターを、以前に文書化した構成ブロック、Helm で HM のインストールに使用した values.yaml ファイル、またはオペレーターで HM のインストールに使用した CRD に含めます。

```
[...]
parameters:
  edb-migration-copilot:
    airgapped_mode: "true" # Forces a local snapshot of tokenizer data to be used␣
↪instead of fetching from Hugging Face Hub.
                           # Only works if `tokenizer` and `tokenizer_model` are set␣
↪to `huggingface` and
                           #`nvidia/llama-3.3-nemotron-super-49b-v1` respectively.
[...]
```

次の airgapped_mode パラメーターを、以前に文書化した helm upgrade コマンドに含めます。後続の helm upgrade を呼び出すときは、すべてのパラメーターを含めることを忘れないでください。それ以外の場合、値はデフォルトでオーバーライドされます。

```
helm upgrade \
    -n edbpgai-bootstrap \
    --install \
    [...]
    --set parameters.edb-migration-copilot.airgapped_mode="true" \
    [...]
```

edb-migration-copilot サービスを再起動します。

!!!重要移行ポータル AI コパイロットには、nvidia/llama-3.3-nemotron-super-49b-v1 事前トレーニング済みトークナイザーのトークナイザーデータのみが同梱されています。tokenizer_model を他のモデルに設定して airgapped_mode: '"true"' を使用すると、移行ポータル AI コパイロットが失敗します。

Using query recommendations in Hybrid Manager

ハイブリッドマネージャーでクエリ推奨事項を使用する

ハイブリッドマネージャーのクエリの推奨事項は、データベースワークロードのパフォーマンスを継続的に向上させるのに役立ちます。

詳細なクエリ診断を実用的な推奨事項と関連付けることにより、ハイブリッドマネージャーでは次のことが可能になります。

- システムの速度を低下させているクエリを特定します。
- これらのクエリを最適化する方法を理解します。
- クエリレイテンシーとリソース消費を削減するアクションを実行します。

クエリのレコメンデーションを使用するのは誰ですか?

クエリの推奨事項は次の場合に役立ちます。

- **DBA およびプラットフォームエンジニア**データベースパフォーマンスの調整。
- **データエンジニア** データパイプラインとバッチジョブを最適化します。
- **開発者** アプリケーションのパフォーマンスのボトルネックを診断する。
- **SRE とオブザーバビリティチーム**環境全体でデータベースの状態をモニタリング。

クエリレコメンデーションとは何ですか?

ハイブリッドマネージャーは、詳細なクエリ診断を収集し、分析します。このプラットフォームは、次のようなワークロードを最適化する機会を特定します。

- データ取得パフォーマンスを向上させるための **インデックスの推奨事項**。
- クエリプランニングと実行の精度を向上させるための **統計の推奨事項** 。

クエリレコメンデーションを使用する理由

クエリの推奨事項は次のことに役立ちます。

- 最小限の手動調整でクエリのパフォーマンスを向上させます。
- 最も高いインパクトを提供するクエリーにチューニング作業を集中します。
- 進化するワークロードに継続的に適応する —クエリパターンとデータ量が変更されると、最適化のニーズも変化します。
- データベースの負荷、インフラストラクチャコスト、およびユーザー側のレイテンシーを削減します。
- 運用上の問題に反応するのではなく、パフォーマンスをプロアクティブに管理します。
- データ駆動の洞察に基づいて、構造化された調整ワークフローを構築します。

クエリの推奨事項を使用する場合

定期的なチューニングとパフォーマンス管理プロセスの一部としてクエリの推奨事項を使用します。

- 新しいアプリケーションの **初期オンボーディング** 中。
- **データベースを移行**新しいプラットフォームまたはバージョンに移行する場合。
- 重要な **スキーマまたはアプリケーションロジックの変更** の後。
- **パフォーマンス低下** または **キャパシティ傾向** を監視する場合。
- **進行中のデータベースの状態レビュー** の一部として —コードを変更せずにクエリパターンは進化します。
- **リリース検証** 中 —新しいワークロードパターンが調整の機会を導入したかどうかを確認します。

クエリレコメンデーションの仕組み

ハイブリッドマネージャーは、次を含むクエリ診断データを収集します。

- クエリーレイテンシーと実行パターン。
- 待機状態の分析
- ワークロードの傾向とクエリ頻度。
- **Query Diagnostics** タブ —ここには、個々のクエリとインラインで推奨事項が表示されます。
- **Recommendations** タブ —選択したデータベースまたはクラスターの現在の推奨事項がすべてリストされます。

問題のあるクエリーの診断から、関連する推奨事項の理解と適用までシームレスに移行できます。

クエリの推奨事項をチューニングプロセスに統合する

ハイブリッドマネージャーの推奨事項をより広範な調整ワークフローに組み込むことができます。

- **毎日または毎週のレビュー** [推奨事項] タブを定期的にチェックして、新しい機会を特定します。
- **ターゲットを絞ったチューニング**クエリ診断でフラグが立てられた高レイテンシーまたは高頻度のクエリに焦点を当てます。インパクトの高い推奨事項欠落しているインデックスなどを適用することから始めます。
- **トレンド分析**推奨事項を適用した後にクエリパフォーマンスがどのように進化するかを監視します。ハイブリッドマネージャーは、調整作業が効果的かどうかを追跡するのに役立ちます。
- **チーム間のコラボレーション**開発者、DBA、およびプラットフォームエンジニアは、同じ推奨事項を使用して、調整作業の優先順位を付け、調整できます。
- **最適化バックログ**未解決の推奨事項を、キャパシティプランニングサイクル中に取り組む調整作業のバックログとして扱います。

ワークフロー クエリから最適化まで

クエリーのレコメンデーションを使用するには

1. クラスターの **Query Diagnostics** タブを開きます。
2. 時間範囲を選択し、クエリパフォーマンスを確認します。
3. 関連する推奨事項を持つクエリを探します。これらはクエリリストに視覚的に示されます。
4. 推奨事項を選択して [**Recommendations**] タブを開き、関連する推奨事項が強調表示されます。
5. 以下を含む推奨事項の詳細を確認します。
   - 選択したクエリーへの影響.
   - メリットのあるその他のクエリー
   - 最適化の適用に関するガイダンス。
6. 推奨事項を適用しますたとえば、インデックスを作成するか、統計を更新します。
7. クエリーを経時的に監視して、パフォーマンスの向上を検証します。

外部クラスターの要件

外部非ハイブリッドマネージャー管理対象クラスターのクエリの推奨事項を表示するには

- **Query Diagnostics* 拡張機能を有効にする必要があります。
- **Query Advisor** 拡張機能を有効にする必要があります。
- ハイブリッドマネージャー **エージェント**がインストールされ、実行されている必要があります。

これらのコンポーネントがないと、ハイブリッドマネージャーは必要なデータを収集して、推奨事項を生成できません。

シナリオ例

次に、クエリの推奨事項が値を提供する一般的な調整シナリオを示します。

- **遅いアプリケーションページ**頻繁に使用され、高いレイテンシーのクエリが **クエリ診断**で特定されます。推奨事項は、フィルタ列にインデックスを追加することをお勧めします。適用されると、応答時間が大幅に向上します。
- **ETL ジョブ最適化**最近のデータの増加により、バッチプロセスの実行が遅くなりました。クエリ診断は、集計クエリのパフォーマンスが低下していることを示しています。統計更新の推奨事項により、プランナーの精度が向上し、ジョブのパフォーマンスが復元されます。
- **キャパシティプランニング**ハードウェアをスケーリングする前に、推奨事項を確認して、コンピューティングを追加せずにチューニングでボトルネックを緩和できるかどうかを確認します。
- **プロアクティブなチューニング**日常的なチューニングサイクル中に、重要なデータベース全体の推奨事項を確認し、改善を体系的に適用します。その結果、高価なクエリーを最適化することにより、データベース全体の負荷が削減されます。

- **リリース検証**新しいアプリケーション機能を展開した後、新しいクエリパターンがチューニングの機会が表面化したかどうかを確認します。推奨事項を早期に適用すると、将来のパフォーマンスの問題が防止されます。

サポートされているレコメンデーションタイプ

ハイブリッドマネージャーは現在以下をサポートしています

- **インデックスの推奨事項** —頻繁に使用される列に不足しているインデックスを追加するなど。
- **統計の推奨事項** —古い統計を更新してプランナーの精度を向上させるなど。

将来のリリースでは、追加の推奨タイプが計画されています。

関連情報

- *Cluster Query Diagnostics*
- *Cluster recommendations*

Troubleshooting Hybrid Manager

Hybrid ManagerHM で一般的な問題をトラブルシューティングできます。

エラーコード

Error Code	Category	Error Message	Explanation	Potential solution
na	Installation	[upm-istio] - failed to install components: [upm-istio]	An error from bootstrap: the LoadBalancer service is pending.	Fix the loadbalancer IAM role first, then delete the LB pod to trigger re-creation.
na	Installation	upm-beaco-ff-base install failed		kubectl edit sc gp2
na	Installation	upm-thanos installation failed		
na	Installation	Failed to pull image		
na	Installation	Loki pods crash	The loki pods crashed and the installation failed.	
na	Portal Login	No healthy upstream	“No healthy upstream” is returned after the login.	
401	Portal Login	401 Access Denied	401 error returned when logging into the portal after the installation.	

次のページに続く

表 32 – 前のページからの続き

Error Code	Category	Error Message	Explanation	Potential solution
500	Portal Login	HTTP Error 500	The back ingress gateway can't access incoming connections.	
na	Database Provisioning	Postgres Cluster Provisioning Stuck at X% Complete		
na	Database Provisioning	CPU and Memory range is much smaller than the nodegroup		
na	Database Update	Disk scale up	The new value is shown on cluster yaml but not applied on pvc/pv.	Edit the storageclass.

エラーの詳細

[upm-istio] - コンポーネントのインストールに失敗しました: [upm-istio]

詳細なエラー

```
5:03:42AM: ---- waiting on 1 changes [4/5 done] ----
5:03:42AM: ongoing: reconcile service/istio-ingressgateway (v1) namespace: istio-
↪system
5:03:42AM:  ^ Load balancer ingress is empty
{"level":"error","msg":"failed to install component: failed to execute kapp command,␣
↪error: Timed out waiting after 15m0s for resources: [service/istio-ingressgateway␣
↪(v1) namespace: istio-system], message: "}
{"level":"error","msg":"Failed to install components: [upm-istio]"}
{"level":"error","msg":"Installation failed, error: failed to install components:␣
↪[upm-istio]"}
```

```
k get svc -n istio-system
```

```
NAME                    TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)
                            AGE
istio-ingressgateway    LoadBalancer   172.20.184.80    <pending>     80:30270/TCP,
↪443:31858/TCP,9443:32692/TCP   3h3m
istiod                  ClusterIP      172.20.146.223   <none>        15010/TCP,15012/
↪TCP,443/TCP,15014/TCP       3h3m
```

```
k describe svc istio-ingressgateway -n istio-system
```

```
Events:
  Type     Reason            Age    From     Message
  ----     ------            ----   ----     -------
  Warning  FailedBuildModel  53m    service  Failed build model due to operation␣
↪error Elastic Load Balancing v2: DescribeLoadBalancers, get identity: get␣
↪credentials: failed to refresh cached credentials, failed to retrieve credentials,
 operation error STS: AssumeRoleWithWebIdentity, https response error StatusCode: 403,
 RequestID: 07cc71c0-3113-4b4e-be7f-3de3b475c49b, api error AccessDenied: Not␣
↪authorized to perform sts:AssumeRoleWithWebIdentity
```

解像度

ロードバランサーの IAM ロールを修正し、LB ポッドを削除します。

```
k delete pod aws-load-balancer-controller-57ccd8bc77-f8lrk -n kube-system
k delete pod aws-load-balancer-controller-57ccd8bc77-j58c2 -n kube-system
k get svc -n istio-system
```

```
NAME                   TYPE           CLUSTER-IP       EXTERNAL-IP
                                                 PORT(S)
       AGE
istio-ingressgateway   LoadBalancer   172.20.184.80    k8s-istiosys-istioing-
↪38d88046aa-37454e78c9648556.elb.us-east-1.amazonaws.com   80:30270/TCP,443:31858/
↪TCP,9443:32692/TCP   3h8m
istiod                 ClusterIP      172.20.146.223   <none>
                                                 15010/TCP,15012/TCP,443/TCP,15014/
↪TCP       3h8m
```

upm-beaco-ff-base のインストールに失敗しました

詳細なエラー

```
☒
NAME                     READY   STATUS    RESTARTS   AGE
app-db-1-initdb-k4nbl    0/1     Pending   0          35m
```

```
kc get pvc -n upm-beaco-ff-base
```

```
NAME       STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS  ␣
↪VOLUMEATTRIBUTESCLASS   AGE
app-db-1   Pending                                                    <unset>
         35m
```

```
kc describe pvc app-db-1 -n upm-beaco-ff-base
```

```
Name:          app-db-1
Namespace:     upm-beaco-ff-base
StorageClass:
Status:        Pending
Volume:
.....
Events:
  Type    Reason         Age                  From                         Message
  ----    ------         ----                 ----                         -------
  Normal  FailedBinding  64s (x142 over 36m)  persistentvolume-controller  no␣
↪persistent volumes available for this claim and no storage class is se
```

解像度

kubectl edit sc gp2 を実行し、次のアノテーションを追加します。storageclass.kubernetes.io/is-default-class: "true"

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
```

Upm-thos のインストールに失敗しました

詳細なエラー

解像度

1. クラッシュループ内のポッドを確認します。
2. ログを確認して、権限関連のエラーを見つけます。

```
kubectl get pod -n monitoring
```

シークレットが正しいかどうかを確認します。

```
kubectl get secret -n monitoring
```

シークレットは、edb-object-storage で定義したものと一致する必要があります。

```
kubectl get secret -n monitoring  thanos-objstore-secret -o yaml
```

```
 apiVersion: v1
data:
  objstore.yml:␣
↪dHlwZTogUzMKY29uZmlnOgogIGJ1Y2tldDogYXBwbGlhbmNlLXNoYW50ZXN0LWVrcy1jbHVzdGVyLWVkYi1wb3N0Z3JlcwogIHJl
```

(次のページに続く)

(前のページからの続き)

```
~ ☒  echo␣
↪dHlwZTogUzMKY29uZmlnOgogIGJ1Y2tldDogYXBwbGlhbmNlLXNoYW50ZXN0LWVrcy1jbHVzdGVyLWVkYi1wb3N0Z3JlcwogIHJl
↪| base64 -d
```

シークレットを修正するには

```
kubectl delete secret thanos-objstore-secret -n monitoring

kubectl delete pod storage-location-operator-controller-manager-xxxxxx -n storage-
↪location-operator
```

Helm アップグレードを再実行し、シークレット thanos-objstore-secret が正しいことを確認します。

```
kubectl delete pod <pod in crashloop> -n monitoring
```

イメージのプルに失敗しました

詳細なエラー

```
NAME                                                READY   STATUS                 ␣
↪RESTARTS   AGE
edbpgai-bootstrap-job-v1.0.6-appl-d12cvm-q2pbg   0/1     Init:AssetPullBackOff   0
      2m49s

Events:
  Type     Reason     Age               From               Message
  ----     ------     ----              ----               -------
  Normal   Scheduled  53s               default-scheduler  Successfully assigned␣
↪edbpgai-bootstrap/edbpgai-bootstrap-job-v1.0.6-appl-d12cvm-q2pbg to i-
↪001f23a316e13e905
  Warning  Failed     22s               kubelet            Failed to pull image
↪"docker.enterprisedb.com/staging_pgai-platform/edbpgai-bootstrap/bootstrap-eks:v1.0.
↪6-appl": rpc error: code = DeadlineExceeded desc = failed to pull and unpack image
↪"docker.enterprisedb.com/staging_pgai-platform/edbpgai-bootstrap/bootstrap-eks:v1.0.
↪6-appl": failed to resolve reference "docker.enterprisedb.com/staging_pgai-platform/
↪edbpgai-bootstrap/bootstrap-eks:v1.0.6-appl": failed to do request: Head "https://
↪docker.enterprisedb.com/v2/staging_pgai-platform/edbpgai-bootstrap/bootstrap-eks/
↪manifests/v1.0.6-appl": dial tcp 18.67.76.32:443: i/o timeout
  Warning  Failed     22s               kubelet            Error: ErrAssetPull
  Normal   BackOff    22s               kubelet            Back-off pulling image
↪"docker.enterprisedb.com/staging_pgai-platform/edbpgai-bootstrap/bootstrap-eks:v1.0.
↪6-appl"
  Warning  Failed     22s               kubelet            Error: AssetPullBackOff
  Normal   Pulling    7s (x2 over 52s)  kubelet            Pulling image "docker.
↪enterprisedb.com/staging_pgai-platform/edbpgai-bootstrap/bootstrap-eks:v1.0.6-appl"
```

解像度

1. AWS コンソールで、関連するサブネットを編集し、**パブリック IPv4 アドレスの自動割り当て** を有効にします。
2. ノードを削除して、EKS に変更をピックアップさせます。

```
k get node
```

```
NAME                  STATUS   ROLES    AGE   VERSION
i-001f23a316e13e905   Ready    <none>   19h   v1.31.4-eks-0f56d01
i-012fd44b2f575a514   Ready    <none>   20h   v1.31.4-eks-0f56d01
i-076d6a447c6551a21   Ready    <none>   20h   v1.31.4-eks-0f56d01
```

```
k delete node i-001f23a316e13e905 i-012fd44b2f575a514 i-076d6a447c6551a21
```

```
node "i-001f23a316e13e905" deleted
node "i-012fd44b2f575a514" deleted
node "i-076d6a447c6551a21" deleted
```

```
kgp -n edbpgai-bootstrap
```

```
NAME                                          READY   STATUS    RESTARTS   AGE
edbpgai-bootstrap-job-v1.0.6-appl-5ee7l7-xzfqt   1/1     Running   0          118s
```

Loki ポッドのクラッシュ

詳細なエラー

```
k get pods -n logging
```

```
NAME                         READY   STATUS             RESTARTS         AGE
loki-backend-0               1/2     CrashLoopBackOff   16 (2m1s ago)    59m
loki-read-5b8555fb64-9srmq   0/1     CrashLoopBackOff   16 (2m40s ago)   59m
loki-read-5b8555fb64-fw6ls   0/1     CrashLoopBackOff   16 (2m7s ago)    59m
loki-read-5b8555fb64-nw2vq   0/1     CrashLoopBackOff   16 (2m17s ago)   59m
loki-write-0                 0/1     CrashLoopBackOff   16 (2m37s ago)   59m
loki-write-1                 0/1     CrashLoopBackOff   16 (2m5s ago)    59m
loki-write-2                 0/1     CrashLoopBackOff   16 (2m16s ago)   59m
```

```
k logs loki-write-0 -n logging
```

```
level=info ts=2025-02-26T09:32:35.173785968Z caller=main.go:126 msg="Starting Loki"␣
↪version="(version=release-3.1.x-89fe788, branch=release-3.1.x, revision=89fe788d)"
level=info ts=2025-02-26T09:32:35.173821129Z caller=main.go:127 msg="Loading␣
```

(次のページに続く)

(前のページからの続き)

```
↪configuration file" filename=/etc/loki/config/config.yaml
level=info ts=2025-02-26T09:32:35.174421237Z caller=server.go:352 msg="server␣
↪listening on addresses" http=[::]:3100 grpc=[::]:9095
level=info ts=2025-02-26T09:32:35.175391511Z caller=memberlist_client.go:435 msg=
↪"Using memberlist cluster label and node name" cluster_label= node=loki-write-0-
↪a0cfbc5d
level=info ts=2025-02-26T09:32:35.17599149Z caller=shipper.go:160 index-store=tsdb-
↪2020-05-15 msg="starting index shipper in WO mode"
level=info ts=2025-02-26T09:32:35.176072171Z caller=table_manager.go:136 index-
↪store=tsdb-2020-05-15 msg="uploading tables"
level=info ts=2025-02-26T09:32:35.176269663Z caller=head_manager.go:308 index-
↪store=tsdb-2020-05-15 component=tsdb-head-manager msg="loaded wals by period"␣
↪groups=0
level=info ts=2025-02-26T09:32:35.176305024Z caller=manager.go:86 index-store=tsdb-
↪2020-05-15 component=tsdb-manager msg="loaded leftover local indices" err=null␣
↪successful=true buckets=0 indices=0 failures=0
level=info ts=2025-02-26T09:32:35.176325994Z caller=head_manager.go:308 index-
↪store=tsdb-2020-05-15 component=tsdb-head-manager msg="loaded wals by period"␣
↪groups=1
level=info ts=2025-02-26T09:32:35.180393332Z caller=module_service.go:82 msg=starting␣
↪module=server
level=info ts=2025-02-26T09:32:35.180477643Z caller=module_service.go:82 msg=starting␣
↪module=memberlist-kv
level=error ts=2025-02-26T09:32:35.180526964Z caller=loki.go:524 msg="module failed"␣
↪module=memberlist-kv error="starting module memberlist-kv: invalid service state:␣
↪Failed, expected: Running, failure: service memberlist_kv failed: failed to create␣
↪memberlist: Failed to get final advertise address: no private IP address found, and␣
↪explicit IP not provided"
level=error ts=2025-02-26T09:32:35.180549544Z caller=loki.go:524 msg="module failed"
 module=ring error="failed to start ring, because it depends on module memberlist-kv,␣
↪which has failed: invalid service state: Failed, expected: Running, failure:␣
↪starting module memberlist-kv: invalid service state: Failed, expected: Running,␣
↪failure: service memberlist_kv failed: failed to create memberlist: Failed to get␣
↪final advertise address: no private IP address found, and explicit IP not provided"
level=error ts=2025-02-26T09:32:35.180582375Z caller=loki.go:524 msg="module failed"␣
↪module=store error="failed to start store, because it depends on module memberlist-
↪kv, which has failed: invalid service state: Failed, expected: Running, failure:␣
↪starting module memberlist-kv: invalid service state: Failed, expected: Running,␣
↪failure: service memberlist_kv failed: failed to create memberlist: Failed to get␣
↪final advertise address: no private IP address found, and explicit IP not provided"
level=error ts=2025-02-26T09:32:35.180588655Z caller=loki.go:524 msg="module failed"␣
↪module=ingester error="failed to start ingester, because it depends on module␣
↪analytics, which has failed: context canceled"
level=error ts=2025-02-26T09:32:35.180593435Z caller=loki.go:524 msg="module failed"␣
↪module=distributor error="failed to start distributor, because it depends on module␣
```

(次のページに続く)

(前のページからの続き)

```
↪analytics, which has failed: context canceled"
level=error ts=2025-02-26T09:32:35.180597415Z caller=loki.go:524 msg="module failed"␣
↪module=analytics error="failed to start analytics, because it depends on module␣
↪memberlist-kv, which has failed: invalid service state: Failed, expected: Running,␣
↪failure: starting module memberlist-kv: invalid service state: Failed, expected:␣
↪Running, failure: service memberlist_kv failed: failed to create memberlist: Failed␣
↪to get final advertise address: no private IP address found, and explicit IP not␣
↪provided"
level=info ts=2025-02-26T09:32:35.180687877Z caller=modules.go:1832 msg="server␣
↪stopped"
level=info ts=2025-02-26T09:32:35.180702097Z caller=module_service.go:120 msg="module␣
↪stopped" module=server
level=info ts=2025-02-26T09:32:35.180708417Z caller=loki.go:508 msg="Loki stopped"␣
↪running_time=85.855531ms
failed services
github.com/grafana/loki/v3/pkg/loki.(*Loki).Run
    /src/loki/pkg/loki/loki.go:566
main.main
    /src/loki/cmd/loki/main.go:129
runtime.main
    /usr/local/go/src/runtime/proc.go:271
runtime.goexit
    /usr/local/go/src/runtime/asm_amd64.s:1695
level=error ts=2025-02-26T09:32:35.180772078Z caller=log.go:216 msg="error running␣
↪loki" err="failed services\ngithub.com/grafana/loki/v3/pkg/loki.(*Loki).Run\n\t/src/
↪loki/pkg/loki/loki.go:566\nmain.main\n\t/src/loki/cmd/loki/main.go:129\nruntime.
↪main\n\t/usr/local/go/src/runtime/proc.go:271\nruntime.goexit\n\t/usr/local/go/src/
↪runtime/asm_amd64.s:1695"
```

解像度

CIDR が 10 ではないため、 Loki からの既知の問題があります。

1. VPC の IPv4 CIDR を確認します。たとえば、11.1.0.0/20 に設定されている場合、10.1.0.0/20 に設定します。

健全な上流はない

詳細なエラー

upm-dex のポッドはクラッシュループしています。ポッドログを確認して、エラーの詳細を表示します。この場合、pgai.portal.authentication.staticPasswords は正しく構成されていないため、正しい構成を更新し、展開を再実行します。

```
upm-dex                       upm-dex-66b96b5857-pdcg2
       1/2     CrashLoopBackOff   206 (3m34s ago)   17h
```

(次のページに続く)

(前のページからの続き)

```
kubectl logs upm-dex-66b96b5857-pdcg2 -n upm-dex
error parse config file /tmp/dex.config.yaml-565347154: error unmarshaling JSON:␣
↪malformed bcrypt hash: crypto/bcrypt: hashedSecret too short to be a bcrypted␣
↪password
```

解像度

upgrade コマンドに 4 パラメーターを追加しました。

```
helm upgrade \
    -n edbpgai-bootstrap \
    --install \
    --set parameters.global.portal_domain_name="${PORTAL_DOMAIN_NAME}" \
    --set parameters.transporter-rw-service.domain_name="${TRANSPORTER_RW_SERVICE_
↪DOMAIN_NAME}" \
    --set parameters.transporter-dp-agent.rw_service_url="https://${TRANSPORTER_RW_
↪SERVICE_DOMAIN_NAME}/transporter" \
    --set parameters.upm-beacon.server_host="${BEACON_SERVICE_DOMAIN_NAME}" \
    --set parameters.upm-beaco-ff-base.cookie_aeskey="${AES_256_KEY}" \
    --set system="eks" \
    --set remoteContainerRegistryURL="${REGISTRY_PACKAGE_URL}" \
    --set internalContainerRegistryURL="${REGISTRY_PACKAGE_URL}" \
    --set bootstrapAsset="${REGISTRY_PACKAGE_URL}/edbpgai-bootstrap/bootstrap-eks:$
↪{EDBPGAI_BOOTSTRAP_IMAGE_VERSION}" \
    --set pgai.portal.authentication.staticPasswords[0].email="owner@mycompany.com" \
    --set pgai.portal.authentication.staticPasswords[0].hash=$2y$10
↪$STTzuGJa3PydqHvlzi2z6OgDU1G/JLTqiuGblH.RemIutWxkztN5m \
    --set pgai.portal.authentication.staticPasswords[0].username="owner@mycompany.com"
 \
    --set pgai.portal.authentication.staticPasswords[0].userID="c5998173-a605-449a-
↪a9a5-4a9c33e26df7" \
    --version "${EDBPGAI_BOOTSTRAP_HELM_CHART_VERSION}" \
    edbpgai-bootstrap enterprisedb-edbpgai/edbpgai-bootstrap
```

401 アクセスが拒否されました

詳細なエラー

解像度

config.yaml ファイルを確認します。この場合、staticPasswords セクションは適切に設定されていません。正しい値を使用して helm upgrade を再実行します。

```
k get secret upm-dex -n upm-dex -o yaml
```

```
apiVersion: v1
data:
  config.yaml:␣
↪aXNzdWVyOiBodHRwczovL3BvcnRhbC5iYXN1cHBvcnQub3JnL2F1dGgKc3RvcmFnZToKICB0eXBlOiBwb3N0Z3JlcwogIGNvbmZp
…

echo "xxx" | base64 -d
staticPasswords:
  - email: owner@mycompany.com
    hash: XXX
    userID: c5998173-a605-449a-a9a5-4a9c33e26df7
    username: owner@mycompany.com
```

別の方法として、次の 1 行コマンドを使用します。

```
k get secret -n upm-dex upm-dex -ojsonpath={.data.config\.yaml} | base64 -d
```

HTTP エラー 500

詳細なエラー

バックイングレスゲートウェイは着信接続にアクセスできません。パブリックネットワークから Web ページにアクセスしている場合、https が許可される唯一のプロトコルであるため、ポート 443 を介して行う必要があります。

解像度

ingressgateway のステータスを確認します。 ingressgateway は istio-ingressgateway です。

```
k get all -n istio-system
```

```
NAME                                    READY   STATUS    RESTARTS   AGE
pod/istio-ingressgateway-7956f6d57c-jd8jm   1/1     Running   0          14m
pod/istio-ingressgateway-7956f6d57c-jmmv8   1/1     Running   0          14m
pod/istio-ingressgateway-7956f6d57c-r65tk   1/1     Running   0          14m
pod/istiod-7f7bbcdcbb-fzggv                 1/1     Running   0          3h26m
pod/istiod-7f7bbcdcbb-mxjpz                 1/1     Running   0          3h26m

NAME                           TYPE           CLUSTER-IP       EXTERNAL-IP
                                                          PORT(S)
               AGE
service/istio-ingressgateway   LoadBalancer   172.20.92.132    k8s-istiosys-istioing-
↪987da680df-8452f790b4602f2c.elb.us-east-1.amazonaws.com   80:30950/TCP,443:32604/
↪TCP,9443:31744/TCP   3h26m
service/istiod                 ClusterIP      172.20.112.158   <none>
                                                          15010/TCP,15012/TCP,443/TCP,
```

(次のページに続く)

(前のページからの続き)

```
↪15014/TCP       3h26m

NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/istio-ingressgateway    3/3     3            3           3h26m
deployment.apps/istiod                  2/2     2            2           3h26m

NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/istio-ingressgateway-68fcd46fff   0         0         0       84m
replicaset.apps/istio-ingressgateway-7956f6d57c   3         3         3       14m
replicaset.apps/istio-ingressgateway-866d9bd74b   0         0         0       3h26m
replicaset.apps/istiod-7f7bbcdcbb                 2         2         2       3h26m
```

ゲートウェイ構成に構成エラーがないかを確認します。ゲートウェイルールは 443 ポートでリッスンするように既に設定されています。

```
k get gateway/upm-portal -n istio-system -oyaml
```

```
apiVersion: networking.istio.io/v1
kind: Gateway
….
spec:
  selector:
    istio: ingressgateway
  servers:
  - hosts:
    - portal.basupport.org
    port:
      name: http
      number: 80
      protocol: HTTP
    tls:
      httpsRedirect: true
  - hosts:
    - portal.basupport.org
    port:
      name: https-443
      number: 443
      protocol: HTTPS
  …
```

次に、展開ログを確認します。

```
2025-02-20T03:42:32.203112Z     warning envoy main external/envoy/source/server/
↪server.cc:835   Usage of the deprecated runtime key overload.global_downstream_max_
↪connections, consider switching to `envoy.resource_monitors.downstream_connections`␣
```

(次のページに続く)

(前のページからの続き)

```
↪instead.This runtime key will be removed in future.      thread=15
2025-02-20T03:42:32.203389Z     warning envoy main external/envoy/source/server/
↪server.cc:928   There is no configured limit to the number of allowed active␣
↪downstream connections. Configure a limit in `envoy.resource_monitors.downstream_
↪connections` resource monitor.   thread=15
2025-02-20T03:42:32.211729Z     info    xdsproxy        connected to delta upstream␣
↪XDS server: istiod.istio-system.svc:15012   id=1
2025-02-20T03:42:32.268369Z     info    ads     ADS: new connection for node:istio-
↪ingressgateway-68fcd46fff-g4nth.istio-system-1
2025-02-20T03:42:32.268437Z     info    cache   returned workload certificate from␣
↪cache        ttl=23h59m59.731565472s
2025-02-20T03:42:32.268890Z     info    ads     SDS: PUSH request for node:istio-
↪ingressgateway-68fcd46fff-g4nth.istio-system resources:1 size:4.0kB resource:default
2025-02-20T03:42:32.279500Z     info    ads     ADS: new connection for node:istio-
↪ingressgateway-68fcd46fff-g4nth.istio-system-2
2025-02-20T03:42:32.279654Z     info    cache   returned workload certificate from␣
↪cache        ttl=23h59m59.72034871s
2025-02-20T03:42:32.279711Z     info    ads     SDS: PUSH request for node:istio-
↪ingressgateway-68fcd46fff-g4nth.istio-system resources:1 size:4.0kB resource:default
2025-02-20T03:42:32.280287Z     info    ads     ADS: new connection for node:istio-
↪ingressgateway-68fcd46fff-g4nth.istio-system-3
2025-02-20T03:42:32.280372Z     info    cache   returned workload trust anchor from␣
↪cache       ttl=23h59m59.719629366s
2025-02-20T03:42:32.280503Z     info    ads     SDS: PUSH request for node:istio-
↪ingressgateway-68fcd46fff-g4nth.istio-system resources:1 size:1.1kB resource:ROOTCA
2025-02-20T03:42:32.320657Z     info    wasm    fetching image staging_pgai-platform/
↪upm-beaco-filters/upm-oidc from registry docker.enterprisedb.com with tag v1.1.44
2025-02-20T03:42:32.323034Z     info    wasm    fetching image staging_pgai-platform/
↪upm-beaco-filters/upm-error-transformer from registry docker.enterprisedb.com with␣
↪tag v1.1.44
2025-02-20T03:42:32.323073Z     info    wasm    fetching image staging_pgai-platform/
↪upm-beaco-filters/upm-authz-checker from registry docker.enterprisedb.com with tag␣
↪v1.1.44
2025-02-20T03:42:35.074684Z     warning envoy wasm external/envoy/source/extensions/
↪common/wasm/context.cc:1198 wasm log: error parsing plugin configuration: Error(
↪"aes_key: Invalid Length, got 3 bytes, expected 32", line: 1, column: 12)  ␣
↪thread=15
2025-02-20T03:42:35.074712Z     error   envoy wasm external/envoy/source/extensions/
↪common/wasm/wasm.cc:110     Wasm VM failed Failed to configure base Wasm plugin    ␣
↪thread=15
2025-02-20T03:42:35.075923Z     critical        envoy wasm external/envoy/source/
↪extensions/common/wasm/wasm.cc:474     Plugin configured to fail closed failed to␣
↪load thread=15
2025-02-20T03:42:35.076559Z     warning envoy config external/envoy/source/extensions/
↪config_subscription/grpc/delta_subscription_state.cc:269  delta config for type.
```

(次のページに続く)

(前のページからの続き)

```
↪googleapis.com/envoy.config.core.v3.TypedExtensionConfig rejected: Unable to create␣
↪Wasm HTTP filter istio-system.upm-oidc        thread=15
2025-02-20T03:42:35.076572Z     warning envoy config external/envoy/source/extensions/
↪config_subscription/grpc/grpc_subscription_impl.cc:138    gRPC config for type.
↪googleapis.com/envoy.config.core.v3.TypedExtensionConfig rejected: Unable to create␣
↪Wasm HTTP filter istio-system.upm-oidc thread=15
2025-02-20T03:42:35.076577Z     warning envoy config external/envoy/source/extensions/
↪config_subscription/grpc/grpc_subscription_impl.cc:138    gRPC config for type.
↪googleapis.com/envoy.config.core.v3.TypedExtensionConfig rejected: Unable to create␣
↪Wasm HTTP filter istio-system.upm-oidc thread=15
2025-02-20T03:42:35.076584Z     warning envoy config external/envoy/source/extensions/
↪config_subscription/grpc/grpc_subscription_impl.cc:138    gRPC config for type.
↪googleapis.com/envoy.config.core.v3.TypedExtensionConfig rejected: Unable to create␣
↪Wasm HTTP filter istio-system.upm-oidc thread=15
2025-02-20T03:42:35.189419Z     info    Readiness succeeded in 3.054667763s
2025-02-20T03:42:35.189690Z     info    Envoy proxy is ready
```

ここでのエラーは、wasm plugin が正常にブートストラップされなかったために発生したため、ingressgateway は実際には準備ができていません。

```
k get pod -listio=ingressgateway -n istio-system
```

```
Enter MFA code for arn:aws:iam::121858151946:mfa/zane.zhou@enterprisedb.com:
NAME                                    READY   STATUS    RESTARTS   AGE
istio-ingressgateway-7956f6d57c-jd8jm   1/1     Running   0          72m
istio-ingressgateway-7956f6d57c-jmmv8   1/1     Running   0          72m
istio-ingressgateway-7956f6d57c-r65tk   1/1     Running   0          72m
```

しかし、ポッドでは、準備状況チェックが実際に失敗したことがわかります。

```
k describe pod/istio-ingressgateway-7956f6d57c-jd8jm -n istio-system
```

```
Warning  Unhealthy  20m (x2 over 20m)  kubelet            Readiness probe failed: Get
↪"http://10.0.30.134:15021/healthz/ready": dial tcp 10.0.30.134:15021: connect:␣
↪connection refused
```

この問題は、wasm plugin の構成の誤りによって発生します。aes_key が正しく設定されていません。構成を修正し、ingressgateway を再展開します。

```
k edit WasmPlugin/upm-oidc
```

次に、次の aes_key をデフォルト値または実際の運用環境でのカスタム生成値に設定します。

```
pluginConfig:
  aes_key: rzkutHl8NJNztPMEJYykZouHslNiA7xmIXH+58ISUVo=
```

次に、展開をリスタートします。

```
k rollout restart deployment/istio-ingressgateway -n istio-system
```

Postgres クラスタープロビジョニングが X%完了で停止する

詳細なエラー

単一ノードクラスターの作成が 87%完了で停止しました。

解像度

CPU とメモリの範囲はノードグループよりもかなり小さいです

詳細なエラー

ノードグループは m5.4xlarge で、これは 16vCPU と 64GB RAM、ノードなしです。クラスターの作成時に、コンソールに表示される範囲は CPU 0〜3.92 コアとメモリ 0〜14.3 Gi です。

解像度

これは、サイズ 0 のノードプールによるものです。ノードプールにノードがある場合、問題ありません。しかし、問題が発生したときにノードを作成すると、機能しません。ノードが作成された後も、コンソールには同じ値が表示されます。

保留中 https://enterprisedb.atlassian.net/browse/UPM-45883

ディスクのスケールアップ

詳細なエラー

コンソールのスケールディスク。新しい値はクラスター yaml に表示されますが、pvc / pv には適用されません。

```
kubectl get cluster p-2hgee2782y -o yaml
```

```
storage:
  resizeInUseVolumes: true
  size: 5Gi
```

```
k get pv
```

```
pvc-4a35b3ba-507e-4d4c-9017-360326e57b9d   2Gi    RWO            Delete          ␣
↪Bound    p-2hgee2782y/p-2hgee2782y-1
```

解像度

CNP オペレータログには次のエラーが含まれています。

```
kubectl logs postgresql-operator-controller-manager-7cc67597df-4ggdf -n postgresql-
↪operator-system
```


```
{
  "level": "error",
  "ts": "2025-01-17T10:02:35.739693483Z",
  "msg": "Reconciler error",
  "controller": "cluster",
  "controllerGroup": "postgresql.k8s.enterprisedb.io",
  "controllerKind": "Cluster",
  "Cluster": {
    "name": "p-2hgee2782y",
    "namespace": "p-2hgee2782y"
  },
  "namespace": "p-2hgee2782y",
  "name": "p-2hgee2782y",
  "reconcileID": "2de2ade7-2bf7-45f4-a2af-3f135a4f7ad9",
  "error": "persistentvolumeclaims \"p-2hgee2782y-1\" is forbidden: only dynamically␣
↪provisioned pvc can be resized and the storageclass that provisions the pvc must␣
↪support resize",
  "stacktrace": "sigs.k8s.io/controller-runtime/pkg/internal/controller.(␣
↪*Controller[...]).reconcileHandler\n\tcloud-native-postgres/vendor/sigs.k8s.io/
↪controller-runtime/pkg/internal/controller/controller.go:316\nsigs.k8s.io/
↪controller-runtime/pkg/internal/controller.(* Controller[...]).processNextWorkItem\
↪n\tcloud-native-postgres/vendor/sigs.k8s.io/controller-runtime/pkg/internal/
↪controller/controller.go:263\nsigs.k8s.io/controller-runtime/pkg/internal/
↪controller.(*Controller[...]).Start.func2.2\n\tcloud-native-postgres/vendor/sigs.
↪k8s.io/controller-runtime/pkg/internal/controller/controller.go:224"
}
```


storageclass ～allowVolumeExpansion: true を確認・編集します。

```
kubecl edit storageclass gp2
```

```
k get storageclass gp2 -o yaml
```

```
allowVolumeExpansion: true
```

CNP オペレーターポッドを削除して、変更を適用します。

```
kubectl  delete pod postgresql-operator-controller-manager-7cc67597df-tgl7b -n␣
↪postgresql-operator-system
```

```
kubectl get pv
```

```
pvc-4a35b3ba-507e-4d4c-9017-360326e57b9d   5Gi        RWO            Delete           ␣
↪Bound    p-2hgee2782y/p-2hgee2782y-1
```

stats-collector-db ディスクの問題

ほとんどの場合、 stats-collector-db でディスクの問題が発生している場合、次のいずれかが発生しています。

1. ディスクが容量に近づいていることを示すアラートマネージャーのアラート。これは、ディスクがまだいっぱいになっていないため、クラスターがまだ実行されている場合に発生します。
2. ポッドがクラッシュループにあることを示すアラートマネージャーのアラート。これは、ディスクがいっぱいであるためにクラスターがクラッシュした場合に発生します。また、クラスターの**モニタリング** タブでは、関連するディスク統計を取得できません。

シナリオ 1 クラスターの実行中にディスクの使用率が高くなる

クラスターは正常に動作しているが、ディスク使用量が高いことに気付いた場合、2 つのオプションがあります。

ソリューション A データ保持期間を短縮する

データ保持期間を短縮することは、保存するデータの量を減らして、ディスク使用量を削減する簡単な方法です。デフォルトの保持期間は 168 時間 7 日です。このソリューションを使用して、保持期間を 96 時間 4 日に短縮します。

保持期間を短縮すると、利用可能なディスクが約 42%増加します。リテンション率を低下させるには

1. プラットフォームへの kubectl アクセスを持つ端末から次のコマンドを使用して、ConfigMap を編集します。

```
kubectl -n upm-api-stats-collector edit cmd
upm-api-stats-collector-configmap-<your-config-map-string>
```

1. ConfigMap で、retention-period パラメーターを見つけ、その値を 168h から 96h に変更します。
2. upm-api-stats-collector を再起動します。

```
kubectl rollout restart deploy/upm-api-stats-collector -n upm-api-stats-collector
```

これらの手順を完了すると、DB の使用量は安定した範囲内に留まり、Alert Manager のアラートは消えます。

解決策 B ディスクボリュームサイズを増やす

7 日間のデータを保持する必要がある場合、問題を解決するには、ディスクのストレージ容量を増加する必要があります。

1. クラスター構成を編集します。

```
kubectl edit cluster stats-collector-db -n upm-api-stats-collector
```

1. stats-collector-db 仕様を見つけて、 .spec.storage.size の値を優先するサイズに増やします。

2. upm-api-stats-collector を再起動します。

```
kubectl rollout restart deploy/upm-api-stats-collector -n upm-api-stats-collector
```

シナリオ 2 ディスクがいっぱいで、クラスターがクラッシュする

ディスクがいっぱいである場合、クラスターがクラッシュループに陥ってしまう可能性があります。例

```
upm-api-stats-collector      stats-collector-db-1 0/1     CrashLoopBackOff   197␣
↪(4m34s ago)   4d7h
```

この問題に対処するには 2 つのオプションがあります。

解決策 A データベースを再作成します

履歴統計データを保持する必要がない場合

1. stats-collector-db クラスターを削除します。

```
kubectl delete cluster stats-collector-db -n upm-api-stats-collector
```

1. S3 バケットから stats-collector-db バックアップフォルダーを削除します。

```
S3/edb-internal-backups/<your-S3-string>/databases/transporter-db/
```

1. helm upgrade を実行して、デフォルト設定で空のデータベースを再作成します。

```
helm upgrade -n edbpgai-bootstrap \
--install -f ./values.yaml \
--version "${EDB_PLATFORM_VERSION}" \
edbpgai-bootstrap "${HCP_HELM_REPO_NAME}/edbpgai-bootstrap
```

1. upm-api-stats-collector を再起動します。

```
kubectl rollout restart deploy/upm-api-stats-collector -n upm-api-stats-collector
```

解決策 B ディスクボリュームサイズを増やす

既存のデータを保持する必要がある場合は、ディスクのボリュームサイズを増やす必要があります。

1. クラスター構成を編集します。

```
kubectl edit cluster stats-collector-db -n upm-api-stats-collector
```

1. stats-collector-db 仕様を見つけて、 .spec.storage.size の値を増加します。

2. クラッシュループ内のポッドを削除します。

3. `ump-api-stats-collector` を再起動します。

```
kubectl rollout restart deploy/upm-api-stats-collector -n upm-api-stats-collector
```

再起動後、データベースのボリュームが大きくなり、クラッシュループが解決されます。

> **警告**
>
> ボリュームサイズをデフォルトより大きくすると、バージョン`1.2.x` を実行しているクラスターのパッチバージョンのアップグレードがブロックされます。

ディスクボリュームサイズを増やしたものの、HM v1.2.x でマイナーバージョンのアップグレードを実行する必要がある場合は、最初にデータベースのサイズをデフォルトのサイズ 50GB に戻す必要があります。データベースのサイズ変更については、サポートチームに連絡してください。!!!

第7章　AI Factory in Hybrid Manager

- /purl/hm/ai-factory ナビゲーション
- “#はじめに”
- はじめに
- アーキテクチャ
- sovereign-ai-on-hm
- ユースケースとペルソナ
- よくある質問
- GPU 推奨
- クイックスタート
- “#マニュアル”
- how-to-enable-ai-factory
- gen-ai
- モデル
- “#セットアップ”
- 前提条件
- deploy-with-hm-ui
- 高度な展開
- “#ハウツー (使用モデル)”
- モデルクラスター内のモデルを使用して
- “#How-to Gen AI ビルダーを使用する”
- using-models-in-gen-ai-builder
- 「#操作＆観察」
- 可観測性
- トラブルシューティング Hybrid Manager の **AI Factory ワークロード**は、スケーラブルな AI、機械学習、および Gen AI 機能を Hybrid Manager プラットフォームにもたらします。Postgres、ベクトル検索、モデルサービング、AI アシスタントにわたって緊密に統合されているため、ハイブリッドマネージャーが管理するクラスターとデータ全体で AI を運用できます。

ハイブリッドマネージャーの AI Factory を使用すると、次のことができます。

- 内部または外部向けに Gen AI アシスタントとエージェントを展開する
- 統合された KServe ベースの推論と GPU アクセラレーションを使用して AI モデルを大規模に提供
- ナレッジベースを作成し、ベクトル検索を使用して検索拡張生成 RAG を実行します
- モデルライブラリを管理し、ハイブリッドマネージャーガバナンスの下で信頼できるモデルを展開します
- AI 機能をアプリケーションとデータパイプラインに直接統合します

ハブクイックリンク *AI Factory Hub — Gen AI Builder on Hybrid Manager —* **モデルサービング**

詳細なアーキテクチャとコア概念については、*AI Factory Hub* を参照してください。

7.1 構築できる AI ソリューションの例

ハイブリッドマネージャーの AI ファクトリーは、多くのドメインにわたるソリューションを有効にします。

- エンタープライズ検索およびナレッジアシスタント企業文書、データベース、イントラネットコンテンツと統合する RAG ベースのアシスタントを構築します。
- カスタマーサポートチャットボット独自のデータとドメイン固有のモデルを活用したアシスタントを展開し、会話型 AI とナレッジベースを組み合わせます。
- AI 駆動型データアプリ アプリの AI 搭載エンドポイント - セマンティック検索、レコメンデーション、類似性検索、または自然言語クエリーを公開します。
- 内部ツールのためのオペレーショナル AI DevOps、カスタマーサクセス、HR オートメーション、セールスイネーブルメントなどを支援する AI エージェントとツールを構築します。
- ドメイン固有のモデル提供独自のまたは微調整されたモデル LLM、埋め込みモデル、ランキングモデルを、ハイブリッドマネージャークラスター内のスケーラブルな推論サービスとして提供します。

単一のアシスタントまたはモデルエンドポイントでスモールスタートし、AI Factory アーキテクチャを使用して完全な AI 搭載アプリケーションにスケーリングできます。

7.2 ラーニングパス

構造化された学習パスに従って、ハイブリッドマネージャーの AI ファクトリーをマスターします。

- *AI Factory 101* — 初歩の概念と使用法
- *AI Factory 201* — Gen AI および AI Factory のワークロードの構築と管理
- *AI Factory 301* — 高度な統合、スケーリング、ガバナンス、および最適化

7.3 ユースケースとソリューション

AI Factory を使用してソリューションを構築するのに役立つ詳細なガイダンスとパターンを提供します。

- *Common workloads and use cases* — AI Factory 搭載アプリケーションの実証済みパターン
- *Industry Solutions* — 業界固有の例とベストプラクティス

ハイブリッドマネージャーワークロードの## AI Factory

ハイブリッドマネージャーは、AI ファクトリー機能のフルセットをサポートし、コントロールプレーンに緊密に統合されています。

7.3.1 Gen AI ワークロード

- *Gen AI in Hybrid Manager* — ハイブリッドマネージャーの Gen AI 機能
- *Agent Studio & Builder (consolidated)* —アシスタント、ツール、構造、ルールセットを作成します。ナレッジベースの管理

7.3.2 モデルの管理と提供

- モデルライブラリーアーキテクチャ — モデルアセットを管理および統治する
- モデルサービング — Kubernetes でのモデル推論サービスの展開とスケーリング
- *GPU resource management* — サービング用の GPU 容量を構成および割り当てます

7.3.3 ベクトルエンジン

- ベクトルエンジン — Postgres を使用した統合ベクトル検索および類似性機能

7.4 Hybrid Manager の学習コンテンツ

ハブの AI Factory コンテンツに加えて、ハイブリッドマネージャーは、ハイブリッドマネージャー固有の使用のための追加の学習コンテンツを提供します。

- *Concepts and architecture*
- *How-To Guides*
- ラーニングパス

7.5 AI Factory がハイブリッドマネージャーにどのように適合するか

Hybrid Manager は、データベース、分析、AI ワークロードのコントロールプレーンです。AI Factory は、他の HM 機能を補完します。

- データベース Postgres クラスターをプロビジョニングおよび管理します。 AI Factory は、ベクター検索、RAG、およびアプリケーションエンドポイントのためにこれらに接続します。

- 分析 データを Iceberg/Delta にオフロードし、 SQL 経由でクエリします。 AI Factory のパイプラインとベクトルエンジンは、取得のためにこれらのデータセットと統合します。
- オペレーション HM の可観測性とガバナンスを使用して、データベースと分析とともに Gen AI、パイプライン、およびモデル提供をモニタします。

HM の AI ファクトリーの前提条件

- AI Factory が対応し、Gen AI、パイプライン、ベクトルエンジン、モデルサービングの資格が必要なハイブリッドマネージャープロジェクト。
- 提供に必要な GPU リソースへのアクセス。*GPU resource management* を参照してください。

7.6 はじめる

ハイブリッドマネージャーで AI ファクトリーを使用してビルドを開始するには

- *AI Factory learning paths* をフォローする
- *Common workloads and use cases* および *industry solutions* の探索
- *How-To Guides* を読む
- *Gen AI in HM* または **モデルサービング** を使用して、最初の Gen AI アシスタントまたはモデルを展開する

Hybrid Manager の AI Factory は、ハイブリッドデータ資産全体で AI を運用化するためのスケーラブルで安全なプラットフォームを提供します。

7.6.1 Getting Started

Getting started (AI Factory on HM)

ハブクイックリンク *AI Factory Hub* — *Gen AI Builder on Hybrid Manager* — **モデルサービング**

Hybrid ManagerHM は、管理された Kubernetes ネイティブプロジェクトで AI Factory ワークロードを実行します。このページを使用して、HM コンテキストを含む正規ハブドキュメントにジャンプします。

HM でできること

- ビルドアシスタントとエージェント *Gen AI Builder on Hybrid Manager* を参照してください
- KServe でモデルを提供する *Model Serving (hub)* を参照してください
- ナレッジベースの作成 *Knowledge Base (hub)* を参照してください

HM の次のステップ

- プロジェクトで AI ファクトリーを有効にする — TODO: HM 有効化ランブックへのリンク
- 必要に応じて、GPU ノードグループをプロビジョニングします —*Setup GPU* を参照してください
- 最初のアシスタントを作成します — *Create Assistant* を参照してください

TODO

- プロジェクト作成とワークロードの有効化のための HM UI スクリーンショットを追加
- 内部イネーブルメントガイドリンクの確認

AI Factory Architecture on Hybrid Manager

アーキテクチャの概要

AI Factory は、Hybrid Manager の Kubernetes インフラストラクチャ内のコンテナ化されたサービスのコレクションとして展開し、統合された **モデルガバナンス**、*inference serving* 、および *Gen AI Builder on Hybrid Manager* コンポーネントを介して *Sovereign AI on Hybrid Manager* 機能を提供します。このアーキテクチャは、ローカルの *GPU resources* とオブジェクトストレージを活用して、顧客制御の Kubernetes クラスター内ですべての AI ワークロードを処理することにより、完全なデータ主権を保証します。

システムは、ガバナンスとオーケストレーションのためのコントロールプレーン、モデルサービングとアプリケーション実行のランタイムレイヤー、モデルアーティファクトと *RAG* **のナレッジベースを作成するにはどうすればよいですか?** のストレージレイヤーの 3 つのアーキテクチャレイヤーで動作します。これらのレイヤーは、Kubernetes API とカスタムリソースを介して統合され、プロジェクトとワークロード間の分離を維持しながら統合管理を提供します。

コアコンポーネント

モデルライブラリーアーキテクチャ

モデルライブラリーアーキテクチャ は、プラットフォーム全体でモデルのライフサイクルとガバナンスを管理するコントロールプレーンサービスとして動作します。このサービスは、承認されたモデルの集中レジストリを維持し、モデルが運用環境に到達する前にセキュリティとコンプライアンスポリシーを適用します。

ライブラリは、いくつかの相互接続されたサービスで構成されています。

- 外部コンテナレジストリを監視するレジストリ同期サービス
- 組織ガバナンスルールに対してモデルを評価するポリシーエンジン
- モデルバージョン、パフォーマンスベンチマーク、および承認を追跡するメタデータサービス
- オブジェクトストレージバックエンドでモデルアーティファクトを管理するストレージインターフェイス

モデルのメタデータは、ハイブリッドマネージャーによって管理される PostgreSQL データベースに持続し、他のプラットフォームデータとの一貫性を保証します。このライブラリは、Kubernetes カスタムリソースを介してモデルをプロジェクト名前空間に公開し、一元的なガバナンスを維持しながら宣言的モデルの展開を可能にします。*Model Library explained* も参照してください。

推論サーバーインフラストラクチャ

推論サーバーは、プロジェクト名前空間内の KServe *InferenceServices* として展開され、特殊なコンテナポッドを介してスケーラブルな **モデルサービング** を提供します。これらのポッドは、さまざまなフレームワークとハードウェア構成に最適化されたモデルランタイムエンジンをカプセル化します。

推論ポッド構成には次のものが含まれます。

- モデルランタイムコンテナ
- GPU 割り当て、メモリ制限、および CPU 要件を定義するリソース仕様 *Setup GPU* および *Update GPU resources* を参照してください。
- モデルストレージと構成データに接続するボリュームマウント
- エンドポイント構成とランタイムパラメーターを含む環境変数
- liveness および readness プローブのヘルスチェック定義

自動スケーリング構成は、要求レイテンシー、GPU 使用率、キューの深さなどのメトリックに応答し、パフォーマンス目標を満たしながら最適なリソース使用率を保証します。デプロイメントオプションについては、**モデルの展開** および *Configure ServingRuntime* を参照してください。

Gen AI Builder ランタイム

Gen AI Builder on Hybrid Manager は、AI アプリケーションのビジュアル開発とランタイム実行を提供する多層アプリケーションとして展開されます。このアーキテクチャは、ユーザーインターフェイス、オーケストレーションロジック、および実行環境間の関心を分離します。

ビルダーランタイムには以下が含まれます。

- ビジュアル開発環境を提供する Web インターフェイスポッド
- エージェントとツールの実行を調整するオーケストレーションサービスポッド
- LLM 接続を使用して分離された AI ワークフローを実行するエージェント実行ポッド
- データアクセスと統合のための再利用可能な機能を提供するツールサービスポッド**データを外部ツールに接続する** を参照してください
- 埋め込み用の *pgvector and Vector Engine concepts* を使用した PostgreSQL を介した状態管理

各コンポーネントは特定のリソース割り当てとセキュリティコンテキストで実行され、ユーザーのワークロード間の分離を保証しながら、サービスメッシュポリシーを介した制御された通信を有効にします。オーケストレーションレイヤーは、ワークフローの実行を管理し、会話状態を維持し、LLM 呼び出し、ツール呼び出し、データの取得の間を調整します。*Gen AI Builder* **アシスタント** *&* **スレッド** 、 *Rulesets* を参照してください。

インフラストラクチャの統合

Kubernetes リソース管理

AI Factory は、Kubernetes リソースプリミティブを活用して、ワークロード全体での予測可能なパフォーマンスと公平なリソース割り当てを保証します。リソース管理は、名前空間のクォータ、ポッド仕様、および優先度クラスを介して複数のレベルで発生します。

リソース割り当て戦略には次のものが含まれます。

- 名前空間レベルのクォータにより、プロジェクトごとの GPU とメモリの合計消費量を制限
- クリティカルなワークロードの最小保証リソースを保証するポッドリソース要求
- 個々のワークロードがクラスターリソースを独占しないようにするリソース制限

- プロダクション推論が優先スケジューリングを確実に受け取る優先度クラス
- クラスター操作中にサービスの可用性を維持するポッドの中断バジェット

スケジューラーは、ポッドを配置するときに GPU 要件を考慮し、ノードセレクターとアフィニティルールを使用して、ポッドが適切に装備されたノードに着地するようにします。テイントとトレランスは、非 GPU ワークロードが GPU 対応ノードを消費するのを防ぎ、AI ワークロードの可用性を最大化します。完全なセットアップガイドについては、*Setup GPU* を参照してください。

GPU インフラストラクチャ

GPU リソースは、NVIDIA デバイスプラグインとコンテナランタイムを介して統合され、コンテナ化されたワークロードからのネイティブ GPU アクセスが有効になります。このインフラストラクチャは、シングル GPU 開発インスタンスからマルチ GPU 運用展開までのさまざまな GPU 構成をサポートします。

GPU 管理機能には次のものが含まれます。

- デバイスプラグインの検出と使用可能な GPU リソースのアドバタイズメント
- ポッドからの CUDA アクセスを有効にするコンテナランタイム構成
- 小規模なモデル間で大規模な GPU をパーティション化するためのマルチインスタンス GPU のサポート
- 開発およびテストワークロードのタイムスライシング構成
- 機能に基づいた自動ノードラベル付けのための GPU 機能検出

リソースの割り当てでは、ワークロードをスケジュールするときに GPU メモリ要件、CUDA のコンピューティング機能、および相互接続トポロジを考慮します。運用展開は通常、専用の GPU 割り当てを受け取りますが、開発ワークロードは、タイムスライシングまたは MIG パーティションを介して GPU を共有する場合があります。

ストレージアーキテクチャ

AI Factory は、モデルアーティファクト、データセット、ナレッジベースにオブジェクトストレージを利用し、プライマリストレージバックエンドとして機能する MinIO またはクラウドプロバイダーサービス S3、Azure Blob、GCS を使用します。このアーキテクチャは、コンピューティングをストレージから分離し、独立したスケーリングとコストの最適化を可能にします。

ストレージ統合パターンには次のものが含まれます。

- ロードパフォーマンスのために最適化された圧縮形式を使用したモデルアーティファクトストレージ
- 効率的な並列処理のためのパーティショニング戦略を使用したデータセットストレージ
- 類似性検索操作用に最適化されたベクトル埋め込みストレージ
- トレーニングの再開とモデルのバージョン管理を可能にするチェックポイントストレージ
- キャッシュレイヤーにより、頻繁にアクセスされるモデルの繰り返しダウンロードが削減されます

ストレージアクセスは、サービスアカウントの資格情報またはクラウドプロバイダーの ID メカニズムを介した認証を備えた標準化された S3 API を介して発生します。永続的なボリューム要求は、頻繁にアクセス

されるモデルのローカルキャッシュを提供し、ネットワークオーバーヘッドを削減し、推論レイテンシーを改善します。

ネットワークアーキテクチャ

サービスコミュニケーション

内部サービス通信は、サービス間呼び出しに安定したエンドポイントを提供する DNS 解決を使用した Kubernetes サービス検出を介して発生します。サービスメッシュは、アプリケーションの変更を必要とせずにセキュリティレイヤーと可観測性レイヤーを追加します。

通信パターンには次のものが含まれます。

- クラスターローカル DNS 名を使用したサービス間の呼び出し
- サービスエンドポイントを使用した複数のポッドレプリカ間のロードバランシング
- サービス低下中のカスケード障害を防止するサーキットブレーク
- 一時的な障害の指数関数的バックオフを使用した再試行メカニズム
- 無期限の要求ブロックを防ぐタイムアウト構成

ネットワークポリシーは、通信境界を適用し、名前空間間のトラフィックフローを制限し、不正なサービスアクセスを防止します。これらのポリシーは、ゼロトラストネットワーキング原則を実装し、すべてのサービス間通信に明示的な承認を必要とします。

外部アクセス

AI サービスへの外部アクセスは、認証とレート制限を備えた制御されたイングレスポイントを介して発生します。複数のアクセスパターンは、セキュリティ境界を維持しながら、さまざまなクライアント要件をサポートします。

アクセスメカニズムには次のものが含まれます。

- TLS を終端するイングレスコントローラとバックエンドサービスへのルーティング
- 認証、承認、およびレート制限を提供する API ゲートウェイ
- 詳細なトラフィック管理を可能にするサービスメッシュゲートウェイ
- 使用可能なインスタンス全体にトラフィックを分散するロードバランサー
- ストリーミング推論応答の WebSocket サポート

認証は OAuth2 / OIDC プロトコルを介してエンタープライズ ID プロバイダーと統合され、API キーはサービスアカウントへのプログラムによるアクセスを提供します。レート制限は、クライアント全体の公平なアクセスを保証しながら、リソースの枯渇を防ぎます。

高可用性の考慮事項

コンポーネントの冗長性

重要なサービスは冗長性を持って展開され、障害またはメンテナンス操作中の可用性が保証されます。冗長戦略は、コンポーネントのステートフル性とパフォーマンス要件によって異なります。

可用性パターンには次のものが含まれます。

- ステートレス推論サーバーのマルチレプリカ展開
- ステートフルオーケストレーションサービスのアクティブ/パッシブ構成
- シングルライターセマンティクスを必要とするコンポーネントのリーダー選択メカニズム
- 該当する場合、アベイラビリティーゾーン全体の地理的分散
- アップグレード中にサービスの可用性を維持するローリングアップデート戦略

ヘルスモニタリングは、コンポーネントの障害を検出し、自動復旧手順をトリガーします。Liveness プローブは、異常なコンテナを再起動しますが、readness プローブは、初期化中のポッドへのトラフィックルーティングを防止します。スケジューラーは、障害が発生したノードのポッドを自動的に置き換え、目的のレプリカ数を維持します。

データの耐久性

データの耐久性は、基になるストレージシステムの保証に依存し、重要なデータのアプリケーションレベルの保護を追加します。オブジェクトストレージはモデルアーティファクトとデータセットの高い耐久性を提供し、PostgreSQL レプリケーションはメタデータの可用性を保証します。

耐久性メカニズムには次のものが含まれます。

- 複数のアベイラビリティーゾーンにわたるオブジェクトストレージレプリケーション
- メタデータデータベースの PostgreSQL ストリーミングレプリケーション
- 構成および状態データのバックアップ手順
- モデルアーティファクトとアプリケーションコードのバージョン管理
- 致命的な障害のディザスターリカバリー手順

オペレーションモニタリング

可観測性スタック

このプラットフォームは、すべての AI コンポーネントから収集されたメトリック、ログ、およびトレースを介して包括的な可観測性を提供します。この可観測性により、プロアクティブな問題検出とパフォーマンスの最適化が可能になります。

モニタリング機能には次のものが含まれます。

- 推論サーバーおよびアプリケーションポッドからの Prometheus メトリック収集 *Monitor InferenceService* を参照してください
- システムの状態とパフォーマンスの傾向を視覚化する Grafana ダッシュボード
- コンテナログとアプリケーション出力を集約した一元化ロギング
- サービス全体にわたって要求フローをキャプチャする分散トレース
- 異常な条件の通知をトリガーするアラートルール

カスタムメトリックは、推論レイテンシー、トークン生成レート、GPU メモリ使用量、モデル精度ドリフトなどの AI 固有の指標を追跡します。これらのメトリックは、運用モニタリングとキャパシティプランニングの決定の両方をサポートします。

パフォーマンス分析

パフォーマンスモニタリングは、ユーザーエクスペリエンスとリソース効率に影響を与える主要な指標に焦点を当てています。メトリック収集は、システム全体の集計から個々の要求のトレースまで、複数の粒度で発生します。

パフォーマンス指標には次のものが含まれます。

- クライアントから応答までのエンドツーエンドの要求レイテンシ
- ネットワーク遅延とキューイング遅延を除くモデルの推論時間
- リソース効率を示す GPU 使用率
- 最適化の機会を特定するメモリ消費パターン
- キャパシティの制約を明らかにするキューの深さ

分析ツールは、レイヤー全体でメトリックを相関させ、ボトルネックと最適化の機会を特定します。この分析は、スケーリングの決定、リソース割り当ての調整、およびアーキテクチャの改善に役立ちます。

Sovereign AI on Hybrid Manager

ハブクイックリンク *Sovereign AI (hub) — AI Factory Hub*

ハイブリッドマネージャーにおけるソブリン AI が意味するもの

ハイブリッドマネージャー上の AI ファクトリーを使用すると、AI ワークロードは**データと同じ場所にあります** 。

- **ご使用の環境**アシスタント、モデル提供、およびベクトル検索は、Hybrid Manager プロジェクトと Kubernetes クラスター内で直接実行されます。
- **あなたのデータ。**ソースドキュメント、埋め込み、および会話履歴は、明示的に構成しない限り、環境を離れることはありません。
- **あなたのモデル。**モデルイメージはモデルライブラリーでキュレートされ、クラスター内のエンドポイントから提供されます。
- **あなたの可視性。**会話スレッドとモデルログは、Postgres と分析に使用するのと同じプラットフォームを介して観察できます。

接続オプション

- **標準接続済み。**ほとんどの環境はインターネット接続で実行され、コンテナイメージ、更新、またはライブラリをプルします。接続しても、**データはプロジェクト内に残ります** - SaaS AI とは異なり、プロバイダーと共有されません。

- **エアギャップ**高セキュリティな展開の場合、インターネットアクセスなしでハイブリッドマネージャーを操作できます。モデルイメージとレジストリをプリロードすると、すべてのワークロードが完全に制御されたクラスター内に残ります。

外部 AI サービスとの対比

- **外部 LLM サービス** 例、OpenAI、Anthropic、Cohere プロンプト、ドキュメント、および埋め込みはネットワークを離れ、サードパーティ環境で処理されます。
- **ハイブリッドマネージャー**すべての取得、生成、およびストレージは、プロジェクトとインフラストラクチャ内で発生します。モデルはデータの横に提供されるため、機密コンテンツがあなたの制御を離れることはありません。エアギャップ展開では、外部接続が必要ないことがさらに保証されます。

詳細はこちら

- *Sovereign AI (hub)* — 原則とパターン。
- *AI Factory Hub* —アシスタント、ナレッジベース、モデル提供に関するより深いガイド。

Use Cases and Personas for AI Factory on Hybrid Manager

概要

ハイブリッドマネージャーの AI ファクトリーは、主権 AI インフラストラクチャを介して多様な組織のユースケースを可能にします。組織はこれらの機能を活用して、Kubernetes 環境内のデータとモデルの完全な制御を維持しながら、インテリジェントなアプリケーションを構築します。

主なユースケース

内部ナレッジマネージメント

組織は AI アシスタントを展開し、従業員に内部文書、ポリシー、組織の知識への自然言語アクセスを提供します。

実装パターン

金融サービス企業は、コンプライアンス文書、規制当局への申告、および内部手順をナレッジベースにインデックス付けします。従業員は会話型インターフェイスを使用してこれらのシステムを照会し、データ主権を維持しながらソース引用付きの正確な回答を受け取ります。

技術コンポーネント

- SharePoint、Confluence、およびデータベースコンテンツのインデックスを作成するナレッジベース
- クエリの理解と応答生成のための LLM エンドポイント
- アクセス制御のためのロールベースのフィルタリングを備えた検索システム

実装ガイダンスについては、 *Knowledge Base Creation* を参照してください。

カスタマーサービスオートメーション

AI を活用したカスタマーサポートシステムは、日常的な問い合わせを処理しながら、複雑な問題を人間のエージェントにエスカレーションします。

実装パターン

E コマース企業は、安全な API を介して注文システム、製品カタログ、返品ポリシーにアクセスするアシスタントを展開します。これらのアシスタントは、品質保証のための完全な監査証跡を維持しながら、顧客の問い合わせの 60〜70%を自動的に解決します。

技術コンポーネント

- 注文管理システム統合のための REST API ツール
- 製品ドキュメントやよくある質問を含むナレッジベース
- 会話履歴とエスカレーションワークフローのスレッド追跡

実用的な例については、 *Customer Service Agent Quickstart* を参照してください。

Enterprise RAG Systems

検索拡張生成アプリケーションは、組織データの LLM 応答を基礎付け、精度を維持しながら幻覚を削減します。

実装パターン

医療機関は、臨床ガイドライン、研究論文、および治療プロトコールを介して RAG システムを実装しています。医療専門家は、ソース資料への引用付きで証拠に基づいた推奨事項を受け取り、トレーサビリティを保証しながら臨床上の意思決定をサポートします。

技術コンポーネント

- pgvector を使用して PostgreSQL に保存されたベクター埋め込み
- セマンティックマッチングとキーワードマッチングを組み合わせたハイブリッド検索
- 結果の最適化のためのモデルの再ランキング

アーキテクチャの詳細については、 *RAG Pipeline Documentation* を参照してください。

AI 対応 API

組織は、既存のアプリケーションと統合するための安全な API を介して AI 機能を公開します。

実装パターン

ソフトウェア会社は、標準の REST インターフェイスを介してアクセスできるモデルエンドポイントを展開することにより、製品内のインテリジェント機能を提供します。これらの API は、予測可能なパフォーマンス特性を維持しながら、ドキュメントの要約、感情分析、コンテンツ生成などのタスクを処理します。

技術コンポーネント

- OpenAI 互換エンドポイントを公開する InferenceServices
- 認証とレート制限を備えた API ゲートウェイ

- 高可用性のためのモデルレプリカ間のロードバランシング

デプロイメント手順については、 *Model Serving Guide* を参照してください。

ユーザーペルソナ

プラットフォームエンジニア

プラットフォームエンジニアは、AI Factory インフラストラクチャを管理し、信頼性の高いモデル提供と最適なリソースの使用を保証します。

主な責任

- GPU ノードとリソース割り当ての構成
- InferenceServices の展開と監視
- モデルライブラリとレジストリ接続を管理する
- セキュリティポリシーとネットワーク構成の実装

主要なワークフロー

1. *GPU Setup Guide* に従って GPU リソースをプロビジョニング
2. *Model Library Documentation* ごとにモデルレジストリを構成する
3. *Observability (AI Factory on HM)* を使用して推論パフォーマンスをモニタする

データサイエンティスト

データサイエンティストは AI Factory を活用して、インフラストラクチャの複雑さを管理せずにモデルを展開し、インテリジェントなアプリケーションを構築します。

主な責任

- ナレッジベースの作成と最適化
- RAG アプリケーションの取得戦略の構成
- モデルのパフォーマンスと精度を評価する
- デザインアシスタントの動作とツールの統合

主要なワークフロー

1. *Knowledge Base Management* に従ってナレッジベースを構築する
2. *Retriever Creation Guide* を使用してリトリーバーを構成する
3. *Assistant Configuration* を介してアシスタントを展開する

アプリケーション開発者

開発者は、API と SDK を介して AI 機能をビジネスアプリケーションに統合します。

主な責任

- モデルエンドポイントをアプリケーションに統合する
- スレッドを使用した会話管理を実装する
- アシスタント機能のカスタムツールを構築する
- エラー条件とフォールバック戦略を処理します

主要なワークフロー

1. *KServe Integration* を介してモデルエンドポイントにアクセス
2. *Thread Management* を使用して会話トラッキングを実装する
3. *Tool Development Guide* に従ってカスタムツールを作成する

ビジネスアナリスト

ビジネスアナリストは、 Gen AI Builder のビジュアルインターフェイスを使用して、コーディングの専門知識なしで AI アプリケーションを作成します。

主な責任

- アシスタントの動作と知識ドメインを定義する
- データソース接続の構成
- アシスタントの応答をテストおよび検証する
- 使用パターンと効果を監視する

主要なワークフロー

1. ビジュアルビルダーインターフェイスを使用したデザインアシスタント
2. 構成ウィザードを介してデータソースを接続する
3. インタラクションをテストし、動作を繰り返し調整する

業界固有の実装

金融サービス

ユースケース取引パターンを分析し、必要なレポートを生成する規制コンプライアンスアシスタント。

コンポーネント

- 規制文書のナレッジベース
- トランザクションデータベースにアクセスするツール
- コンプライアンス追跡のための監査ログ

ヘルスケア

ユースケース証拠に基づいた治療の推奨を提供する臨床決定支援システム。

コンポーネント

- 医学文献とガイドラインに関する RAG
- 電子医療記録との統合
- 患者データの厳密なアクセス制御

製造業

ユースケースセンサーデータと技術マニュアルを使用して、機器の問題を診断するメンテナンスアシスタント。

コンポーネント

- 機器ドキュメントのナレッジベース
- IoT センサーからのリアルタイムデータの統合
- 予知保全モデルのエンドポイント

実装の考慮事項

スケーラビリティープランニング

さまざまなユースケースには、さまざまなレベルのスケールが必要です。

- **大規模なカスタマーサービス**自動スケーリングを備えた複数のモデルレプリカ
- **内部ナレッジシステム** キャッシュ最適化による中程度のスケーリング
- **特殊なアシスタント** 専用のリソースを使用した単一のインスタンス

セキュリティ要件

ユースケースはセキュリティ構成を決定します。

- **パブリック向けアプリケーション** 認証を使用した外部イングレス
- **内部システム** RBAC を使用したクラスターローカルアクセス
- **規制対象業界** 監査ログと暗号化の強化

パフォーマンスの最適化

ユースケースの特性に基づいて最適化する

- **リアルタイムアプリケーション**エッジキャッシングを使用して提供する低レイテンシーモデル
- **バッチ処理** より大きなバッチサイズでのより高いスループット
- **インタラクティブアシスタント**バランスの取れたレイテンシーとスループット

ユースケース別のスタートガイド

ナレッジマネージメントの場合

1. *Knowledge bases in the asset library* を確認する
2. *Data Source Integration* をフォローする
3. *Hybrid Search* を実装する

カスタマーサービス用

1. *Assistant Concepts* から始めます
2. 動作制御用に *Rulesets* を構成する
3. *Quickstart UI* を使用して展開する

API 開発の場合

1. *InferenceService Creation* を介してモデルを展開する
2. *Python Client Guide* に従ってエンドポイントにアクセスします
3. *Observability Documentation* ごとにモニタリングを実装する

追加リソース

アーキテクチャリファレンス

- *AI Factory Architecture on Hybrid Manager*

実装ガイド

関連ドキュメント

- *Hybrid Manager Overview*
- *Analytics Personas*

Frequently Asked Questions - AI Factory on Hybrid Manager

目次

- プラットフォーム機能
- インストールとセットアップ
- モデル管理
- *Gen AI Builder* ランタイム
- 運用とメンテナンス
- 障害対応
- セキュリティとコンプライアンス

- パフォーマンスとスケーリング

プラットフォーム機能

HM 1.3 はどのようなタイプのモデルをサポートしていますか?

Hybrid Manager 1.3 は、NVIDIA NIM コンテナを介してのみ大規模言語モデル LLM 展開をサポートしています。従来の機械学習モデル分類、回帰、時系列予測は、このリリースではサポートされていません。

サポートされている NVIDIA NIM モデルカテゴリ

- **テキスト生成** チャットおよび完了タスク用の大規模な言語モデル
- **テキスト埋め込み**セマンティック検索および RAG アプリケーションのモデル
- **Text Reranking** 検索結果最適化のモデル
- **マルチモーダルモデル** CLIP および OCR 機能を含むビジョンモデル

カスタムモデルを展開できますか?

カスタムモデルを NVIDIA NIM コンテナとしてパッケージして、HM 1.3 と互換性を保つ必要があります。標準の機械学習フレームワーク scikit-learn、XGBoost、従来の ML の TensorFlow はサポートされていません。カスタム LLM は、NIM コンテナ仕様と API 標準に準拠している場合に、展開できます。

カスタム NIM 展開手順については、 *Private Registry Integration* を参照してください。

AI Factory とクラウド AI サービスの違いは何ですか?

AI Factory は、AI 操作に対する完全な主権を提供します。

- モデルは Kubernetes インフラストラクチャ内で実行されます
- データは組織の境界内に残ります
- 推論のための外部 API 依存関係はありません
- 規制コンプライアンスのための完全な監査証跡

インストールとセットアップ

インフラストラクチャの最小要件は何ですか?

コア要件

- NVIDIA GPU オペレーターを使用した Kubernetes 1.27+
- NIM コンテナ L40S、A100、H100 と互換性のある NVIDIA GPU
- モデルアーティファクト用の 100GB 以上のオブジェクトストレージ
- NVIDIA NGC レジストリまたはエアギャップ構成へのネットワーク接続

NIM モデルタイプ別の GPU 要件

- テキスト補完 Llama 3.3 70B 4 x L40S GPU

- テキスト埋め込み 1 x L40S GPU
- テキストのリランキング 1 x L40S GPU
- ビジョンモデル 1 x L40S GPU

包括的な仕様については、 *Prerequisites for AI Factory on Hybrid Manager* を参照してください。

NIM モデルの GPU ノードを構成するにはどうすればよいですか?

GPU ノードの準備には以下が含まれます。

1. クラスターに NVIDIA GPU オペレーターをインストールする
2. `nvidia.com/gpu=true` で GPU ノードにラベルを付ける
3. 専用のスケジューリングに GPU テイントを適用する
4. NIM 要件の CUDA 互換性を確認する

GPU Setup Documentation で利用可能な詳細な手順。

AI Factory はエアギャップ環境で動作できますか?

エアギャップ展開には事前の準備が必要です。

1. NVIDIA NIM イメージをプライベートレジストリにミラーリングする
2. モデルプロファイルのダウンロードとキャッシュ
3. プロファイルをオブジェクトストレージにアップロードする
4. プライベートレジストリアクセス用にモデルライブラリーを構成する

Air-Gap Configuration に文書化されている手順を完了します。

モデル管理

NVIDIA NIM モデルを展開するにはどうすればよいですか?

NIM モデル展開ワークフロー

1. HM コンソールでモデルライブラリーにアクセスします
2. カタログから NVIDIA NIM モデルを選択します
3. リソースの構成 GPU 割り当て、メモリ、レプリカ
4. InferenceService をプロジェクト名前空間に展開する
5. 生成されたエンドポイントを介したアクセス

ステップバイステップガイド *Create InferenceService* 。

デフォルトで使用可能な NVIDIA NIM モデルはどれですか?

HM 1.3 のデフォルト NIM モデル

- **llama-3.3-nemotron-super-49b** 高度な推論とチャット
- **llama-3.2-nemoretriever-300m-embed** テキスト埋め込み
- **llama-3.2-nv-rerankqa-1b** クエリドキュメントのリランキング
- **nvclip** マルチモーダル埋め込み
- **paddleocr** 光学式文字認識

NIM モデルバージョンを管理するにはどうすればよいですか?

バージョン管理戦略

- モデルライブラリーは、各 NIM イメージのバージョンタグを維持します
- ブルーグリーン展開により、ゼロダウンタイムの更新が可能
- カナリア展開により、段階的なトラフィックのシフトが可能
- InferenceService 構成の更新を介したロールバック

Gen AI Builder

Gen AI のドキュメントはどこで見つけられますか?

プライマリジェネレーション AI リソース

- *Gen AI Builder on Hybrid Manager* - 包括的なリファレンス
- *Concepts and architecture* - コンポーネントの概要
- *AI Factory Architecture on Hybrid Manager* - システムデザイン

RAG のナレッジベースを作成するにはどうすればよいですか?

ナレッジベース作成プロセス

1. データソースの構成データベース、ドキュメント、API
2. NIM 埋め込みモデルを介したコンテンツの処理
3. pgvector を使用して PostgreSQL にベクターを保存する
4. 検索の取得戦略を構成する

実装ガイド *Knowledge Base Creation* 。

アシスタントとは何ですか?どのように機能しますか?

アシスタントは、ユーザー、ナレッジベース、および外部システム間の相互作用を調整します。これらは、スレッドを介して会話コンテキストを維持しながら、生成に NIM モデルを活用します。アシスタントは、検索、ツールの使用、および構造化推論機能を組み込んでいる点で、単純なチャットボットとは異なります。

運用とメンテナンス

NIM モデルのパフォーマンスをモニタするにはどうすればよいですか?

モニタリングには以下が含まれます。

メトリックコレクション

- 推論レイテンシーの Prometheus メトリック
- GPU 使用率とメモリ消費量
- トークン生成のスループット
- 要求の成功率

ビジュアライゼーション

- HM コンソールに統合された Grafana ダッシュボード
- モデル固有のメトリックのカスタムパネル
- SLA 違反のアラート構成

参照 *Observability (AI Factory on HM)* 。

NIM モデルの更新はどのように処理すればよいですか?

運用展開の更新手順

1. **検証** 開発名前空間に新しいバージョンを展開します
2. **テスト** パフォーマンステストと精度テストを実行します
3. **展開** カナリアまたはブルーグリーン戦略を実装します
4. **モニタリング** 移行中のメトリックの追跡
5. **決定** メトリックに基づいて完全なロールアウトまたはロールバック

障害対応

NIM モデルの起動に失敗します-診断手順は?

一般的な初期化障害

1. **GPU の使用不可** GPU リソースがモデル要件と一致することを確認します。
2. **イメージのプルの失敗** NGC 資格情報とネットワーク接続を確認します
3. **プロファイルキャッシュがありません**エアギャップセットアップで使用可能なプロファイルを確認します

4. **メモリ不足** モデルサイズのメモリ割り当てを検証する

診断コマンド

```
kubectl describe inferenceservice <name> -n <namespace>
kubectl logs <pod-name> -n <namespace>
kubectl get events -n <namespace>
```

高い推論レイテンシー - 最適化戦略?

パフォーマンス最適化のアプローチ

- **バッチ処理** スループットを最適化するためにバッチサイズを増加します
- **モデル量子化** サポートされている場合は INT8 量子化を使用します
- **応答キャッシュ**頻繁なクエリーをアプリケーションレイヤーでキャッシュします
- **ホリゾンタル スケーリング**負荷分散のために追加のレプリカを展開します

RAG アプリケーションの取得品質は低いですか?

取得のトラブルシューティング

1. **埋め込み品質** 適切な NIM 埋め込みモデル選択を確認します
2. **ドキュメントチャンク**チャンクサイズを調整し、パラメーターをオーバーラップします
3. **検索パラメーター** トップ k および類似性のしきい値を調整する
4. **インデックスの完全性**すべてのドキュメントが正常に処理されたことを確認します

セキュリティとコンプライアンス

アクセス制御はどのように実装すればよいですか?

AI リソースのロールベースのアクセス制御には、次のものが含まれます。

- 名前空間とリソース権限のための Kubernetes RBAC
- 展開承認のためのモデルライブラリーのアクセス制御
- 外部エンドポイントアクセスの API キー管理
- サービス間通信のネットワークポリシー

どのような暗号化が実装されていますか?

暗号化カバレッジ

- **保存時** Kubernetes シークレット暗号化、データベース暗号化
- **転送中** API 呼び出しの TLS、サービスメッシュ内の mTLS
- **モデルアーティファクト** 暗号化されたオブジェクトストレージ
- **ナレッジベース** PostgreSQL の暗号化されたベクターストレージ

どの操作が監査されますか?

監査ログキャプチャ

- NIM モデルの展開と構成の変更
- 推論要求設定可能な詳細レベル
- ナレッジベースのクエリと更新
- スレッド追跡を介したアシスタントの会話
- AI リソースの管理操作

パフォーマンスとスケーリング

リソースクォータはどのように構成すればよいですか?

リソースクォータは、名前空間レベルでのリソースの枯渇を防ぎます。プロジェクト要件と利用可能なインフラストラクチャ容量に基づいて、GPU クォータ、メモリ制限、およびストレージ制約を構成します。

垂直方向と水平方向のどちらの場合にスケーリングする必要がありますか?

ホリゾンタル スケーリング 追加のレプリカ

- 高い同時要求量
- ステートレス推論ワークロード
- 負荷分散の要件

垂直スケーリング インスタンスごとのリソースの増加

- 大規模なモデルのメモリ要件
- バッチ処理の最適化
- 単一要求のレイテンシーの最小化

HM 1.3 はどのモデルサイズをサポートできますか?

モデルサイズのコンストレイン

- **シングル GPU** 最大 13B パラメーターのモデル
- **マルチ GPU** テンソル並列処理を使用して最大 70B+パラメーターをモデル化
- **メモリ制限** GPU ごとに 80GB A100、48GB L40S

NVIDIA NIM は、利用可能なリソースに基づいて、モデルのシャーディングと並列処理を自動的に処理します。

追加リソース

はじめに

- *Prerequisites for AI Factory on Hybrid Manager*
- *AI Factory Architecture on Hybrid Manager*

実装ガイド

- モデルサービング
- *Gen AI Builder on Hybrid Manager*
- *Knowledge bases in the asset library*

トラブルシューティングリソース

- *Model Verification*
-

ここで対処されていない問題については、 EDB サポートに連絡するか、*AI Factory Hub* にコンサルトしてください。

GPU Recommendations for Default NIM Models

概要

ハイブリッドマネージャーからは、AI モデルの 2 つの主なコンシューマーが存在します。

- AI ナレッジベースを作成および維持するための PG.AI Knowledge BaseAIDB Postgres 拡張機能。
- PG.AI GenAI Builder エージェントニック AI アシスタントをビルドするためのコンテナ化されたグリップテープ。

デフォルトの NIM モデル

Model type	NIM model	NVIDIA NIM documented resource requirements
Text completion	llama-3.3-70b-instruct	4 × L40S
Text embeddings	arctic-embed-l	1 × L40S
Image embeddings	nvclip	1 × L40S
OCR	paddleocr	1 × L40S
Text reranking	llama-3.2-nv-rerankqa-1b-v2	1 × L40S

最小 GPU 要件

上記のデフォルトモデルに基づいて、それらを同時に実行できる最小は 8 × L40S GPU です。

クラウドマッピング

- AWS EKS 2 × `g6e.12xlarge` ノードを含むノードグループを推奨します。
- GCP GKE 2 × `a2-highgpu-4g` ノードを含むノードプールを推奨します。

注 GCP は、L40S GPU を提供していません。 A100 GPU を搭載した推奨 A2 ノードは、上記の NIM モデルでサポートおよび文書化されています。

AI Factory Quickstart

ハブクイックリンク *Gen AI Builder on Hybrid Manager* — **ナレッジベースとパイプライン** — **モデルサービング**

ビルドするもの

このクイックスタートでは、ナレッジベースからの質問に回答する実用的な Gen AI アシスタントと、アプリケーションから呼び出すことができる展開されたモデルエンドポイントをゼロから構築します。

前提条件

- デプロイメントの GPU の準備 *Setup GPU*
- Gen AI のコンセプトとパイプラインに関する知識 *Gen AI Builder on Hybrid Manager* 、**ナレッジベースとパイプライン**

ステップ 1-ナレッジベースを作成する

ナレッジベースを作成し、コンテンツを取り込んで、アシスタントがコンテキストを取得できるようにします。

- **ステップ** *1-***ナレッジベースを作成する**
- オプショナル ナレッジベースの概念を確認してください *Knowledge bases in the asset library*

ステップ 2-アシスタントを作成する

会話アシスタントを構築し、ナレッジベースを添付します。

- **ステップ** *2-***アシスタントを作成する**
- オプショナル レビューアシスタント **アシスタントとは何ですか***?***どのように機能しますか***?**

ステップ 3-モデルエンドポイントを展開する

KServe を使用してモデルを展開して、アプリケーションが直接呼び出せるようにします。

- *Deploy NIM containers*
- *Create an InferenceService*
- オプショナル ランタイムのカスタマイズ *Configure ServingRuntime*

ステップ 4-アプリからエンドポイントを呼び出す

単純なクライアントで展開されたモデルをテストし、エンドポイントに到達する方法を学びます。

- *Quickstart (Python)*
- *Access KServe endpoints*

ステップ 5-モニタリングと反復

モデルの状態とパフォーマンスを監視します。

- *Monitor InferenceService*
- オプショナル ナレッジベースのパフォーマンスの調整 *Knowledge Base performance*

次のステップ

- 段階的に学習する **ラーニングパス**
- 探索パターン *Common workloads and use cases* および **一般的な解決策**
- Gen AI および Model Serving ハブを介した API とリファレンスマテリアルの詳細

7.6.2 Manual

Enabling AI Factory

ハイブリッドマネージャーで **GenAI Builder** を使用する前に、 **AI Factory** を有効にする必要があります。これには、2 つの主な手順が含まれます。

1. 暗号化用の **フェルネットキー**を設定しますまだ作成されていない場合。
2. GenAI Builder をデータレイクの **S3 互換バケット** に接続します。

1.Fernet キーを生成しますまだ作成していない場合

AI Factory は、Fernet キーを使用して機密データを暗号化します。ハイブリッドマネージャーのセットアップ中に`-install-secrets.sh` スクリプトのいずれかを実行した場合、このキーは既に持っている可能性があります。

そうでない場合は、次のコマンドを使用して手動で生成します HM クラスターに `kubectl` を構成する必要があります。

```
FERNET_KEY=$(dd if=/dev/urandom bs=32 count=1 2>/dev/null | base64)

kubectl create namespace upm-griptape

kubectl apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
```

(次のページに続く)

(前のページからの続き)

```
  name: fernet-secret
  namespace: upm-griptape
stringData:
  FERNET_KEY: ${FERNET_KEY}
EOF
```

これにより、GenAI Builder が使用する Kubernetes シークレットが `upm-griptape` 名前空間に作成されます。

2.S3 互換バケットを作成する

GenAI Builder では、構造、ツール、インデックス付きデータを保存するために **S3 バケット** データレイクが必要です。

バケットプロバイダーから次の情報を収集します。

Field	Example
Bucket name	*mycompany-genai-datalake*
Access key	AWS access key ID or equivalent
Secret access key	AWS secret access key or equivalent
URL	*https://s3.us-west-2.amazonaws.com/*
Region	*us-west-2*

!!!注 AWS を使用する場合、バケット名はグローバルに一意である必要があります。

オプショナル CORS 構成

GenAI Builder Web UI を使用する予定がある場合は、バケットに CORS ルールを構成して、UI がアクセスできるようにします。AWS S3 の場合、一般的な構成は次のとおりです。

```
[
  {
    "AllowedHeaders": ["*"],
    "AllowedMethods": ["PUT", "POST", "DELETE", "GET", "HEAD"],
    "AllowedOrigins": ["https://${PORTAL_DOMAIN_NAME}"],
    "ExposeHeaders": []
  }
]
```

`${PORTAL_DOMAIN_NAME}` を HM ポータルドメインに置き換えます。

Google Cloud StorageGCS

GCS は、 **S3 相互運用レイヤー** を介して使用できます。

1. GCS 設定で S3 の相互運用性を有効にします。
2. ロールを持つサービスアカウントを作成または更新します。

- ストレージ管理者
- サービスアカウントトークン作成者

3. HMAC キーペアを生成します。
4. バケット CORS の構成 cors.json として保存

```
[
  {
    "origin": ["https://<portal-url>"],
    "method": ["GET", "PUT", "POST", "DELETE", "HEAD"],
    "responseHeader": ["*"],
    "maxAgeSeconds": 3600
  }
]
```

以下で応募してください

```
gsutil cors set cors.json gs://<bucket-name>
```

3. バケットを GenAI Builder に接続する

バケットの準備ができたら、

1. ハイブリッドマネージャーで、 **Launchpad** → **GenAI Builder** を開きます。
2. 管理者アカウントでログインします。

ユーザー名これは、HM の構成に使用したユーザー名と同じです。ユーザー名を取得するには

```
kubectl exec -it resourcedb-1 -n upm-griptape -- psql resources -c "select * from␣
↪griptape_cloud_user"
```

パスワード 初回パスワードはデフォルトで pgai_admin です。

3. **Administrator Console** で、AI Factory 構成フォームを開きます。
4. バケットの詳細を入力します。

Field	Example
S3 URL	*https://s3.us-west-2.amazonaws.com/*
S3 Bucket	*mycompany-genai-datalake*
S3 Access Key	*AKIA⋯*
S3 Secret Key	*wJalr⋯*
S3 Region	*us-west-2*

5. **Submit** をクリックします。

4. 成功 AI ファクトリーが有効になりました

保存したら

- GenAI Builder は、データレイクにコンテンツを保存および取得できます。
- **ライブラリ、ナレッジベース、およびアシスタント**を作成できるようになりました。
- 構成の詳細 Fernet キーやバケット設定などは、ハイブリッドマネージャー内で処理されます。

AI Factory が有効になり、使用する準備が整います。

Gen AI Builder on Hybrid Manager

概要

Hybrid Manager 内の Gen AI Builder を使用すると、組織は、大規模言語モデルと組織データを組み合わせたインテリジェントなアプリケーションを作成できます。すべてが制御された Kubernetes インフラストラクチャ内で実行されます。このプラットフォームは、外部依存なしで検索拡張生成 RAG システム、会話アシスタント、自動ワークフローを構築するためのビジュアルおよびプログラム的なインターフェイスを提供します。*Gen AI Builder on Hybrid Manager* を参照してください。

この統合により、完全なデータ主権が保証されます。ナレッジベース、モデルの相互作用、会話履歴はインフラストラクチャ内に残りますが、ローカルに展開されたモデルを介して最先端の AI 機能を活用します。

コアアーキテクチャ

LLM 統合

Gen AI Builder は、ハイブリッドマネージャーの **モデルサービング** インフラストラクチャを介して展開されたモデルに接続し、テキスト生成、埋め込み、および再ランキング操作のための言語モデルへのシームレスなアクセスを提供します。

モデルアクセスパターン

- 低遅延の推論のためのクラスター DNS を介した内部エンドポイント
- アプリケーション統合のための安全なゲートウェイを介した外部エンドポイント
- 標準クライアントライブラリを有効にする OpenAI 互換 API *OpenAI API compatibility* を参照してください

サポートされているオペレーション

- 会話型インタラクションのチャットコンプリート
- セマンティック検索とナレッジベース作成のための埋め込み
- 取得ワークフローでの結果最適化のための再ランキング

ナレッジベースインフラストラクチャ

ナレッジベースは、*pgvector and Vector Engine concepts* で強化された PostgreSQL データベース内の埋め込みとインデックス作成を介して、組織データを検索可能な AI 対応形式に変換します。*Knowledge bases in the asset library* および **ナレッジベースパイプライン** を参照してください。

データ処理パイプライン

1. **インジェスト**データベース、ドキュメント、API を含む *data sources* に接続します
2. **チャンキング** コンテンツを意味的に意味のあるセグメントに分割する
3. **エンベディング** デプロイされたモデルを使用してベクトル表現を生成
4. **インデックス作成**フィルタリング用のメタデータを使用してベクトルを PostgreSQL に保存します

ストレージアーキテクチャ

- pgvector 拡張子を使用して PostgreSQL に保存されたベクター埋め込み
- フィルタリングとアトリビューションのために維持されるドキュメントメタデータ
- 参照と再生成のために保存された元のコンテンツ
- 動的ナレッジベースでサポートされているインクリメンタル更新

インテリジェントなアプリケーションの構築

アシスタント開発

アシスタントは、取得、推論、アクションを組み合わせた構成可能なワークフローを介して、ユーザー、ナレッジベース、および外部システム間の相互作用を調整します。**アシスタントとは何ですか?どのように機能しますか?** を参照してください。

構成コンポーネント

- **ディレクティブ** アシスタントの目的と動作を定義します
- **ナレッジ統合** コンテキストに関連する *RAG* **のナレッジベースを作成するにはどうすればよいですか?** を接続します
- **ツールバインディング****データを外部ツールに接続する** を介して外部システムの相互作用を有効にします
- **モデル選択** 生成タスクに適切な LLM を選択します

例 カスタマーサービスアシスタント

```
assistant:
  name: customer-support
  directive: "Help customers with product inquiries and order status"
  knowledge_bases:
    - product-documentation
    - order-database
  tools:
```

(次のページに続く)

(前のページからの続き)

```
    - order-tracking-api
    - ticket-creation
  model: llama-3-8b-instruct
```

取得拡張生成

RAG パターンは、LLM 応答を組織のナレッジに根付かせ、会話機能を維持しながら幻覚を削減します。*Retrievers* を参照してください。

取得フロー

1. ユーザークエリーは、ナレッジベース全体のセマンティック検索をトリガーします
2. ベクトルの類似性に基づいて取得された関連文書
3. ユーザークエリーとともに LLM に提供されるコンテキスト
4. 取得した情報を使用して生成された応答

最適化戦略

- ベクトル一致とキーワード一致を組み合わせたハイブリッド検索
- ドメイン固有の取得のためのメタデータフィルタリング
- 結果の関連性を向上させるための再ランキング *Using NVIDIA NIM* を参照してください
- 大規模なドキュメントのコンテキストウィンドウ管理

ツールの統合

ツールは、セキュリティ境界を維持しながら外部システムおよび API との相互作用を有効にすることにより、アシスタント機能を拡張します。**データを外部ツールに接続する** を参照してください。

ツールタイプ

- **REST API** 認証を使用して外部サービスに接続します
- **データベースクエリー** 運用データベースに対して SQL を実行します
- **内部サービス** クラスター内の Kubernetes サービスを呼び出します
- **カスタムファンクション**特殊なロジックをコンテナ化されたサービスとして展開します

セキュリティ制御

- Kubernetes シークレットを介した資格管理
- 悪用を防ぐためのレート制限
- コンプライアンス追跡のための監査ログ
- アシスタント構成ごとのスコープ制限

一般的な使用例

エンタープライズナレッジマネージメント

組織は、異種の情報ソースを、自然言語インターフェイスを介してアクセスできる統合された検索可能なナレッジベースに統合します。

実装パターン

- 複数のリポジトリからドキュメントを取り込みます Confluence、SharePoint、Google Drive
- クロスソース検索のセマンティックインデックスを作成する
- フィルタリングされたアクセスを使用してさまざまな部門にアシスタントを展開する
- コンプライアンス要件の監査証跡を維持する

例 技術文書アシスタントソフトウェア会社は、すべての技術文書、API リファレンス、およびサポートチケットのインデックスを作成します。エンジニアは自然言語を使用してシステムを照会し、ソースの引用を含む状況依存の回答を受け取ります。

カスタマーサービスオートメーション

インテリジェントなアシスタントは、製品知識とリアルタイムのシステムアクセスを組み合わせて、顧客の問い合わせに対応します。

機能

- ドキュメントナレッジベースを使用して製品の質問に回答する
- API 統合を介した注文ステータスの確認
- 複雑な問題のサポートチケットを作成する
- 必要に応じて人間のエージェントにエスカレーション

例 E-commerce サポートボットオンライン小売業者は、顧客の問い合わせの 70%を自動的に処理するアシスタントを展開し、統一インターフェイスを介して注文システム、製品カタログ、返品ポリシーにアクセスします。

コンプライアンスと監査のサポート

組織は Gen AI を活用して、複雑な規制要件をナビゲートし、コンプライアンス文書を維持します。

機能

- 特定の要件について規制知識ベースを照会します
- 運用データからコンプライアンスレポートを生成
- 会話スレッドを介して意思決定の根拠を追跡する
- レビューのために不変の監査ログを維持する

例金融コンプライアンスアシスタント銀行は、コンプライアンス担当者が規制を解釈し、必要なレポートを生成し、完全なトレーサビリティを持つ意思決定プロセスを文書化するのを支援するアシスタントを作成します。

運用上の考慮事項

パフォーマンスの最適化

システムのパフォーマンスは、複数の要素に依存します。運用展開には、注意的な調整が必要です。

主要なメトリック

- **取得レイテンシー** 関連ドキュメントを取得する時間
- **Generation Latency** 応答の LLM 推論時間
- **エンドツーエンドレイテンシー** クエリーから応答までの合計時間
- **スループット** 処理される同時要求

最適化テクニック

- 並列検索のためのナレッジベースの分割
- よくある質問の回答キャッシュ
- モデルの量子化による推論速度の向上
- データベースアクセスのための接続プーリング

リソース管理

Gen AI アプリケーションでは、コンピューティング、メモリ、ストレージのディメンション全体にわたって注意深くリソースを割り当てる必要があります。

リソース要件

- **Knowledge Base Storage** ベクターとメタデータの PostgreSQL 容量
- **モデルサービス** LLM 推論用の GPU リソース
- **アプリケーションポッド** オーケストレーションロジック用の CPU とメモリ
- **ネットワーク帯域幅** コンポーネント間のデータ転送

スケーリング戦略

- 検索サービスのホリゾンタル スケーリング
- 開発ワークロードの GPU 共有
- プロダクションアシスタントのための専用リソース
- 要求パターンに基づいた自動スケーリング

セキュリティフレームワーク

包括的なセキュリティ制御は、アプリケーションのライフサイクル全体で機密データを保護します。

アクセス制御

- アシスタントアクセスのためのロールベースの権限

- ユーザーコンテキストに基づいたナレッジベースのフィルタリング
- アシスタント構成ごとのツールの承認
- 外部アクセスの API キー管理

データ保護

- ナレッジベースコンテンツの保存時の暗号化
- すべてのサービス間通信の TLS
- ログの機密情報のサニタイズ
- Kubernetes シークレットの安全な資格情報ストレージ

モニタリングとオブザーバビリティ

会話トラッキング

スレッド管理は、デバッグとコンプライアンスのためのアシスタントの相互作用の包括的な可視性を提供します。*Gen AI Builder* **アシスタント & スレッド** を参照してください。

ネジコンポーネント

- ユーザーのクエリとアシスタントの応答
- 関連性スコアを含む取得されたドキュメント
- パラメーターと結果を使用したツール呼び出し
- トークンの使用量と遅延メトリック

分析機能

- 応答品質評価
- 知識ギャップの特定
- パフォーマンスのボトルネックの検出
- ユーザー満足度の追跡

システムモニタリング

運用メトリックにより、信頼性の高いサービス提供とキャパシティプランニングが保証されます。

監視エリア

- ナレッジベースのクエリパターンとパフォーマンス
- モデルのエンドポイントの使用率とレイテンシー
- ツール API の成功率と応答時間
- リソース消費傾向

アラートシナリオ

- SLA しきい値を超える高レイテンシー

- 機能に影響するツール呼び出しの失敗
- リソース枯渇のリスク
- 問題を示す異常な使用パターン

実装ガイド

はじめに

組織は、展開を拡大する前に、価値を実証する焦点を絞ったユースケースから始める必要があります。

初期セットアップ

1. ナレッジベース作成のためのエンベディングモデルを展開する
2. 初期ナレッジベースのデータソースの構成
3. 生成タスクのための LLM の展開
4. 基本的な検索を備えた単純なアシスタントを作成する

検証手順

- サンプルクエリを使用した取得品質のテスト
- ソース文書に対する応答の精度を検証する
- 予想される負荷の下でのパフォーマンスを測定する
- セキュリティ制御とアクセスパターンを確認します

ベストプラクティス

ナレッジベースデザイン

- 帰属のための明確なドキュメント境界を維持する
- 新しいコンテンツの定期的な更新サイクルを実装します
- メタデータを効果的に使用してフィルタリング
- 取得品質メトリックの監視

アシスタントの構成

- 制限的なディレクティブから始めて、徐々に拡張します
- エッジケースと障害シナリオのテスト
- ツール障害のフォールバック戦略の実装
- バージョン管理アシスタントの構成

運用展開

- ベースラインパフォーマンスメトリックを確立する
- 段階的なロールアウト戦略の実装

- ユーザーのフィードバックと満足度を監視する
- 構成のバックアップを維持する

ハイブリッドマネージャーとの統合

プロジェクト組織

Gen AI リソースはハイブリッドマネージャープロジェクト内で展開され、名前空間の分離とリソース制御を継承します。

プロジェクトリソース

- PostgreSQL データベースとしてのナレッジベース
- Kubernetes 展開としてのアシスタント
- サービスとしてのツールの統合
- 永続ストレージとしてのスレッド

UI ワークフロー

Hybrid Manager は、深い技術的な専門知識を必要とせずに Gen AI 開発のためのビジュアルインターフェイスを提供します。

ビジュアル機能

- ドラッグアンドドロップのアシスタント構成
- ナレッジベース作成ウィザード
- ツール登録インターフェイス
- 会話テスト環境

プログラムによるアクセス

SDK と API により、既存のアプリケーションおよび自動ワークフローとの統合が可能になります。

統合オプション

- アプリケーション開発用の Python SDK
- サービス統合のための REST API
- GitOps ワークフローの Kubernetes オペレーター
- 自動化スクリプト用の CLI ツール

次のステップ

Gen AI Builder 機能の検討を開始します。

1. **コンセプトの確認** *Concepts and architecture* および *AI Factory Architecture on Hybrid Manager* を理解する

2. **ナレッジベースの作成** *knowledge base guide* に従ってください
3. **Build Assistant** *assistant creation guide* を使用します
4. **アプリケーションの展開** *quickstart UI* を使用して実装します

詳細なドキュメントについては

- **アシスタントとは何ですか?どのように機能しますか?**
- *Knowledge Bases Manual*
- *Tools Documentation*
- *SDK Reference*

Model Management on Hybrid Manager

目的と利点

ハイブリッドマネージャー内のモデル管理は、Kubernetes インフラストラクチャで実行されている AI モデルに一元的なガバナンスと展開機能を提供します。このシステムにより、組織はエンタープライズグレードの **モデルサービング** インフラストラクチャを活用しながら、AI 機能の完全な制御を維持できます。

この統合は、AI を大規模に展開する組織の重要な要件に対応します。承認されたレジストリを介したモデルガバナンス、GPU アクセラレーションを使用したスケーラブルな推論サービング、およびハイブリッドマネージャーのコントロールプレーンを介した統合管理。制御されたインフラストラクチャ内でモデルを実行することにより、最先端の AI 機能にアクセスしながら、データ主権を維持します。

コアコンセプト

モデルライブラリー

モデルライブラリーアーキテクチャ は、AI モデルイメージの集中ガバナンスシステムとして機能します。Hybrid Manager のアセットライブラリーストラクチャ内で動作し、運用展開の準備ができている検証済みモデルの厳選されたビューを提供します。

ライブラリは、多段階のガバナンスを実装します。

- 信頼できるコンテナレジストリからの自動同期
- セキュリティスキャンと脆弱性評価
- 組織ポリシーに基づいた承認ワークフロー
- バージョン管理とドキュメントのメタデータ管理

ライブラリ内のモデルは、*Gen AI Builder on Hybrid Manager* アシスタント、*RAG* **のナレッジベースを作成するにはどうすればよいですか?** パイプライン、カスタム推論アプリケーションを含むすべての AI Factory 機能を強化します。ライブラリのガバナンスフレームワークを通じて検証されたモデルのみが運用環境に到達できます。

モデルサービング

モデルサービング は、Kubernetes クラスター内の KServe を使用して、承認されたモデルをスケーラブルな推論エンドポイントに変換します。このインフラストラクチャは、自動スケーリング、状態管理、およびリソースの最適化を備えた実稼働グレードのモデルの展開を提供します。

主要なサービス提供機能には次のものが含まれます。

- デプロイされたモデルエンドポイントを定義する ***InferenceService*** リソース
- さまざまなモデルフレームワーク用に最適化された ***ServingRuntime*** 構成
- 高パフォーマンスの推論のための GPU の割り当てとスケジューリング *Setup GPU* および *Update GPU resources* を参照してください。
- 認証を使用した内部および外部エンドポイントアクセス

管理インターフェイス

Hybrid Manager は、Web コンソールを介して統合管理を提供し、完全な構成可能性を維持しながら Kubernetes の複雑さを抽象化します。このインターフェイスにより、次のことが可能になります。

- ライブラリからサービングまでモデル展開のビジュアルワークフロー
- リソースの割り当てとスケーリング構成
- 推論メトリックと GPU 使用率のモニタリングダッシュボード
- アクセス制御とエンドポイント管理

実装ワークフロー

モデル登録

組織は、信頼できるモデルソースへのリポジトリ接続を構成することから始めます。モデルライブラリーは、定義されたルールに基づいて外部レジストリと同期し、新しいモデルバージョンを自動的に検出および検証します。

```
External Registry → Repository Rules → Security Scanning → Model Library
```

リポジトリルールは、環境に入るモデルを決定し、インジェスト時点で組織ポリシーを実装します。この自動化されたアプローチにより、ガバナンス標準を維持しながら、手動オーバーヘッドが削減されます。

モデルの展開

検証されたモデルは、サービスインフラストラクチャを構成するガイド付きワークフローを介して展開されます。

1. **モデル選択**バージョン、パフォーマンス特性、リソース要件を含むメタデータを使用してライブラリ内の使用可能なモデルを参照します。
2. **実行時構成**モデルフレームワーク vLLM、TensorRT-LLM、カスタム用に最適化された *ServingRuntimes* を選択または作成します

3. **リソースの割り当て**予想されるワークロードに基づいて、GPU、メモリ、および CPU 要件を定義します
4. **エンドポイント構成**認証を使用した内部クラスターアクセスまたは外部 API エンドポイントを設定します *Access KServe endpoints* を参照してください。

システムは、KServe が管理する InferenceService リソースを作成し、ポッドのスケジューリング、ヘルスモニタリング、トラフィックルーティングを自動的に処理します。

運用管理

導入されたモデルは、要求に基づいた自動スケーリングを使用した継続的な監視の下で動作します。ハイブリッドマネージャーは、次のことを介して可視性を提供します。

- レイテンシーやスループットを含むリアルタイムの推論メトリック
- リソース最適化のための GPU 使用率追跡
- プロアクティブなメンテナンスのエラー率と状態
- リソース消費に基づいたコスト分析

GPU インフラストラクチャ

リソースタイプ

ハイブリッドマネージャーは、ワークロード要件に一致するさまざまな GPU 構成をサポートします。

開発 GPU リソース共有のタイムスライシングサポートを使用したテストと実験用の下位層 GPUT4、RTX シリーズ**製品 GPU** レイテンシー用の専用割り当てを備えた高性能 GPUA100、H100 センシティブな推論ワークロード**マルチインスタンス GPU** パーティション化された GPU により、複数の小型モデルがハードウェアリソースを効率的に共有できます

アロケーション戦略

GPU のスケジューリングは、使用可能なリソースをアドバタイズし、割り当てポリシーを適用する Kubernetes デバイスプラグインを介して発生します。システムは以下をサポートしています。

- 実稼働ワークロードのための排他的な GPU 割り当て
- 開発とテストのためのタイムスライス共有
- 効率的なマルチモデルサービングのための MIG パーティション
- ワークロード配置のノードアフィニティルール

プロジェクトレベルのリソースクォータは、チーム間の公平なアクセスを保証しながら、GPU の独占を防ぎます。自動スケーリングは、推論需要に基づいて GPU の割り当てを調整し、サービス要件を満たしながらコストを最適化します。

統合パターン

Gen AI Builder アプリケーション

Gen AI Builder は、アシスタントとエージェントのモデルエンドポイントを活用します。Model Serving を介して展開された LLM は、テキスト生成機能を提供し、埋め込みモデルは RAG パイプラインをサポートします。

アプリケーションは、クラスターローカル DNS を介してモデルにアクセスします。

```
http://model-name.namespace.svc.cluster.local/v1/chat/completions
```

ナレッジベースパイプライン

ナレッジベースは、ドキュメントのベクトル化とセマンティック検索に埋め込みモデルを利用します。モデルライブラリーは、取り込みおよび取得操作全体で一貫したモデルバージョンを保証します。

パイプライン統合のメリットは次のとおりです。

- 既知のパフォーマンス特性を備えた検証済みの埋め込みモデル
- ライブラリ同期を介した自動モデル更新
- ナレッジベース操作全体で一貫したベクトルディメンション

カスタムアプリケーション

組織は、ビジネス固有の要件に特化したモデルを展開します。サービングインフラストラクチャは、ビジョンモデル、リランキングモデル、カスタムの微調整されたバリアントを含むさまざまなモデルタイプをサポートしています。

外部アプリケーションは、API キー認証を使用して安全なエンドポイントを介してモデルにアクセスし、セキュリティ境界を維持しながら既存のビジネスシステムとの統合を可能にします。

セキュリティとガバナンス

アクセス制御

ロールベースのアクセス制御は、マルチプルのレベルでモデル操作を制御します。

- **ライブラリーアクセス** モデルを登録および承認できる人を制御します
- **展開権限** モデルをサービングに展開できる人を管理します
- **エンドポイントアクセス**モデルエンドポイントを呼び出すことができる人を決定します

エンタープライズ ID プロバイダーとの統合により、コンプライアンス要件の詳細な監査証跡を維持しながら、シングルサインオンが可能になります。

モデルガバナンス

ガバナンスフレームワークにより、検証されたモデルのみが運用環境に到達します。

- セキュリティスキャンは、展開前に脆弱性を特定します
- デジタル署名は、モデルの信頼性と完全性を検証します
- 承認ワークフローは、マルチステークホルダーの検証を強制します
- リテンションポリシーはモデルのライフサイクルと非推奨を管理します

データ保護

すべてのモデル操作は、インフラストラクチャ内のデータ主権を維持します。

- Kubernetes クラスターでのみ実行されるモデル
- 推論データが制御された環境を離れることはありません
- ネットワークポリシーはトラフィックの分離を適用します
- 暗号化は、転送中および保存中のデータを保護します

モニタリングと最適化

パフォーマンスメトリック

包括的なモニタリングは、複数のディメンションにわたってモデルのパフォーマンスを追跡します。

- **推論レイテンシー** エンドツーエンドの要求の処理時間
- **トークンスループット** 言語モデルの生成レート
- **バッチ効率** バッチ機能の利用
- **キャッシュヒット率** 応答キャッシュの効果

これらのメトリックは、バッチサイズの調整、リソース割り当ての調整、キャッシュ戦略の調整を含む最適化の決定を通知します。

リソース使用率

GPU モニタリングは、ハードウェア効率の可視性を提供します。

- 最適化の機会を特定するメモリ使用パターン
- スケーリング要件を示すコンピューティング使用率
- コスト分析のための消費電力
- ハードウェアの状態の温度モニタリング

コスト管理

システムは、コスト帰属のリソース消費を追跡します。

- モデルおよびプロジェクトごとの GPU 時間
- モデルアーティファクトのストレージコスト
- 推論トラフィックのネットワーク転送
- リソースコストに影響を与えるスケーリングパターン

ベストプラクティス

モデル選択

明確な要件分析に基づいてモデルを選択します。

- モデル機能をユースケース要件に合わせる
- リソースの制約とコストへの影響を考慮する
- テストによるパフォーマンス特性の検証
- 将来の参照用にモデル選択の根拠を文書化する

導入戦略

体系的な展開アプローチを実装します。

- 控えめなリソース割り当てから開始します
- 最適化する前に実際の使用量をモニタする
- モデルの更新にカナリア展開を使用する
- 問題のある展開のロールバック機能を維持する

オペレーショナルエクセレンス

高い運用基準を維持します。

- ベースラインパフォーマンスメトリックを確立する
- 運用前に包括的なモニタリングを実装する
- 文書構成の決定事項と手順
- リソースの使用率とコストの定期的なレビュー

はじめに

次の手順でモデル管理を開始します。

1. 設定 → 機能から、ハイブリッドマネージャープロジェクトで**モデル機能を有効にします**
2. **リポジトリルールを構成** して、信頼できるモデルソースを接続する

3. ガイド付き展開ワークフローを使用して **最初のモデルを展開**
4. 統合されたダッシュボードを介した **モニタパフォーマンス**

詳細な実装ガイダンスについては

- *Model Library Configuration*
- *Create InferenceService*
- *GPU Setup Guide*
- *Monitoring InferenceServices*

Hybrid Manager 内のモデル管理は、AI の展開を実験的なプロトタイプから実稼働グレードのサービスに変換し、エンタープライズ機能を提供しながら主権を維持します。

7.6.3 How to Use Gen AI Builder

Using models in Gen AI Builder

ハブクイックリンク *Gen AI how-tos* — **モデルサービング** — **モデルライブラリーアーキテクチャ**

モデルがどのように消費されるか

ハイブリッドマネージャーでは、 **Gen AI Builder** を使用して、**モデルライブラリーアーキテクチャ**から提供されるモデルエンドポイントを使用するアシスタントをビルドできます。

- **Assistant** を作成または編集するときに、使用するモデルを選択します**ステップ 2-アシスタントを作成する** を参照してください。
- ここで利用可能なモデルは **ハイブリッドマネージャー**によって管理されています。モデルライブラリーから取得され、プロジェクトに展開され、内部エンドポイントを介して公開されます。
- これにより、モデル呼び出しが環境内にとどまります。デフォルトでは、**外部 API 呼び出しは** 行われません。

ナレッジベースとパイプライン

Gen AI Builder のナレッジベースには、 **パイプライン**によって生成されたエンベディングを入力できます。

- パイプラインは、ドキュメントを取り込み、**ベクターインデックス**に準備します *Vector Engine concepts* を参照してください。
- ナレッジベースはこれらのパイプラインを参照し、取得拡張生成 RAG に対して照会可能にします。
- アシスタントは、モデル応答とナレッジベースの取得を組み合わせて、組織のデータに統合出力します。

Knowledge bases in the asset library および *Pipelines (hub)* を参照してください。

環境とサービスの検出

Gen AI Builder がハイブリッドマネージャー内で実行される場合

- **サービス検出** と **エンドポイント**は、プラットフォームによって自動的に管理されます。
- 外部 URL を構成する必要はありません。モデルはアシスタント作成 UI に直接表示されます。
- 必要な **環境変数**サービスルーティングまたは認証用は、HM によって注入されます。

これは、アシスタントをビルドすることに集中することを意味します — **ハイブリッドマネージャーは、ワイヤリングモデル、データ、および可観測性を一緒に処理します。**

一般的なタスク

ハブからこれらのハウツーを使用して、ビルドを開始します。

- **ステップ** *2-***アシスタントを作成する**
- **ステップ** *1-***ナレッジベースを作成する** および *Manage Knowledge Bases*
- *View threads*

重要なポイント

ハイブリッドマネージャーでは、モデルとデータは **同じ場所にあります** 。

- **モデル** モデルライブラリーから HM プロジェクトに展開されます。
- **データ**パイプラインを介して取り込まれ、ナレッジベースに保存されます。
- **アシスタント**クラスターからトラフィックを残さずに両方を組み合わせます。

7.6.4 Setup

Deploy AI Factory with Hybrid Manager UI

概要

このガイドでは、ハイブリッドマネージャー HCP Web インターフェイスを介した AI Factory コンポーネントの展開手順を提供します。この展開プロセスには、GPU インフラストラクチャのセットアップ、NVIDIA NIM モデルの展開、および Gen AI アプリケーションの構成が含まれます。

前提条件

インフラストラクチャ要件

AI Factory コンポーネントを展開する前に

- GPU ノードが構成された Kubernetes クラスター
- NVIDIA GPU オペレーターのインストール
- NVIDIA NGC レジストリまたはプライベートレジストリへのアクセス
- モデルプロファイルのオブジェクトストレージエアギャップ展開

GPU ノード構成

GPU ノードが NVIDIA NIM 要件を満たしていることを確認します。

Model Type	NIM Model	GPU Requirements	Container Image
Text Completion	llama-3.3-nemotron-super-49b-v1	4 x L40S	nvcr.io/nim/nvidia/llama-3.3-nemotron-super-49b-v1:1.8.5
Text Embeddings	llama-3.2-nemoretriever-300m-embed-v1	1 x L40S	nvcr.io/nim/nvidia/llama-3.2-nemoretriever-300m-embed-v1:latest
Image Embeddings	nvclip	1 x L40S	nvcr.io/nim/nvidia/nvclip:latest
OCR	paddleocr	1 x L40S	nvcr.io/nim/baidu/paddleocr:latest
Text Reranking	llama-3.2-nv-rerankqa-1b-v2	1 x L40S	nvcr.io/nim/nvidia/llama-3.2-nv-rerankqa-1b-v2:latest

GPU ノードに以下が含まれていることを確認します。

- ラベル `nvidia.com/gpu=true`
- テイント `nvidia.com/gpu`

ステップ 1 レジストリ認証を構成する

インターネットに接続された展開

インターネットアクセスのあるクラスターの場合、NVIDIA NGC 認証を構成します。

1. NVIDIA NGC Portal から NGC API キーを取得します

2. HCP UI を介して認証シークレットを作成します。

- **Project Settings** → **Secrets** に移動します
- NGC_API_KEY を使用して `nvidia-nim-secrets` を作成する
- `ngc-cred` Docker レジストリシークレットを作成します

または、kubectl を使用します。

```
NGC_API_KEY=<your-ngc-api-key>

#  Create runtime secret

kubectl -n default create secret generic nvidia-nim-secrets \
    --from-literal=NGC_API_KEY=${NGC_API_KEY}

kubectl -n default annotate secret nvidia-nim-secrets \
    replicator.v1.mittwald.de/replicate-to=m-.*
```

(次のページに続く)

(前のページからの続き)

```
#  Create image pull secret

kubectl -n default create secret docker-registry ngc-cred \
    --docker-server=nvcr.io \
    --docker-username=$oauthtoken \
    --docker-password=${NGC_API_KEY}

kubectl -n default annotate secret ngc-cred \
    replicator.v1.mittwald.de/replicate-to=m-.*
```

エアギャップ展開

インターネットアクセスのない環境の場合

1. **ミラー NIM イメージ** skopeo を使用して、必要なイメージをプライベートレジストリにコピーします
2. **モデル URL の更新**プライベートレジストリの場所を参照するように HCP を構成する
3. **キャッシュプロファイル**モデルプロファイルをダウンロードしてオブジェクトストレージに保存します
4. **ストレージパスの構成**モデルの展開中にキャッシュされたプロファイルを参照する

詳細な手順については、 *Air-Gapped Configuration* を参照してください。

ステップ 2 モデルライブラリーにアクセスする

Model ライブラリーインターフェイスに移動します。

1. Hybrid Manager コンソールにログインします
2. プロジェクトセレクターからプロジェクトを選択します
3. **AI Factory** → **Model ライブラリー** に移動します

モデルライブラリーには、使用可能な NVIDIA NIM モデルが表示されます。

- **llama-3.3-nemotron-super-49b-v1** 高度な推論とチャット機能 128K コンテキスト
- **llama-3.2-nemoretriever-300m-embed-v1** 高品質のテキスト埋め込み
- **llama-3.2-nv-rerankqa-1b-v2** 多言語クエリドキュメントのリランキング
- **paddleocr** 超軽量 OCR システム
- **nvclip** 画像とテキストのマルチモーダル埋め込み

ステップ 3 モデルサーバークラスターを展開する

導入用モデルの選択

1. モデルライブラリーで **Deploy Model** をクリックします
2. 利用可能なオプションからターゲットモデルを選択します
3. モデル要件とドキュメントリンクを確認します

展開パラメーターの構成

モデルサーバークラスター設定を構成します。

インスタンス構成

- **サーバーインスタンス** デフォルト 1 高可用性のための増加
- **最小インスタンス** 1 サポートされている場合はスケールからゼロの 0
- **最大インスタンス** 負荷要件に基づきます

リソースの割り当て

- **メモリ** モデル要件に基づいて構成します
- テキスト補完モデル 64-128 GB
- 組み込みモデル 32-64 GB
- **CPU** 通常はデフォルト値で十分
- **GPU** 文書化された要件と一致する
- llama-3.3-nemotron 4 GPU
- その他のモデル 1 GPU 通常

スケーリング構成

- **同時要求のしきい値** 自動スケーリングトリガーの構成
- **Scale to Zero** サポートされている場合に有効にします初期リリースでは使用できない場合があります

デプロイモデル

1. 構成の概要を確認します
2. **Deploy** をクリックして展開を開始します
3. モデルサーバークラスタービューでの展開ステータスの監視

展開により作成されるもの

- KServe InferenceService リソース
- リソースが構成された予測子ポッド
- モデルアクセスのサービスエンドポイント

ステップ 4 展開ステータスを監視する

View Model Server クラスター

AI Factory → **Model Server Clusters** に移動して、以下を表示します。

- クラスター表示名
- ロードバランサーのイングレス URI
- 展開ステータス アクティブ/正常、保留中、失敗
- モデルの詳細とタグ
- リソース使用率メトリック

アクセスクラスターの詳細

表示するクラスター名をクリックします。

- 詳細な構成パラメーター
- リアルタイムの健康メトリック
- Grafana ダッシュボードの統合
- 推論レイテンシーチャート
- システム状態インジケーター

ステップ 5API アクセスを構成する

API トークンを生成する

デプロイされたモデルへの外部アクセスの場合

1. **AI Factory** → **API Token Management** に移動します
2. **Create Token** をクリックします
3. トークン参照名を提供します
4. 生成されたトークンを安全に保存する

API トークンにより、次のことが可能になります。

- RAG アプリケーション認証
- 外部サービスの統合
- プログラマティックモデルアクセス

アクセスエンドポイント

モデルは OpenAI 互換のエンドポイントを公開します。

内部アクセス クラスター内

```
http://<service-name>.<namespace>.svc.cluster.local/v1/chat/completions
```

外部アクセス イングレスを使用する場合

```
https://<ingress-url>/v1/chat/completions
```

ステップ 6 Gen AI アプリケーションをビルドする

ナレッジベースの作成

1. **AI Factory** → **Gen AI Builder** → **Knowledge Bases** に移動します
2. データソースを構成します。
   - PostgreSQL データベース
   - ドキュメントリポジトリ
   - API 接続
3. 展開されたサービスから埋め込みモデルを選択します
4. コンテンツを処理して埋め込みを生成する

アシスタントの構成

1. **AI Factory** → **Assistants** に移動します
2. 次を使用して新しいアシスタントを作成します。
   - 名前と指示子
   - モデルエンドポイントの選択
   - ナレッジベースの統合
   - ツール構成
3. アシスタントアプリケーションの展開

詳細な手順については、 *Create Assistant Guide* を参照してください。

ステップ 7 モデルの更新と管理

モデルサーバークラスターの編集

実行中のクラスターを変更するには

1. **Model Server Clusters** に移動します
2. 編集するクラスターを選択します
3. パラメーターを調整します。
   - インスタンス数
   - リソースの割り当て

- スケーリングしきい値

4. 変更を適用しますローリング更新をトリガーします

ローリングアップデート

更新は、次のことを介して可用性を維持します。

- 再起動前の接続ドレイン
- インスタンス間の状態検証
- 失敗時の自動ロールバック

障害対応

一般的な展開の問題

モデルの開始に失敗します

- GPU の可用性が要件と一致することを確認します
- レジストリ認証シークレットの確認
- ポッドイベントとログを確認する

高い推論レイテンシー

- バッチサイズパラメーターを調整する
- GPU 割り当てを増加します
- 負荷分散のためのスケールレプリカ

API トークンの問題

- トークンの有効期限がまだ残っていることを確認します
- ネットワークポリシーの確認
- イングレス構成の確認

詳細な診断については、 *Troubleshooting AI Factory on Hybrid Manager* を参照してください。

追加リソース

リファレンスドキュメント

- *GPU Setup Guide*
- *Create InferenceService*
- *Model Serving Concepts*

NVIDIA モデルドキュメント

- Llama 3.3 Nemotron
- Arctic Embed

- NV-ReRankQA
- PaddleOCR
- NV-CLIP

Prerequisites for AI Factory on Hybrid Manager

インフラストラクチャ要件

Kubernetes Cluster Foundation

AI Factory には、AI ワークロード用の十分なリソースを備えた適切に構成された Hybrid Manager Kubernetes クラスターが必要です。クラスターは GPU スケジューリングをサポートし、さまざまなワークロードタイプに合わせて構成された適切なノードグループが必要です。

クラスター要件

- GPU デバイスプラグインをサポートする Kubernetes 1.27 以降
- GPU ノード管理のためにインストールされた NVIDIA GPU オペレーター
- オーケストレーションコンポーネントに十分な CPU とメモリ
- サービスメッシュ通信をサポートするネットワークポリシー

GPU ノード構成

GPU リソースは、モデル提供操作に不可欠です。ノード構成は、モデル要件および予想されるワークロード特性と一致する必要があります。

GPU ノードの要件

- CUDA 12.1+をサポートする NVIDIA GPU
- `nvidia.com/gpu=true` でラベル付けされた GPU ノード
- 専用スケジューリング用の GPU テイント `nvidia.com/gpu`
- ターゲットモデルサイズに十分な GPU メモリ

モデルリソース要件

Model Type	Example Model	GPU Requirements	Container Image
Text Completion	llama-3.3-nemotron-super-49b	4 x L40S	nvcr.io/nim/nvidia/llama-3.3-nemotron-super-49b-v1:1.8.5
Text Embeddings	llama-3.2-nemoretriever-300m	1 x L40S	nvcr.io/nim/nvidia/llama-3.2-nemoretriever-300m-embed-v1:latest
Image Embeddings	nvclip	1 x L40S	nvcr.io/nim/nvidia/nvclip:latest

次のページに続く

表 5 – 前のページからの続き

Model Type	Example Model	GPU Require-ments	Container Image
OCR	paddleocr	1 x L40S	nvcr.io/nim/baidu/paddleocr:latest
Text Reranking	llama-3.2-nv-rerankqa-1b	1 x L40S	nvcr.io/nim/nvidia/llama-3.2-nv-rerankqa-1b-v2:latest

モデルごとのリソース仕様の詳細については、NVIDIA NIM documentation を参照してください。

レジストリ構成

インターネットに接続された展開

インターネットアクセスのあるクラスターの場合、NVIDIA NGC レジストリ認証を構成してモデルイメージのプルを有効にします。

NGC API キー構成

NVIDIA NGC documentation に従って NGC API キーを取得します。

必要なシークレットを作成します。

```
#  Set your NGC API key

NGC_API_KEY=<your-ngc-api-key>

#  Create model runtime secret

kubectl -n default create secret generic nvidia-nim-secrets \
    --from-literal=NGC_API_KEY=${NGC_API_KEY}

#  Enable secret replication across namespaces

kubectl -n default annotate secret nvidia-nim-secrets \
    replicator.v1.mittwald.de/replicate-to=m-.*
```

イメージプルシークレット構成

イメージプルの Docker レジストリ認証を構成します。

```
#  Create image pull secret

kubectl -n default create secret docker-registry ngc-cred \
    --docker-server=nvcr.io \
    --docker-username=$oauthtoken \
    --docker-password=${NGC_API_KEY}
```

(次のページに続く)

(前のページからの続き)

```
#  Enable secret replication

kubectl -n default annotate secret ngc-cred \
    replicator.v1.mittwald.de/replicate-to=m-.*
```

エアギャップ展開

インターネットアクセスのない環境の場合、ハブガイドを使用してイメージと展開アセットを事前に準備し、ハイブリッドマネージャーでプライベートレジストリアクセスを構成します。

モデルイメージの移行

これらのハブ参照に従って、必要なモデルイメージを準備してプライベートレジストリにミラーリングします。

- プライベートレジストリとイメージガバナンス **モデルライブラリーアーキテクチャ**
- KServe マニフェストと展開フロー *Using NVIDIA NIM in your environment*

モデルレジストリの更新

デフォルトのモデル参照を更新して、プライベートレジストリをポイントします。モデルライブラリー構成と HM API /コンソールパスを操作します。

プロファイルキャッシュ NIM

一部の NVIDIA NIM モデルは、オフライン操作のためにローカルで利用できる必要があるランタイムプロファイルを使用します。プロファイルの検出とキャッシュ戦略については、NVIDIA のドキュメントに従ってください。

- NVIDIA NIM ドキュメント https://docs.nvidia.com/nim/
- ハブ使用パターン *Using NVIDIA NIM in your environment*
- *Air Gapped Cache*

ネットワーク要件

サービスコミュニケーション

AI Factory コンポーネントには、サービス間通信用の特定のネットワーク構成が必要です。

ネットワークポリシー

- モデルサービングポッドとアプリケーション名前空間間のトラフィックを許可する
- 外部モデルエンドポイントアクセスのイングレスを有効にする
- 可観測性のためのサービスメッシュ通信のサポート

DNS 解決

- 内部サービス検出のためのクラスターローカル DNS

- 必要に応じて、パブリックエンドポイントアクセス用の外部 DNS

ストレージアクセス

必要なストレージシステムへのネットワーク接続を確認します。

オブジェクトストレージ

- モデルアーティファクトとプロファイル用の S3 互換ストレージ
- モデルのダウンロード操作に十分な帯域幅
- ストレージアクセスのための IAM ロールまたは資格情報

セキュリティ前提条件

RBAC 構成

AI Factory 操作のロールベースのアクセス制御を構成します。

必要なロール

- プロジェクトレベルの管理のための名前空間管理者
- 展開操作のモデルオペレーター
- モニタリングと可観測性のための読み取り専用アクセス

証明書管理

安全な通信のために SSL 証明書を準備します。

証明書の要件

- モデルエンドポイントエクスポージャの TLS 証明書
- サービスメッシュ通信のための内部 CA
- コンプライアンスのための証明書のローテーション手順

次のステップ

前提条件を満たしたら、次のことに進みます。

1. 詳細な GPU 構成の場合 *GPU Setup Guide*
2. レジストリ統合のための *Model Library Configuration*
3. *Create First InferenceService* 初期モデルを展開する

計画のサポートが必要な場合は、次の場所に相談してください。

- *AI Factory Architecture on Hybrid Manager*
- *Model Serving Concepts*

7.6.5 How to Use Models

Using models in Model Clusters & AIDB

ハブクイックリンク モデルサービング — *Access KServe endpoints*

アプリケーションから

アプリケーションは、 **KServe InferenceServices** を介して展開されたモデルエンドポイントを直接呼び出すことができます。

- **クイックスタート Python)** *Quickstart using inference endpoint*

これらのエンドポイントは、要求を送信し、展開したモデルから予測または世代を取得するための標準の REST API を提供します。

AIDB から SQL パターン

AIDB では SQL からモデルを呼び出し、Postgres 内で直接利用できるようにします。これは、パイプラインにモデルを埋め込んだり、データベース内推論を有効にしたりする場合に役立ちます。

- **OpenAI API 互換性** *Using models via OpenAI-compatible SQL calls*
- **NVIDIA NIM:** *Using NVIDIA NIM in AIDB*

ハイブリッドマネージャーの詳細

モデルがハイブリッドマネージャーを介して展開される場合

- **サービス URL.** 各モデルは、HM プロジェクト内の内部 KServe エンドポイントとして公開されます。URL は、モデルライブラリーまたはモデル提供の詳細ページに表示されます。
- **認証.** エンドポイントはプラットフォームによって保護されています。同じプロジェクト内で実行されているアプリケーションは、それらに直接到達できます。外部から接続する場合は、HM 入力およびプロジェクトスコープの資格情報を使用して認証を構成します。
- **オブザーバビリティ.** これらのエンドポイントからの要求とログは HM オブザーバビリティに流れ、使用状況メトリック、レイテンシー、エラー追跡を提供します。

重要なポイント

- **アプリケーション** → REST を介して KServe 推論エンドポイントを呼び出します。
- **AIDB** → SQL パターンを使用して、OpenAI スタイルの呼び出しや NVIDIA NIM などのモデルをインラインで呼び出します。
- **Hybrid Manager** はサービスの検出、認証、およびオブザーバビリティを処理するため、構築に集中できます。

7.6.6 Operate and Observe

Observability (AI Factory on HM)

ハブクイックリンク *Model Serving observability — Gen AI Builder on Hybrid Manager*

ハイブリッドマネージャーの可観測性

ハイブリッドマネージャーに AI Factory コンポーネントを展開すると、プラットフォームは **Model Serving** KServe と **Gen AI Builder** の両方のワークロードの統合されたオブザーバビリティを提供します。

モデルサービング KServe

- プラットフォームのダッシュボードとログを使用して、**InferenceServices** と GPU ワークロードを監視します。
- メトリックには、要求レイテンシー、スループット、GPU / CPU 使用率、およびエラー率が含まれます。
- *Monitor InferenceService* を参照してください

Gen AI Builder アシスタント & スレッド

- 各 **アシスタントスレッド**はシステムに記録されるため、入力、取得コンテキスト、出力をトレースできます。
- スレッドデータは、ハイブリッドマネージャー内の Gen AI Builder UI で表示できます。
- 使用メトリック実行数、レイテンシー、エラーは、プラットフォームの可観測性スタックに統合されています。
- *View threads (hub)* を参照してください

メトリックとログへのアクセス

- **ダッシュボード.** ハイブリッドマネージャーは、ビルトインの可観測性ダッシュボード Grafana を介して AI Factory メトリックを表示します。
- **ログ** `kubectl logs` または HM UI ログビューアを使用して、Gen AI Builder ポッド、KServe InferenceServices、およびレトリーバージョブを検査します。
- **トレーシング** OpenTelemetry ドライバーは、より深いトレースのために Gen AI Builder に含まれています。 *Gen AI observability drivers (hub)* を参照してください。

一般的なコマンド

クラスターレベルのトラブルシューティングまたはカスタムダッシュボードの場合、メトリックとログを直接取得することもできます。

```
#  Check logs for a running InferenceService

kubectl logs -n <project-namespace> svc/<model-service-name>
```

(次のページに続く)

(前のページからの続き)

```
#  Port-forward Grafana if running locally

kubectl port-forward -n observability svc/grafana 3000:3000
```

重要なポイント

- **モデルサービス** → 完全なメトリックを使用して KServe サービスとして監視されます。
- **Gen AI Builder** → 使用状況と監査のために追跡されるアシスタントとスレッド。
- **Hybrid Manager** は、統合された可観測性レイヤーを提供します。個別の監視スタックを構成する必要はありません。

Troubleshooting AI Factory on Hybrid Manager

診断の概要

このガイドでは、ハイブリッドマネージャー環境内の AI Factory コンポーネントの体系的なトラブルシューティング手順を提供します。問題は通常、インフラストラクチャの問題、モデル提供の失敗、およびアプリケーションレベルのエラーの 3 つのカテゴリに分類されます。

クイックリファレンスリンク

- *Frequently Asked Questions - AI Factory on Hybrid Manager*
- *GPU Setup Guide*
- *InferenceService Creation*
- *Gen AI Builder on Hybrid Manager*

インフラストラクチャの問題

GPU リソースの問題

症状

- InferenceService ポッドは保留中のままです
- 「不十分な nvidia.com/gpu」イベント
- モデルの初期化タイムアウト

診断手順

```
#  Check GPU node availability

kubectl get nodes -l nvidia.com/gpu=true
```

(次のページに続く)

(前のページからの続き)

```
#  Verify GPU resource allocation

kubectl describe node <gpu-node-name> | grep -A 5 "Allocated resources"

#  Check GPU driver status

kubectl logs -n gpu-operator nvidia-driver-daemonset-<pod>
```

一般的な解決策

GPU ラベルの欠落 GPU を備えたノードは適切にラベル付けする必要があります。

```
kubectl label nodes <node-name> nvidia.com/gpu=true
```

GPU テイントの問題専用 GPU スケジューリングのテイント構成を確認します。

```
kubectl taint nodes <node-name> nvidia.com/gpu=NoSchedule
```

ドライバーの互換性 NVIDIA ドライバーバージョンが NIM コンテナの CUDA 要件と一致することを確認します。gpu-operator 名前空間のドライバーログに初期化エラーがないか確認します。

ストレージアクセス障害

症状

- モデルのダウンロードの失敗
- “ImagePullBackOff”ステータス
- エアギャップ環境でのプロファイルキャッシュエラー

診断手順

```
#  Check secret configuration

kubectl get secret nvidia-nim-secrets -n default -o yaml

#  Verify image pull secret

kubectl get secret ngc-cred -n <namespace> -o yaml

#  Test registry connectivity

kubectl run test-pull --image=nvcr.io/nim/nvidia/nvclip:latest --dry-run=client
```

解決戦略

レジストリ認証 認証が失敗した場合は、NGC 資格情報を再作成します。

```
kubectl delete secret ngc-cred -n default
kubectl create secret docker-registry ngc-cred \
  --docker-server=nvcr.io \
  --docker-username=$oauthtoken \
  --docker-password=<NGC_API_KEY> \
  -n default
```

エアギャップ環境オブジェクトストレージのプロファイルキャッシュの可用性と、モデル展開仕様の正しいパス構成を確認します。

モデル提供の失敗

InferenceService の準備ができていません

症状

- InferenceService は「NotReady」ステータスを示します
- 予測ポッドのクラッシュまたは再起動
- ヘルスチェックの失敗

調査コマンド

```
#  Check InferenceService status

kubectl get inferenceservice <name> -n <namespace>

#  Examine detailed conditions

kubectl describe inferenceservice <name> -n <namespace>

#  Review pod logs

kubectl logs <predictor-pod> -n <namespace> -c kserve-container
```

一般的な原因と修正

メモリ不足 NIM モデルには、大量のメモリが必要です。ポッドリソース要求を確認します。

```
kubectl describe pod <predictor-pod> -n <namespace> | grep -A 3 "Requests"
```

必要に応じて、InferenceService 仕様のメモリ割り当てを増加します。

モデルの読み込みタイムアウト大規模なモデルは、デフォルトの初期化タイムアウトを超過する場合があります。InferenceService 構成の readness プローブ設定を調整します。

プロファイルの不一致キャッシュされたプロファイルが GPU アーキテクチャと一致することを確認します。互換性のあるプロファイルをリストし、キャッシュの内容を確認します。

高い推論レイテンシー

症状

- 応答時間が SLA 要件を超過
- 予想を下回るトークン生成レート
- GPU の十分な使用率

パフォーマンス分析

```
#  Monitor GPU utilization

kubectl exec <predictor-pod> -n <namespace> -- nvidia-smi

#  Check request metrics

kubectl port-forward -n <namespace> svc/<service-name> 9090:9090

#  Access metrics at localhost:9090/metrics
```

最適化アプローチ

バッチサイズチューニング ServingRuntime 構成のバッチ処理パラメーターを調整して、スループットを向上させます。

レプリカのスケーリング負荷を分散するために InferenceService レプリカを追加します。

```
kubectl scale inferenceservice <name> -n <namespace> --replicas=3
```

リソース割り当て GPU メモリ割り当てがモデル要件と一致することを確認します。GPU メモリが不十分であると CPU フォールバックが強制され、パフォーマンスに重大な影響を与えます。

Gen AI アプリケーションの問題

ナレッジベースの取得の失敗

症状

- 空または無関係の検索結果
- 埋め込み生成エラー
- Vector データベースの接続障害

診断プロセス

```
#  Check embedding model status

kubectl get inferenceservice <embedding-model> -n <namespace>

#  Verify PostgreSQL connectivity

kubectl exec -it <app-pod> -n <namespace> -- pg_isready -h <postgres-host>

#  Review application logs

kubectl logs <gen-ai-app-pod> -n <namespace>
```

解決手順

埋め込みモデルの問題埋め込みモデルの展開とエンドポイントのアクセス可能性を確認します。インデックス作成と取得の間で一貫した埋め込みディメンションを保証します。

データベース接続 PostgreSQL サービスの可用性と pgvector 拡張機能のインストールを確認します。アプリケーション構成で接続資格情報を確認します。

アシスタントの応答エラー

症状

- アシスタントは応答の生成に失敗します
- ツール呼び出しの失敗
- スレッドの状態の不一致

調査方法

詳細な実行トレースについては、スレッドログにアクセスします。

```
#  Query thread history (application-specific)

kubectl exec <app-pod> -n <namespace> -- cat /app/logs/threads.log
```

一般的な解決策

モデルエンドポイントは使用できません アシスタント設定で LLM InferenceService のステータスとエンドポイント構成を確認します。

ツール統合の失敗外部 API の接続と認証を確認します。ツール定義が API 仕様と一致することを確認します。

コンテキストウィンドウの超過取得チャンクサイズを削減するか、大規模なドキュメントのコンテキスト管理戦略を実装します。

ログ分析

ログの場所

AI Factory コンポーネントは、複数のレベルでログを生成します。

システムログ

- Kubernetes イベント `kubectl get events -n <namespace>`
- ノードログ GPU ノードの`/var/log/messages` または journalctl

アプリケーションログ

- InferenceService 予測ポッドコンテナログ
- Gen AI アプリケーション アプリケーションポッドログ
- モデルライブラリー HM コントロールプレーンログ

メトリックとモニタリング

- Prometheus メトリック HM モニタリングスタックから利用可能
- カスタムダッシュボード HM コンソールの Grafana を介してアクセスできます

ログ集計

包括的な分析のための集中ロギングを構成します。

```
#  Stream logs from multiple components

kubectl logs -f -l app=inference-service -n <namespace> --all-containers

#  Export logs for analysis

kubectl logs <pod> -n <namespace> --since=1h > diagnostic.log
```

アラート設定

クリティカルアラート

重要な状態のモニタリングアラートを構成します。

GPU の可用性 GPU ノードが使用不可になった場合、または GPU の割り当てが失敗した場合のアラート。

モデルの状態 InferenceService の準備状況と再起動頻度をモニタします。

パフォーマンスの低下推論レイテンシーのパーセンタイルとトークン生成レートを追跡します。

リソースの枯渇メモリ圧力、GPU メモリの飽和、またはストレージ容量に関するアラート。

アラート統合

HM アラートマネージャー構成を介して、アラートを組織の通知システムと統合します。

エスカレーションプロシージャ

サポートリソース

内部のトラブルシューティングが不十分であることが判明した場合

1. **ドキュメントレビュー**
   - *AI Factory Hub* に相談します
   - モデル固有のドキュメントを確認します
2. **コミュニティリソース**
   - モデル固有の問題に関する NVIDIA NIM ドキュメント
   - インフラストラクチャの問題を提供するための KServe コミュニティ
3. **EDB サポート**
   - HM サポートツールを使用して診断バンドルを収集する
   - 関連するログ、構成、エラーメッセージを含めます
   - 参照固有のコンポーネントバージョンと展開仕様

診断情報の収集

包括的な診断情報を準備します。

```
#  Generate support bundle

kubectl cluster-info dump --output-directory=./cluster-dump

#  Collect AI Factory specifics

kubectl get inferenceservice -A -o yaml > inferenceservices.yaml
kubectl get pods -A -l serving.kserve.io/inferenceservice -o wide > serving-pods.txt
kubectl describe nodes -l nvidia.com/gpu=true > gpu-nodes.txt
```

予防措置

定期的な健康診断

プロアクティブなモニタリングを実装します。

- 毎週の GPU ドライバーとオペレーターのステータス検証
- 毎日の推論サービスの状態評価
- 継続的なパフォーマンスベースライン追跡

キャパシティプランニング

リソースの傾向を監視します。

- GPU 使用率パターン
- メモリ消費量の増加
- ストレージ使用量の予測
- リクエスト量の傾向

アップデート管理

構成要素通貨を維持します。

- NVIDIA NIM モデルの更新を追跡します
- セキュリティアドバイザリの監視
- 更新のためのメンテナンス時間枠を計画する
- 非実稼働環境でのテスト更新

第8章　Analytics in Hybrid Manager

EDB Hybrid ManagerHM に埋め込まれている分析機能を理解して活用するための中心的なリソースへようこそ。

ハブクイックリンク *Analytics Hub* — **コアコンセプト** — **関連するハウツー**

ハイブリッドマネージャーは、次の統合サポートを提供します。

- Apache Iceberg ®およびカタログサービス
- デルタレイクのクエリー
- EDB Postgres 分散 PGD を使用した階層テーブル
- オブジェクトストレージの高速分析のためのレイクハウスクラスター
- 分析スタックの統合管理と可観測性

これらの機能を使用することにより、運用オーバーヘッドを削減して、Postgres で最新の費用対効果の高い分析アーキテクチャを構築できます。ハイブリッドマネージャーで完全に管理できます。

8.1 ハイブリッドマネージャーでアナリティクスを使用する理由

EDB Hybrid Manager は、分析を統合して次のことを支援します。

- **統合された可視性の獲得** HM が管理する EDB Postgres フリート全体の主要なメトリックと運用傾向を表示します。
- **パフォーマンスの最適化**パフォーマンスのボトルネックを特定し、HM 管理データベースとレイクハウスクラスター内のクエリ動作を理解します。
- **リソースを効率的に管理する**分析ワークロードを含む、HM 管理インスタンスのリソース消費を監視および調整します。
- **オペレーションの状態の強化**分析コンポーネントと基になるデータベースインフラストラクチャの可用性とステータスを追跡します。
- **データアーキテクチャの簡素化** EDB Postgres Lakehouse クラスターのプロビジョニングと管理、PGD での階層テーブルの実装、およびオブジェクトストレージとの統合。すべて HM からです。

8.2 コアコンセプトと機能

8.2.1 Apache Iceberg ®

ハイブリッドマネージャーでは、Iceberg は次の目的で使用されます。

- Lakehouse クラスターによって照会される、Iceberg 形式で大規模な分析テーブルを管理します。
- Iceberg 形式のオブジェクトストレージへのコールドパーティションの PGD オフロードをサポートします。
- カタログ管理テーブルを介して外部ツール Spark、Trino などとの相互運用性を有効にします。
- HM 管理サービス Lakekeeper または外部カタログを介したカタログ管理を促進します。

ハイブリッドマネージャーの *Apache Iceberg* ®

次のステップ *Configure an Iceberg catalog connection*

8.2.2 デルタレイク

ハイブリッドマネージャーは、Delta Lake テーブルの照会をサポートしています。

- Lakehouse クラスターは、オブジェクトストレージに保存されている既存の Delta Lake テーブルを照会できます。
- HM 管理のレイクハウスをデルタベースのパイプライン Spark、Databricks などと統合します。
- レイクハウスクラスターを介して既存のデルタデータレイクの高速分析を有効にします。

Delta Lake in Hybrid Manager

次のステップ *Read Iceberg/Delta with or without a catalog*

8.2.3 PGD を使用した階層テーブル

階層テーブルにより、大規模な時系列データセットの効率的な管理が可能になります。

- PGD の AutoPartition は、時間ベースのパーティショニングを自動化します。
- コールドパーティションは、オブジェクトストレージで Iceberg 形式にオフロードされます。
- Lakehouse クラスターは、ホットデータとコールドデータにわたるクエリアクセスを提供します。
- ハイブリッドマネージャーは、階層化テーブルをエンドツーエンドで構成および監視するのに役立ちます。

ハイブリッドマネージャーの階層化テーブル

次のステップ ハイブリッドマネージャーの階層化テーブル

8.2.4 レイクハウスクラスター

Lakehouse クラスターは、HM の管理対象コンピューティングノードです。

- オブジェクトストレージでのスケーラブルなベクトル化されたクエリ実行を提供します。
- Iceberg と Delta Lake の両方のフォーマットをサポートします。
- カタログ HM 管理または外部との統合。
- より広範な HM エコシステム PGD オフロード、階層テーブル、BI 統合との統合。

EDB Postgres Lakehouse Clusters on Hybrid Manager

次のステップ *Create a Lakehouse cluster*

8.3 これらの機能がどのように連携するか

- **データのオフロード** PGD クラスターは、コールドデータを Iceberg 形式のオブジェクトストレージにオフロードします。
- **カタログサービス** HM はカタログと統合して、テーブルのメタデータを管理します。Lakehouse と PGD は、これらのカタログを介して照会します。
- **レイクハウスクラスター**管理対象コンピューティングノードは、オフロードされたデータでスケーラブルな分析クエリを実行します。
- **Delta Lake 統合** Lakehouse クラスターは、インプレースにある既存の Delta Lake テーブルも照会します。
- **統合管理** Hybrid Manager は、分析アーキテクチャをプロビジョニング、構成、監視、および最適化するための単一のコントロールプレーンを提供します。

8.4 はじめに: クイックスタートチェックリスト

このチェックリストに従って、ハイブリッドマネージャー分析機能の適用を開始します。

- *Analytics concepts (hub)* —コアパターンと機能を理解します。
- *Create a Lakehouse cluster* —Lakehouse クラスターを展開して、オブジェクトストレージでのスケーラブルなクエリーを有効にします。
- *Configure an Iceberg catalog connection* —HM 管理または外部カタログを接続して、Iceberg テーブルを管理します。
- *Read with or without a catalog* — Lakehouse クラスターから Iceberg および Delta Lake テーブルを照会します。
- *Read/write without a catalog* — 開発とテストのためのファイルベースの直接アクセス。
- *Explore Persona-Based Guidance* — DBA、DevOps / SRE、データサイエンティスト、および開発者向けに調整された推奨事項。
- *Explore architectures and examples* — 一般的な分析問題を解決するための概念とパターン。
- *Analytics How-To Index* — 構成と操作のステップバイステップガイド。

8.5 さらに探索する

- *Analytics concepts (hub)*
- ハイブリッドマネージャーの *Apache Iceberg* ®
- *Delta Lake in Hybrid Manager*

- ハイブリッドマネージャーの階層化テーブル
- *EDB Postgres Lakehouse Clusters on Hybrid Manager*
- *Persona-Based Guide*
- *Architectures and examples*
- *Analytics How-To Index*

これらの機能を活用することにより、ハイブリッドマネージャーは、 EDB Postgres 展開全体で最新の柔軟でパフォーマンスの高い分析アーキテクチャを構築できます。すべてが統合された運用効率の高いエクスペリエンスを通じて。

8.5.1 Analytics Concepts in Hybrid Manager

Hybrid Manager を使用すると、 EDB Postgres の最新のデータレイクハウスと分析ソリューションを構築できます。

ハブクイックリンク *Analytics Hub* — コアコンセプト — 関連するハウツー

このページでは、ハイブリッドマネージャーで適用される主要な概念を紹介します。背景に *Analytics Hub* の詳細な説明へのリンクがあります。

アーキテクチャの概要

ハイブリッドマネージャーは、次のレイヤーを統合します。

- スケーラブルな分析コンピューティングのための**レイクハウスクラスター**
- コスト効率の高い分析データストレージのための **Object Storage**
- トランザクションワークロードと自動階層テーブルオフロード用の **PGD クラスター**
- メタデータと相互運用性のための **カタログサービス** HM 管理または外部
- **PGAA および PGFS** が Postgres をデータレイクにブリッジする

このアーキテクチャにより、Postgres ファーストとマルチエンジンのデータレイクハウスデザインの両方が可能になります。

一般的な紹介については、 *EDB Postgres Lakehouse Clusters on Hybrid Manager* を参照してください。

ハイブリッドマネージャーの##レイクハウスクラスター

Lakehouse クラスターは、ハイブリッドマネージャーでスケーラブルな分析コンピューティングを提供します。

- HM を介してプロビジョニングおよび管理
- ベクトル化されたクエリ実行のための PGAA 拡張機能を装備
- オブジェクトストレージ Iceberg、Delta Lake のデータを照会するように設計
- 外部ツール Spark、Trino との相互運用性

レイクハウスクラスターは、ハイブリッドマネージャーの分析の中心です。

詳細はこちら *EDB Postgres Lakehouse Clusters on Hybrid Manager*

ハイブリッドマネージャーの Apache Iceberg ®

Hybrid Manager は、オフロードデータと分析データのプライマリテーブル形式として Iceberg を統合します。

- PGD クラスターは、階層化テーブルの Iceberg 形式にデータをオフロードします
- Lakehouse クラスターは Iceberg テーブルを効率的に照会します
- カタログサービス Lakekeeper または外部は、Iceberg テーブルのメタデータを管理します

Iceberg は、HM のレイクハウスアーキテクチャのためのオープンで信頼性の高い基盤を提供します。

概念の概要 **ハイブリッドマネージャーの** *Apache Iceberg* ®

ハイブリッドマネージャーの## Delta Lake

Delta Lake は、さらなる柔軟性を提供します。

- Lakehouse クラスターは、オブジェクトストレージの既存の Delta Lake テーブルを照会できます
- Spark / Databricks パイプラインを使用する環境で一般的な
- HM の Delta Lake のサポートは現在読み取り専用です

ハイブリッドマネージャーは、ETL なしで Delta Lake データの Postgres ベースの分析を有効にします。

概念の概要 *Delta Lake in Hybrid Manager*

ハイブリッドマネージャーの階層化テーブル

階層テーブルは、大規模な時系列または履歴データセットの自動ライフサイクル管理を提供します。

- PGD クラスターは BDR AutoPartition を使用してパーティションを管理します
- 古いパーティションは、Iceberg 形式のオブジェクトストレージに自動的にオフロードされます
- Lakehouse クラスターと PGD 親テーブルは、ホットデータとコールドデータの両方を照会できます

階層テーブルは、ハイブリッドマネージャーでトランザクションワークロードと分析ワークロードを結合するための重要な最適化パターンです。

概念の概要 **ハイブリッドマネージャーの階層化テーブル**

ハイブリッドマネージャーの PGAA および PGFS

PGAA アナリティクスアクセラレーター

PGAA 拡張機能は以下を提供します。

- ベクトル化されたクエリーの実行
- PGFS を介したオブジェクトストレージとの統合
- Iceberg および Delta Lake テーブルの照会のサポート

- カタログの統合

PGAA は、レイクハウスクラスターと PGD オフロードパイプラインの両方を強化します。

PGFS Postgres ファイルシステム

PGFS はオブジェクト格納場所を定義します。

- PGAA がオブジェクトストレージのデータの読み取り/書き込みに使用します
- S3 互換ストレージ、GCS などをサポート
- PGD クラスターと Lakehouse クラスターの両方で構成

PGFS は、データレイク統合のためのシンプルですが重要なビルディングブロックです。

これらの概念がハイブリッドマネージャーでどのように適合するか

Layer	Role
PGD Clusters	Transactional layer, source for Tiered Tables
Lakehouse Clusters	Analytical compute, queries across Iceberg/Delta and Postgres
Object Storage	Cost-efficient analytical data storage
Catalog Services	Centralized metadata for Iceberg tables
PGAA + PGFS	Bridge between Postgres and object storage

次のトピック

Solving Analytics Problems in Hybrid Manager

8.5.2 Delta Lake in Hybrid Manager

Delta Lake は、データレイクに ACID トランザクションと信頼性をもたらすオープンソースのテーブル形式です。

ハブクイックリンク *Analytics Hub* — **コアコンセプト** — **関連するハウツー**

Hybrid ManagerHM は、Delta Lake 機能を EDB Postgres 展開に統合し、Lakehouse クラスターがオブジェクトストレージに保存されている Delta Lake テーブルを照会できるようにします。

Delta Lake の一般的な概要については、 *Delta Lake in Hybrid Manager* を参照してください。

ハイブリッドマネージャーで Delta Lake を使用する理由

- **既存のデルタレイクを照会する** 高速な Postgres SQL を使用して、Delta Lake フォーマットで既に構築されているデータレイクを活用します。
- **分析パイプラインの簡素化**不要な ETL を回避します-Postgres から適切な場所に Delta Lake テーブルを照会します。

- **広範なエコシステムの統合** HM 管理の Postgres を、Spark、Trino、Flink、およびその他のツールによって生成された Delta Lake データに接続します。
- **費用対効果の高いレイクハウスアーキテクチャ**大規模なデータセットをオブジェクトストレージに保存し、レイクハウスクラスターを介してクエリします。

重要な用語とアーキテクチャの概要

ハイブリッドマネージャー PGFS、PGAA、レイクハウス、アナリティクスオフロードなどで使用される分析用語の定義については、*Analytics concepts (hub)* を参照してください。

ハイブリッドマネージャーで Delta Lake を使用するのはどのような場合ですか?

次の場合にハイブリッドマネージャーで Delta Lake を使用します。

- デルタレイクフォーマットで構築された **既存のデータレイクを照会**、ETL またはデータの重複なし。
- **Postgres とデータレイクエコシステムの統合** - Postgres SQL クライアントから Delta Lake テーブルを照会します。
- 運用中の Postgres データとデータレイクデータにわたって**統合分析を有効にします** 。
- 使い慣れた Postgres ツールを使用して、Delta Lake コンテンツの**サポート BI ツールとアドホッククエリー** 。
- 大規模なデルタデータセットでのスケーラブルで高速な SQL のために、**レイクハウスクラスターを活用します** 。

重要: PGAA は現在、Delta Lake テーブルで **読み取り専用クエリー**をサポートしています。PGAA を介したデルタテーブルの書き込みまたは更新はサポートされていません。

ハイブリッドマネージャーの Delta Lake の主要機能

既存の Delta Lake テーブルの照会

内容 オブジェクトストレージに保存されている Delta Lake テーブルで SQL クエリーを実行します。

理由 BI ツールと Postgres ユーザーが既存の Delta Lake データを複製または移動せずに照会できるようにします。

方法 Lakehouse クラスターで PGFS ストレージの場所と PGAA 外部テーブルを定義します。

ここで Delta Lake フォーマット `_delta_log` + Parquet ファイルを使用した S3 互換オブジェクトストレージ。

ハウツー *Read Iceberg/Delta with or without a catalog*

Postgres + Delta Lake の統合の簡素化

内容 HM Lakehouse クラスターを、Spark またはその他のツールで作成した Delta Lake テーブルに接続します。

理由運用システムとデータレイクシステム全体で統合されたレポートと分析を構築します。

方法 Delta Lake パスを指す PGFS ストレージ場所と PGAA リーダーテーブルを作成します。

ここで Delta Lake パイプラインが使用する共有オブジェクトストレージの場所。

ハウツー *Configure PGFS*

統合 SQL ベースの分析のサポート

内容 運用データと Delta Lake データの両方で Postgres SQL クエリーを有効にします。

理由アプリケーション開発者、データサイエンティスト、および BI ユーザーが、複雑なツールを使用せずにデータレイクコンテンツを照会できるようにします。

方法 Lakehouse クラスターで PGAA リーダーテーブルを使用します。オプションで Postgres データと結合します。

ここで オブジェクトストレージの Delta Lake テーブル+ Lakehouse クラスターまたは FDW / dblink 経由の Postgres テーブル。

ハウツー *Read Iceberg/Delta with or without a catalog*

ハイブリッドマネージャーでの Delta Lake のスタートガイド

ハイブリッドマネージャーで Delta Lake の使用を開始するには

1. *Provision a Lakehouse cluster* 。
2. Delta Lake オブジェクトストレージを指す *Configure PGFS* 。
3. Lakehouse クラスターで pgaa 拡張機能を有効にします。
4. Delta Lake パスの PGAA リーダーテーブルを作成します。
5. 標準の Postgres クライアントを使用して Delta Lake テーブルを照会します。

関連するハウツー

- *Read Iceberg/Delta with or without a catalog*
- *Configure PGFS*

次のトピック

ハイブリッドマネージャーの階層化テーブル

8.5.3 Apache Iceberg ® in Hybrid Manager

Apache Iceberg ®は、オブジェクトストレージに保存される大規模な分析データセット用のオープンテーブル形式です。

ハブクイックリンク *Analytics Hub* — **コアコンセプト** — **関連するハウツー**

Hybrid ManagerHM は、Iceberg 機能を EDB Postgres 展開に統合し、次のことを可能にします。

- Lakehouse クラスターを介したオブジェクトストレージの効率的なクエリー

- PGD クラスターから Iceberg 形式への構造化データのオフロード
- Iceberg テーブルの集中カタログ管理
- 外部分析エンジン Spark、Trino、Flink などとの相互運用性

Iceberg の一般的な概要については、 **ハイブリッドマネージャーの** *Apache Iceberg* ® を参照してください。

ハイブリッドマネージャーで Iceberg を使用する理由

Apache Iceberg ®は、大規模なデータセットの分析のためのオープンで信頼性の高いパフォーマンスの基盤を提供します。ハイブリッドマネージャーを介して使用すると、次のことができます。

- ベクトル化された実行を使用して、Lakehouse クラスターから Iceberg テーブルを高いパフォーマンスで照会します。
- トランザクション PGD データを Iceberg 形式にオフロードして、費用効率の高い階層ストレージと長期分析を実現します。
- HM を介して Iceberg カタログを一元管理するか、外部カタログに接続します。
- Iceberg 形式を使用して、Postgres と他のツール Spark、Trino、Flink 間でデータをシームレスに共有します。

重要な用語とアーキテクチャの概要

ハイブリッドマネージャー PGFS、PGAA、レイクハウス、カタログ、AutoPartition、Analytics Offload などで使用される分析用語の定義については、*Analytics concepts (hub)* を参照してください。

ハイブリッドマネージャーで Iceberg を使用する必要があるのはどのような場合ですか?

次の場合に Iceberg をハイブリッドマネージャーで使用します。

- **費用対効果の高い方法で PGD データをアーカイブし、Postgres またはその他のツールでクエリします。
- **データレイクハウスアーキテクチャ**を HM 管理または外部の Iceberg カタログと統合します。
- ETL を使用しないラージオブジェクトストレージデータセットで**アドホック分析を有効にします** 。
- **階層テーブルを実装**して、PGD で大規模な時系列データセットとストレージライフサイクルを管理します。
- Iceberg 形式を使用して **Postgres と外部データ処理**ツールを統合します。

ハイブリッドマネージャーの Iceberg の主要機能

既存の Iceberg テーブルのクエリー

内容オブジェクトストレージに既に保存されている Iceberg テーブルで SQL クエリーを実行します。

理由 ETL または重複なしで他のツール Spark、Trino、Flink、PGD オフロードで作成されたデータを再利用できます。

方法 Lakehouse クラスターを使用して、Iceberg データを指す PGAA 外部テーブルを定義します。

ここでファイルベースまたはカタログ管理の S3 互換オブジェクトストレージの Iceberg テーブル。

ハウツー *Read Iceberg/Delta with or without a catalog*

Iceberg カタログの統合

内容 Lakehouse および PGD ノードを Iceberg カタログ HM 管理または外部に接続します。

理由 Iceberg テーブルのメタデータを一元管理し、一貫したセマンティクスでクエリを実行します。

方法 PGAA ファンクション `pgaa.add_catalog` 、`pgaa.attach_catalog` を使用して、カタログを接続およびインポートします。

ここで HM 管理の Lakekeeper カタログ、またはサードパーティのカタログ Glue、Nessie、Polaris。

ハウツー *Configure an Iceberg catalog connection*

注 階層テーブル または **PGD オフロード**を使用する場合、相互運用性とライフサイクル管理のためにカタログ管理の Iceberg テーブルを使用することを強くお勧めします。

PGD データの Iceberg へのオフロード

内容 PGD トランザクションデータを分析ストレージ用に Iceberg 形式にオフロードします。

理由データのクエリー可能を維持しながら、ストレージコストを最適化し、PGD 操作負荷を削減します。

方法 PGFS の場所および/または Iceberg カタログを構成し、PGD テーブルで分析レプリケーションを有効にします。

ここでオブジェクトストレージにオフロードされた Iceberg テーブル、Lakehouse または他のツールでクエリ可能。

ハイブリッドマネージャーの階層化テーブル の概念を学習する

ハイブリッドマネージャーでの Iceberg のユースケース

- クエリアクセスを維持しながら PGD データを Iceberg 形式でアーカイブする
- HM 管理カタログを使用した一元化データレイクハウスアーキテクチャ
- Lakehouse クラスターを介した大規模なデータのアドホック分析
- PGD データセットの階層化ストレージとライフサイクル管理の実装
- Iceberg 形式を使用して PGD オフロードデータを Spark、Trino、Flink パイプラインに公開する

関連 *Analytics concepts (hub)*

ハイブリッドマネージャーでの Iceberg のスタートガイド

ハイブリッドマネージャーで Iceberg の使用を開始するには

1. *Provision a Lakehouse cluster* 。
2. *Configure an Iceberg catalog connection* HM 管理または外部。

3. PGD と統合する場合、 ハイブリッドマネージャーの階層化テーブル を学習します。
4. *Read Iceberg/Delta with or without a catalog* 。
5. 標準の Postgres クライアントを使用して分析クエリを実行します。

関連するハウツー

- *Configure an Iceberg catalog connection*
- *Read Iceberg/Delta with or without a catalog*

次のトピック

Delta Lake in Hybrid Manager

8.5.4 EDB Postgres Lakehouse Clusters on Hybrid Manager

EDB Postgres Lakehouse クラスターは、オブジェクトストレージでオープンレイクハウステーブルフォーマット Apache Iceberg ®、 Delta Lake を照会するための高速分析コンピューティングを提供します。

ハブクイックリンク *Analytics Hub* — コアコンセプト — 関連するハウツー

Hybrid ManagerHM を使用すると、コンソールと API を介してレイクハウスクラスターをプロビジョニングおよび管理し、既存の PGD クラスターおよびより広範な分析エコシステムと統合できます。

EDB Postgres Lakehouse アーキテクチャの基本的な概念については、*EDB Postgres Lakehouse Clusters on Hybrid Manager* を参照してください。

ハイブリッドマネージャーで Lakehouse クラスターを使用する理由

- **マネージド分析コンピューティング** HM でレイクハウスクラスターをプロビジョニングして、オブジェクトストレージで高速クエリーを実行します。
- **コンピューティングとストレージの分離**データをオブジェクトストレージに保持し、必要に応じて分析コンピューティングをスケーリングします。
- **PGD との統合** PGD 階層テーブルおよびオフロードデータでレイクハウスクラスターを使用します。
- **相互運用性** Lakehouse クラスターは、他の分析エンジンと共有している Iceberg および Delta Lake テーブルを照会できます。
- **使いやすい Postgres インターフェイス** EDB Postgres Lakehouse クラスターで標準の SQL クライアントとツールを使用します。

重要な用語とアーキテクチャの概要

ハイブリッドマネージャー PGFS、PGAA、レイクハウスクラスター、アナリティクスオフロードなどで使用されるコア分析用語の定義については、*Analytics concepts (hub)* を参照してください。

ハイブリッドマネージャーでレイクハウスクラスターを使用する必要がある場合

次の場合にハイブリッドマネージャーでレイクハウスクラスターを使用します。

- オブジェクトストレージに保存されている大規模なデータセットで**高速でスケーラブルな分析** を実行します。
- **PGD 階層テーブル**またはオフロードされた PGD トランザクションデータからのクエリデータ。
- **Postgres を外部データレイクと統合します** Iceberg または Delta Lake に構築されています。
- データレイクコンテンツへの Postgres SQL アクセスで、**サポート BI ツールとアドホックユーザー** 。
- **分析コンピューティングをトランザクションデータベースとは別に管理** - 必要な場合にのみスケーリング。

ハイブリッドマネージャーのレイクハウスクラスターの主要な機能

オブジェクトストレージでの高速クエリの実行

内容 オブジェクトストレージに保存されている Apache Iceberg ®および Delta Lake テーブルで分析クエリを実行します。

理由大規模なデータセットを Postgres ストレージに移動せずに、高速でスケーラブルな分析を有効にします。

方法 HM でレイクハウスクラスターをプロビジョニングし、ベクトル化されたクエリ実行で PGAA を使用してデータをクエリします。

ここで PGFS を介して接続された S3 互換オブジェクトストレージバケット。

ハウツー *Read Iceberg/Delta with or without a catalog*

PGD 階層化テーブルとオフロードのサポート

内容 PGD オフロードおよび階層化データのクエリターゲットとしてレイクハウスクラスターを使用します。

理由履歴データの照会可能な状態を維持しながら、PGD 運用ストレージを最適化します。

方法 Iceberg 形式のオブジェクトストレージへの分析レプリケーション用に PGD ノードグループを構成します。

ここでレイクハウスクラスターを介して照会された、Iceberg のオフロード PGD データ。

ハイブリッドマネージャーの階層化テーブル の概念を学習する

外部データレイクのクエリ

内容 他のツール Spark、Trino、Flink で作成された Iceberg または Delta Lake テーブルを照会します。

理由 Postgres から外部データレイクを照会することにより、データの重複と ETL を回避します。

方法 Lakehouse クラスターを外部の Iceberg カタログに、または Delta Lake ストレージに直接接続します。

ここでオプションのカタログ統合を備えた S3 互換オブジェクトストレージ。

ハウツー *Configure an Iceberg catalog connection*

ハウツー *Read Iceberg/Delta with or without a catalog*

分析コンピューティングの集中管理

内容 PGD クラスターと並行して HM で分析クラスターをプロビジョニング、監視、および管理します。

理由分析コンピューティングとトランザクションコンピューティングを 1 つのコントロールプレーンで管理します。

方法 HM コンソールまたは API を使用して、Lakehouse クラスターを作成および構成します。

ここで Lakehouse クラスターは、HM が管理するクラウド環境で実行されます。

ハウツー *Create a Lakehouse cluster*

ハイブリッドマネージャーでのレイクハウスクラスターの入門

ハイブリッドマネージャーで Lakehouse クラスターの使用を開始するには

1. *Provision a Lakehouse cluster* 。
2. オブジェクトストレージを指す *Configure PGFS storage locations* 。
3. オプショナル *Configure an Iceberg catalog connection* 。
4. PGD と統合する場合、 **ハイブリッドマネージャーの階層化テーブル** を学習します。
5. Iceberg/Delta データの Lakehouse テーブルまたは外部リーダーを作成します。
6. Postgres クライアントを使用してデータを照会します。

関連するハウツー

- *Create a Lakehouse cluster*
- *Configure an Iceberg catalog connection*
- *Read Iceberg/Delta with or without a catalog*

次のトピック

ハイブリッドマネージャーの *Apache Iceberg* ®

8.5.5 Tiered Tables in Hybrid Manager

Hybrid ManagerHM の階層テーブルは、古いデータを PGD クラスターから Apache Iceberg ®形式の費用対効果の高いオブジェクトストレージに移動することにより、大規模な時系列または履歴データセットを管理するための自動ソリューションを提供します。

ハブクイックリンク *Analytics Hub* — **コアコンセプト** — **関連するハウツー**

ハイブリッドマネージャーは、この機能を次の全体にわたって統合します。

- BDR AutoPartition を使用した PGD クラスター

- オフロードのための PGAA 拡張機能
- クエリーのためのレイクハウスクラスター
- 一元化カタログサービス Lakekeeper または外部 Iceberg カタログ

階層テーブルの概念的な概要については、 **ハイブリッドマネージャーの階層化テーブル** を参照してください。

ハイブリッドマネージャーで階層化テーブルを使用する理由

- **ストレージコストの削減**「コールド」データをオブジェクトストレージ Iceberg にオフロードし、プライマリ PGD トランザクションテーブルを縮小します。
- **トランザクションパフォーマンスの高速化**効率的な PGD 操作のために、「ホット」データパーティションを小さく保ちます。
- **自動ライフサイクル管理**経過期間に基づいて、階層間でデータを自動的に移動します。
- **透過的分析** PGD 親テーブルまたはレイクハウスクラスターを介してホットデータとコールドデータの両方を照会します。
- **統合管理**ハイブリッドマネージャーを介してすべてのコンポーネントを構成および監視します。

重要な用語とアーキテクチャの概要

ハイブリッドマネージャー PGFS、PGAA、アナリティクスオフロード、レイクハウスなどで使用される分析用語の定義については、*Analytics concepts (hub)* を参照してください。

ハイブリッドマネージャーで階層化テーブルを使用する必要がある場合

次の場合にハイブリッドマネージャーの階層テーブルを使用します。

- **大規模な時系列データセットを費用効率の高い方法で管理** 。
- **パフォーマンスを向上させるために、PGD 操作テーブルをリーンに保ちます**。
- 古いデータを利用可能な状態にしておくことにより、コンプライアンスニーズに対応しますが、PGD ストレージの外部にあります。
- **BI ユーザーが運用データベースに影響を与えることなく履歴トレンドクエリを実行できるようにします**。
- 最小限の手動介入で **データライフサイクル** を自動化します。

階層化テーブルのユースケース

- **時系列データ**ロギング、IoT センサーの読み取り値、アプリケーションテレメトリ。
- **アーカイブ** コンプライアンスのためのコールドデータの長期保持。
- **ヒストリカルトレンド分析** PGD のパフォーマンスに影響を与えることなく、長年のデータを照会する BI ツール。

- **大規模な追加ほとんどのテーブル**完全な分析アクセスを維持しながら、トランザクションのフットプリントを小さく保ちます。

HM アーキテクチャでの階層テーブルの仕組み

- **PGD クラスター**パーティショニングと古いパーティションの Iceberg への自動オフロードを管理します。
- **PGFS ストレージの場所**オフロードのオブジェクトストレージターゲットを定義します。
- **Iceberg カタログ**オプションでカタログ Lakekeeper または外部でオフロードされたテーブルを管理します。
- **レイクハウスクラスター**オフロードされた Iceberg データを照会するためのスケーラブルな分析コンピューティングを提供します。
- **モニタリング** HM モニタリングツールと可観測性クエリを使用して、オフロードステータスとストレージの節約を追跡します。

EDB Hybrid Manager 内の前提条件

HM に階層テーブルを実装する前に

- **アクティブなハイブリッドマネージャーインスタンス**
- **プロビジョニングされた PGD クラスター** PGAA および PGFS 拡張機能が有効になったバージョン 6.0+
- **レイクハウスクラスター推奨** オフロードデータの照会用
- **カタログサービス** オプショナルただし推奨 —HM 管理の Lakekeeper または外部 REST 互換カタログ
- **カタログのマシンユーザーカタログを使用する場合**適切なカタログデータのライタ/読み取り権限がある場合
- **オブジェクトストレージ** S3 互換、プライベートの場合資格情報あり
- **ユーザー権限**データベースユーザーは BDR および PGAA ファンクションの作成/変更/実行権限を持っている必要があります

メイン機能

- **自動パーティショニング** BDR AutoPartition ストラテジーと `analytics_offload_period` を定義します。
- **ストレージ階層化**オフロードデータには PGFS または Iceberg カタログターゲットを使用します。
- **クエリの透過的** PGD 親テーブルクエリは、ローカル層とアイスバーグ層の両方にヒットする可能性があります。Lakehouse クラスターは、Iceberg テーブルを直接照会できます。
- **モニタステータス**オフロードの進行状況を追跡し、Iceberg コンテンツを検証し、領域の節約を観察します。

ハイブリッドマネージャーでの階層化テーブルのスタートガイド

ハイブリッドマネージャーで階層テーブルの使用を開始するには

1. *Configure PGFS* オブジェクトストレージアクセス。
2. ハイブリッドマネージャーの階層化テーブル を学習し、PGD でポリシー AutoPartition、オフロード期間を適用します。
3. オプションエンジン間の相互運用性が必要な場合、カタログ Lakekeeper または外部を使用します。
4. Lakehouse クラスターを介して、オフロードされた Iceberg データを照会します。

関連リソース

- *PGFS reference*
- ハイブリッドマネージャーの階層化テーブル

可観測性のヒント

- PGD クラスターの状態とオフロードの進行状況には HM ダッシュボードを使用します。
- `bdr.analytics_table` およびパーティションビューで分析クエリを実行します。
- `pg_total_relation_size()` を使用して、PGD ノードで再利用された領域を観察します。
- クラウドストレージコンソールまたは分析を使用して、Iceberg オブジェクトサイズの増加を追跡します。

次のトピック

EDB Postgres Lakehouse Clusters on Hybrid Manager

第9章　Sovereign Data and AI Factory

Sovereign Data and AI Factory は、EDB と Supermicro ®が共同設計した完全に統合されたハードウェアおよびソフトウェアプラットフォームです。これにより、組織は外部に依存せずに、プロダクショングレードの Postgres ®および AI ワークロードを独自のデータセンターに展開できます。

この設計されたシステムは、主権、セキュリティ、およびスケールのために最適化されています。事前に組み立てられ、ライフサイクル管理されており、インストールするとすぐに高度な Postgres、vector、AI パイプラインを実行できます。

9.1 ハードウェアベースのシステムを選択する理由は何ですか?

クラウドファーストは常にオプションであるとは限りません。ソブリンデータと AI ファクトリーは、次のような環境に存在します。

- データとモデルの制御は法的または運用上必要です
- クラウド GPU アクセスが一貫していない、オーバーサブスクライブ、または制限されている
- ソブリン AI は、外部推論またはデータ移動なしを必要とします
- Postgres、vector、AI にわたる統合された可観測性が重要です
- ライフサイクルの信頼性はハードウェアとソフトウェア全体で重要

このシステムは、ハードウェア、オーケストレーションプラットフォーム、サポートされているランタイムを密に結合することにより、最新のワークロードから摩擦を取り除くため、インフラストラクチャではなく結果に集中できます。

9.2 中身は何ですか

ラックマウントシステムとしてデータセンターに提供されるプラットフォームには、次のものが含まれます。

- 検証済みの Supermicro サーバー
- ライフサイクル管理された Kubernetes とハイブリッドマネージャー
- ビルトイン HA Postgres オーケストレーション
- pgvector、埋め込みパイプライン、およびモデルサービング機能
- オプションの GPU ベースの AI 推論とエージェントランタイム
- 完全なモニタリング、パッチ適用、およびバージョンアップグレードの自動化

すべての構成は、アクセスを配信する前に処理されます。インストールしたら、ハイブリッドマネージャー Web インターフェイスを使用して、Postgres クラスター、RAG パイプライン、およびエージェントアプリケーションの展開をすぐに開始します。

9.3 サポートされているワークロード

さまざまなソブリンまたはオフライン対応ワークロードを実行できます。

- 高可用性 Postgres データベース
- ハイブリッド内部 DBaaS 環境
- ベクトル検索と埋め込みパイプライン
- ローカル LLM を使用した取得拡張世代 RAG
- GPU アクセラレーションを使用したオンプレミス AI 推論
- 内部チャットボット、コパイロット、エージェントオーケストレーション

例については、 *Common workloads and use cases* を参照してください。

9.4 ハードウェア構成

2 つの検証済み構成を提供します。

- **コアコンピューティング** —高性能 Postgres およびベクターワークロード用。
- **Advanced AI** —推論およびエージェントベースのシステム用の GPU 対応ノードを追加します。

どちらの構成も拡張可能で、完全にサポートされています。詳細は、*System specifications* を参照してください。

9.5 Explore 機能

- *Concepts and architecture*
- *Glossary and key terms*
- *What to expect*
- *Common workloads and use cases*
- *Frequently Asked Questions - AI Factory on Hybrid Manager*

ライフサイクル操作の詳細については、 Using Hybrid Manager を参照してください。

9.5.1 Concepts and architecture

Sovereign Data and AI Factory は、物理的インフラストラクチャ、オーケストレーションソフトウェア、ライフサイクル自動化を単一のシステムに組み合わせ、データセンターに事前に組み立ておよびインストールします。

システムアーキテクチャとコアロールを理解することは、Postgres ワークロードと AI パイプラインがどのように展開、管理、およびスケーリングされる方法を明確にするのに役立ちます。

前提条件として、 *System specifications* の構成の詳細を参照してください。

ノードタイプとロール

システムには、それぞれが明確な責任と構成を持つ複数のノードロールが含まれています。

管理ノード

管理ノードは、ソブリンデータと AI ファクトリーの追加ソフトウェアパッケージ、ライセンス、およびサポートツールを提供します。インストールごとに 1 つの管理ノードが必要です。

ハイブリッドマネージャーノード

3 つのハイブリッドマネージャーノードは、高可用性コントロールプレーンを形成します。これらは、Kubernetes のスケジューリング、クラスターライフサイクル、オブザーバビリティ、および API サービスを管理します。

ファンクションには次のものが含まれます。

- データベースクラスター展開の管理
- Postgres、OS、および GPU ワークロードからの 200 以上のメトリックの監視
- アップデートとパッチの調整
- 構成 UI と API の提供

これらのノードは、データワークロードの実行には使用されません。

コンピューティングノード

コンピューティングノードは、Postgres データベース、分析パイプライン、またはベクターワークロードを実行するために使用されます。これらのノードは以下をサポートしています

- EDB Postgres Advanced Server または Extended Server
- EDB Postgres 分散オプショナル
- Apache Iceberg ®などのレイクハウスフォーマット
- Python ベースの埋め込みまたは変換パイプライン

AI ノードオプショナル

AI ノードは、大規模なモデルの推論とエージェントの実行をサポートする GPU 対応サーバーです。システムには 2 つのタイプが含まれます。

- 中規模モデルサービング NVIDIA Ada L40S 用の **2-GPU ノード**
- LLM およびエージェントランタイム用の **8-GPU ノード** B200 GPU を搭載した NVIDIA HGX

これらのノードは KServe と統合して、オンプレミスのモデル提供と推論ルーティングを提供します。

オーケストレーションとライフサイクル

システムは、ハイブリッドマネージャーによって管理されるプリインストールされた Kubernetes クラスターを実行します。Kubernetes には直接アクセスしません。代わりに、ハイブリッドマネージャーは以下を行うための Web ベースのインターフェイスを提供します。

- 高可用性を備えた Postgres クラスターを展開する
- ベクターパイプラインまたは AI ワークフローのインストールと監視
- ワークロードを追加ノードにスケーリング
- パフォーマンスとリソースの使用状況を監視する

EDB は、ソフトウェアとハードウェアの両方にライフサイクルサービスを提供します。

- メジャーおよびマイナーバージョンのアップグレード
- セキュリティパッチ
- ハードウェアファームウェアと BIOS のアップデート
- 自動クラスターリバランスと再起動処理

「ソブリン」とは何ですか？

この文脈での主権とは、次のことを意味します。

- すべてのデータとモデルは物理的環境にあります
- 外部クラウドへの依存関係やデータ共有はありません
- あなたがハードウェアを所有し、ネットワークアクセスを制御します
- 規制対象業界のための完全に監査対応の隔離された環境

ソブリンデータと AI ファクトリーは以下に適しています。

- 政府または国家のデータインフラストラクチャ
- ローカルデータ規制の対象となる金融サービス
- 厳しいクラウド制限または機密 IP を持つ企業

サポートされているワークロードパターン

各構成は、さまざまなユースケースをサポートしています。

ワークロード |説明 |コアコンピューティング |高度な AI ——— |———— -|———— –|——— — - 高可用性 Postgres | 3 ノード HA クラスターを実行する |チェック |埋め込みをチェックする|内部データをベクター形式に変換する |いいえ | Check 取得拡張世代 RAG | Postgres + LLM を組み合わせる |いいえ |チェック推論 | GPU を使用してモデルをローカルに提供する |いいえ | Check Agentic ワークフロー |自律的に動作する AI エージェントを構築する |いいえ|小切手

構成の詳細については、 *System specifications* を参照してください。

9.5.2 Frequently asked questions

クラウドサービスの代わりにこのシステムを使用するのはなぜですか?

Sovereign Data and AI Factory は、インフラストラクチャの完全な物理的および運用上の制御を提供します。次の目的に設計されています。

- データをオンプレミスに残しておく必要がある環境
- 一貫した予測可能なパフォーマンスを必要とするワークロード
- AI 機能が必要だが、クラウド API または GPU サービスを使用できない組織
- 単一のプラットフォームからの統合された Postgres と AI オーケストレーションを必要とするチーム

ハードウェアのインストールと保守は誰が行いますか?

Supermicro と EDB は、共同でお客様のサイトにシステムをインストールします。すべてのハードウェア、ファームウェア、および OS レベルのメンテナンスは、Postgres、ハイブリッドマネージャー、および AI ツールのサポートとともに、契約の下でカバーされています。

どのようなソフトウェアを実行しますか?

システムは以下を実行します。

- EDB Postgres Advanced Server または Extended Server
- オーケストレーションとオブザーバビリティのためのハイブリッドマネージャー
- AI Factory コンポーネント
- ベクターおよび分析ワークロード用の pgvector および Apache Arrow/Iceberg
- モデルサービング用のコンテナと KServe
- オプション マルチノード HA 用の EDB Postgres 分散

このシステムをクラウドサービスに接続できますか?

できますが、必須または想定されていません。プラットフォームは、完全に自己完結型になるように設計されています。内部ポリシーに応じて、アウトバウンドトラフィックを選択的に有効にすることができます。

AI の使用がサポートされているモデルは何ですか?

システムは、次のような標準のコンテナ化モデルをサポートします。

- `bge-base` 、`instructor-xl` 、`e5` のようなモデルの埋め込み
- `LLaMA2` 、`Mistral` 、`Mixtral` のような言語モデル
- タスク固有のモデル要約、分類など

カスタムモデルは、推論コンテナ形式に準拠している限りにおいて展開できます。

外部データベースをモニタできますか?

はい。ハイブリッドマネージャーは、オブザーバーモードで外部 Postgres クラスターのメトリックを監視および表示できます。ただし、承認されたソブリンデータおよび AI ファクトリーノードのみが内部クラスターに追加し、ライフサイクルワークフローで管理できます。

システムをスケーリングするにはどうすればよいですか?

追加のコンピューティングノードまたは GPU 対応 AI ノードを注文してラックにできます。

サポートには何が含まれますか?

- ソフトウェアスタック EDB の 24 時間 365 日のグローバルサポート
- ハードウェアとファームウェアのサポート
- ハイブリッドマネージャーを介して管理されるセキュリティパッチとバージョンのアップグレード
- 拡張、更新、またはカスタム構成のライフサイクル支援

9.5.3 Glossary and key terms

これらの重要な用語とアーキテクチャコンポーネントは、ソブリンデータと AI ファクトリーのドキュメント全体で使用されています。

管理ノード

システムの中央制御ノード。ハイブリッドマネージャーインターフェイス、認証サービス、およびオーケストレーションロジックをホストします。

AI ノード

モデル推論、ベクトル処理、およびエージェント実行に使用される GPU 対応サーバー。オプションの AI ワークロード構成の一部。

エージェントワークフロー

AI エージェントが内部ツール、メモリ、フィードバックループを使用して独立してタスクを実行するマルチステップの自動化パターン。多くの場合、LLM およびベクターデータに基づいて構築されます。

コントロールプレーン

クラスターのスケジューリング、モニタリング、ライフサイクル操作、およびアクセス制御を担当する 3 つのハイブリッドマネージャーノードのセット。データワークロードとは無関係に動作します。

コンピューティングノード

Postgres クラスター、分析、またはベクターパイプラインを実行するために使用される汎用の高性能ノード。必要に応じてスケールアウトできます。

埋め込み

構造化または非構造化データを、検索や分類などの AI ワークフローに適したベクトル表現に変換するプロセス。

EDB ハイブリッドマネージャー

Kubernetes 環境でデータベースと AI ワークロードをオーケストレーションするための EDB の管理レイヤークラスターの展開、システムの監視、およびライフサイクル操作の管理に使用されます。

推論

機械学習モデルから出力を生成する行為。このシステムでは、推論は、KServe および NVIDIA NIM コンテナを使用して専用の GPU ノードで実行されます。

KServe

推論ワークロードを実行するために使用されるオープンソースのモデル提供フレームワーク。LLM を提供し、モデルを埋め込むために、ハイブリッドマネージャーと GPU ノードに統合されています。

ライフサイクルサービス

セキュリティパッチ、OS 更新、Postgres のアップグレード、ファームウェア管理を含む、ハードウェアとソフトウェアの両方の継続的なサポートと管理。

EDB Postgres Distributed PGD

EDB のマルチノード、高可用性 Postgres ソリューション。グローバルに分散または同期のマルチリージョンクラスターを必要とする顧客のオプション。

取得拡張生成 RAG

構造化データ多くの場合 Postgres が取得されて LLM に渡されて、自然言語応答を生成するワークロードパターン。埋め込み機能と推論機能が必要です。

ソブリン AI

すべてのデータ、モデル、実行環境が、必要に応じて完全にエアギャップされた顧客制御のスペース内に物理的および論理的に残る AI 展開モデル。

Vector ストア

埋め込みベクトルを保存し、類似性検索をサポートするデータベースまたはインデックス。Postgres と pgvector は、このシステムでローカルベクターストレージに使用されます。

9.5.4 System specifications

Sovereign Data and AI Factory システムは、コアコンピューティングとアドバンスト AI の 2 つの構成層で利用できます。各システムは事前に組み立てられ、完全なソフトウェアとハードウェアのサポートでライフサイクル管理されています。

構成オプション

Configuration	Description	Total nodes
Core Compute	高性能 Postgres およびハイブリッド DBaaS ワークロード用に設計	7
Advanced AI	AI および推論ワークロード用の GPU ノードでコアコンピューティングを拡張	11
	NA	
	NA	

構成 1 コアコンピューティング

含まれるもの

- 1x 管理ノード
- 3x ハイブリッドマネージャー/コントロールプレーンノード
- 3x Hyper 2U コンピューティングノード

管理ノード

Category	Product / Model	Description	Qty
System	SYS-111C-NR	1U X13SEDW-F サーバー	1
Processor	P4X-EMR4514Y-SRN6J-MC	Intel Xeon Gold 4514Y 16 コア@ 2.0GHz	1
Memory	MEM-DR516L-SL01-ER56	16GB DDR5-5600 1RX8 LP ECC RDIMM 合計 128GB	8

次のページに続く

表 2 – 前のページからの続き

Category	Product / Model	Description	Qty
Storage	HDS-MUN-MTFDKCC3T2TFS1	Micron 7450 MAX 3.2TB NVMe PCIe 4.0 U.3	2
Storage Controller	AOC-VROCSTNMOD	Intel VROC RSTe 標準 HW キー	1
Networking	AOC-AG-I2M-O	AIOM デュアルポート 1GbE RJ45	1
Networking	AOC-S25G-I2S-O	標準ロープロファイル 25G デュアルポート SFP28	1

コアコンピューティング – Hyper 2U サーバーサーバーごとに x6

Category	Product / Model	Description	Qty
System	SYS-221H-TNR	ハイパー 2U サーバーシャーシ、8x2.5 インチベイ、X13DEM マザーボード	1
Processor	R6538N-SRN6V-MCC	Intel Xeon Platinum 6538N 32 コア@ 2.1GHz	2
Cooling	SNK-P0087V	X13 プラットフォーム用のパッシブ CPU ヒートシンク	2
Memory	MEM-DR532L-SL01-ER56	32GB DDR5-5600 2Rx8 LP ECC RDIMM 合計 1024GB	32
Storage	HDS-SMN0-MZ1L2960HCJR	Samsung PM9A3 960GB NVMe PCIe 4.0 M.2	2
Storage	HDS-TUN-KCD8XPUG15T3	Kioxia CD8-P 15.3TB NVMe PCIe 5.0 2.5 インチ	4
Storage	HDS-TUN-KCD8XPUG1T92	Kioxia CD8-P 1.92TB NVMe PCIe 5.0 2.5 インチ	2
Storage Controller	AOC-VROCPREMOD	Intel VROC プレミアムコントローラ RAID 0、1、5、10	1

次のページに続く

表 3 – 前のページからの続き

Category	Product / Model	Description	Qty
Networking	AOC-A100G-M2CM-O	AIOM 2 ポート 100GbE QSFP28、Mellanox CX-6 DX	2
Power	PWS-2K63A-1R	1U 2600W 冗長電源	2
Cooling	FAN-0239L4	80x80x38 mm、16.8K RPM、中間冷却ファン	4
Expansion	RSC-H-68G5 / RSC-H-6G5L	1U LHS ハイパーライザーカード PCIe 5.0 x16/x8、PCIe 5.0 x16	2
Security	AOM-TPM-9672V-O	TPM 2.0 モジュール	1
		NA	
		NA	

構成 2: 高度な AI

すべてのコアコンピューティングコンポーネントに加えて、以下が含まれます。

- 2x 2-GPU Hyper 2U サーバー
- 2x 8-GPU B200 10U サーバー

2-GPU Hyper 2U サーバーサーバーごとに x2

Category	Product / Model	Description	Qty
System	SYS-221H-TNR	ハイパー 2U サーバーシャーシ、8x2.5 インチベイ、X13DEM マザーボード	1
Processor	R6538N-SRN6V-MCC	Intel Xeon Platinum 6538N 32 コア@ 2.1GHz	2
Cooling	SNK-P0087V	X13 プラットフォーム用 1U パッシブ CPU ヒートシンク	2

次のページに続く

表 4 – 前のページからの続き

Category	Product / Model	Description	Qty
Memory	MEM-DR532L-SL01-ER56	32GB DDR5-5600 2Rx8 LP ECC RDIMM 合計 1024GB	32
GPU	GPU-NVL40S	NVIDIA Ada L40S 48GB GDDR6 PCIe Gen 4	2
Storage	HDS-SMN0-MZ1L2960HCJR	Samsung PM9A3 960GB NVMe PCIe 4.0 M.2	2
Storage	HDS-TUN-KCD8XPUG15T3	Kioxia CD8-P 15.3TB NVMe PCIe 5.0 2.5 インチ	4
Storage	HDS-TUN-KCD8XPUG1T92	Kioxia CD8-P 1.92TB NVMe PCIe 5.0 2.5 インチ	2
Storage Controller	AOC-VROCPREMOD	Intel VROC プレミアムコントローラ RAID 0、1、5、10	1
Networking	AOC-A100G-M2CM-O	AIOM 2 ポート 100GbE QSFP28、Mellanox CX-6 DX	2
Power	PWS-2K63A-1R	1U 2600W 冗長電源	2
Cooling	FAN-0239L4	80x80x38 mm、16.8K RPM、中間冷却ファン	4
Cooling	MCP-310-8294...	GPU プラスチックエアシュラウド	2
Expansion	RSC-H-68G5 / RSC-H-6G5L	1U LHS ハイパーライザーカード PCIe 5.0 x16/x8、PCIe 5.0 x16	2
Security	AOM-TPM-9672V-O	TPM 2.0 モジュール	1

8-GPU B200 10U サーバーサーバーごとに x2

Category	Product / Model	Description	Qty
System	SYS-A21GE-NBRT-G1	Supermicro 10U B200 Gold シリーズ GPU サーバー	1
Processor	Intel ® Xeon ® Platinum 8570	56 コア@ 2.10GHz 350W	1
Memory	DDR5 ECC RDIMM	96GB DDR5 5600MHz ECC RDIMM 合計 3072GB	32
GPU	NVIDIA HGX B200	8x B200 180GB GPU を搭載したベースボード	1
DPU	NVIDIA BlueField-3	BF3220 DPU 200GbE デュアルポート、暗号化対応	1
Storage	PM9A3 M.2	1.9TB M.2 NVMe	1
Storage	7450 PRO 2.5"	3.8TB NVMe PCIe 4.0 SSD 1 DWPD	1
*Networking	AOC-1	• 400 ギガビット MCX75310AA NEAT OSFP アダプター	1
Onboard	—	2x RJ45 10Gbase-T ポート	1
Security	AOM-TPM-9670V-P	TPM 2.0 モジュール	1

9.5.5 Common workloads and use cases

Sovereign Data and AI Factory は、データ、インフラストラクチャ、AI の実行の厳密な制御を必要とする組織のために専用に構築されています。ローカルで管理されながら、幅広いワークロードをサポートします。

クラウドでエンジニアドシステムを使用する理由は何ですか?

多くの最新のワークロードはクラウド環境で実行できますが、すべての組織がクラウド環境に依存できるわけではありません、または依存する必要があります。Sovereign Data and AI Factory は、次のニーズを満たすために存在します。

データ主権とコンプライアンス

- すべてのデータ、モデル、ワークロードは物理的インフラストラクチャ内に残ります。
- 外部 API やサードパーティのストレージは必要ありません。
- 金融、政府、医療、国家安全保障分野での展開を可能にします。

規制された環境

- インターネットへの依存を制限して動作します。
- 導入ライフサイクルをサポートします。
- 以前は閉ざされていた環境で LLM および AI のユースケースを有効にします。

Postgres + AI の統合プラットフォーム

- HA Postgres、ベクトル検索、および GPU 推論を 1 つのシステムに組み合わせ、組織によって制御されます。
- オーケストレーション、モニタリング、およびサポートを簡素化します。
- 断片化したソリューションと重複したインフラストラクチャを排除します。

予測可能なパフォーマンスとコスト

- 下り料金、調整、または予測できない GPU キューはありません。
- 既知のハードウェア全体で最適化された I/O とコンピューティング。
- 総所有コストが明確で制御されています。

ライフサイクル管理されたインフラストラクチャ

- 共同 EDB + Supermicro サポート契約。
- ハードウェアとソフトウェアのパッチが統合および調整されました。
- 顧客主導のインフラストラクチャ管理または DevOps に依存しません。

サポートされているワークロードパターン

次のワークロードは、ソブリンデータおよび AI Factory システムでエンドツーエンドでサポートされています。

高可用性 Postgres

- 自動フェイルオーバーを使用してミッションクリティカルなシステムオブレコードを実行します。
- 可観測性とライフサイクルが焼き付けられます。

ハイブリッド DBaaS

- 1 つのインターフェイスからマルチプルのクラスターを使用して内部チームにサービスを提供します。
- ロールベースのアクセス制御とクラスターレベルの可視性。

エンベディングとベクターパイプライン

- ローカルモデルを使用してデータをエンベディングに変換します。
- Postgres の pgvector を使用して保存および照会します。
- API キーまたはクラウドエンドポイントは必要ありません。

取得拡張生成 RAG

- 構造化内部データを照会し、LLM に渡します。
- 独自の境界内で取得と推論の両方をホストします。

オンプレミスの推論

- 専用の GPU ハードウェアで生成モデルおよび埋め込みモデルを提供します。
- 標準のコンテナ NVIDIA NIM などを使用して、モデル提供を抽象化します。

エージェント AI ワークフロー

- 内部ツールとデータを使用して、マルチステップのタスクを実行する AI エージェントをビルドします。
- 外部 API やデータ漏洩はありません。
- Slack、カスタムダッシュボード、または内部ツールにインターフェイスを展開します。

9.5.6 What to expect

ここでは、最初の 90 日間、配信からアクティブな使用とサポートまで、ソブリンデータと AI Factory システムの受信、セットアップ、および操作時に何が発生するかを説明します。

0〜21 日目: サイトの準備とインストール

注文が確認されたら、サイトの準備状況アンケートに回答して、インストールをガイドします。調査には以下が含まれます。

- 電源、ラック、およびケーブル配線の詳細
- ネットワーキングとセキュリティ設定
- 物理的アクセスと連絡先の調整

何が起こるか

- Supermicro は、選択したデータセンターに完全なシステムを出荷します。
- EDB および Supermicro チームは、オンサイトのラッキング、電源投入、および検証を実行します。
- 設定に基づいて設定が完了します。
- Hybrid Manager ポータルへのログイン資格情報と URL アクセスを受け取ります。

21〜28 日目: 最初のワークロードの展開

システムがライブになると、すぐにワークロードの起動を開始できます。すべての操作は、Hybrid Manager ポータルを介して実行されます。

あなたにできること

- 高可用性 3 ノード Postgres クラスターを展開します。
- Postgres と pgvector を使用して、埋め込みジョブとベクターパイプラインを起動します。
- GPU ノードを使用して、 NVIDIA NIM マイクロサービスを介してローカルにモデルを提供します。
- RAG スタイルのアプリケーションと内部チャットボットを作成します。

Kubernetes に直接アクセスする必要はありません。HM ポータルを介してすべてを管理します。

28〜90 日目 ライフサイクル管理とスケーリング

ワークロードの進化に応じて、システムは EDB および Supermicro からの完全なライフサイクルサポートでスケーリングと更新の準備が整います。

含まれるもの

- Postgres や AI ツールを含む定期的な OS とソフトウェアの更新。
- コンピューティングまたは GPU ノードを追加することによるオプションの拡張。
- Supermicro の 4 時間のゴールド SLA によるハードウェアサポート。
- すべてのレイヤーにわたるビルトインの可観測性と診断。

ソフトウェアとハードウェアの両方が共同サポート契約の対象となっている間、ハイブリッドマネージャーを介してシステムの操作を継続できます。