



EDB Postgres AI for ClickHouse

リリース *26.3.13.31*

EnterpriseDB Corporation

2026 年 06 月 24 日

目次

第 1 章	What ClickHouse is for	3
1.1	ClickHouse と WarehousePG	3
第 2 章	Release notes	5
2.1	EDB Postgres AI for ClickHouse 26.3.13.31 release notes	5
第 3 章	Overview	7
3.1	EDB が ClickHouse を提供する方法	7
3.2	含まれるもの	7
3.3	カバーされていないもの	8
3.3.1	ClickHouse architecture	8
3.3.2	EDB Postgres AI for ClickHouse compatibility	11
3.3.3	Known issues	12
第 4 章	Installing and configuring EDB Postgres AI for ClickHouse	13
4.1	展開オプション	13
4.1.1	シングルノード	13
4.1.2	マルチノードクラスター	13
4.2	インストール方法の選択	13
4.2.1	Installing EDB Postgres AI for ClickHouse on Rocky Linux	13
4.2.2	Installing EDB Postgres AI for ClickHouse with Docker	20
4.2.3	Installing EDB Postgres AI for ClickHouse on Kubernetes	30
第 5 章	Getting started with ClickHouse	35
5.1	ClickHouse に接続する	35
5.2	統合	35
5.3	関連項目	35
第 6 章	Connecting ClickHouse to WarehousePG	37
6.1	ClickHouse からの接続を受け入れるように WarehousePG を構成する	37
6.2	ClickHouse でリンクテーブルを作成する	38
6.3	ClickHouse から WarehousePG データを読み取る	39
6.4	ClickHouse から WarehousePG へのデータの書き込み	39
6.5	クエリフローの理解	39
第 7 章	Integrating with Iceberg catalogs	41
7.1	カタログへの接続	41
7.2	データの照会	42

ClickHouse は、リアルタイム分析のためのカラム型データベースエンジンです。1 秒ごとに数百万のイベントを取り込んで、すぐに照会可能にして、数 10 億行にわたって 1 秒未満の集計を提供します。これは、大量のライブデータに対する 1 秒未満の回答を必要とするワークロードに最適なエンジンです。

ClickHouse の EDB Postgres AI は、オープンソースの ClickHouse の EDB 署名パッケージ、SLA、CVE 応答、およびブレイク/修正の専門知識を提供します。フォークではありません。管理対象サービスではありません。EDB のエンタープライズサポートを利用して、独自の ClickHouse 展開を実行します。

ClickHouse の EDB Postgres AI は、オープンソースの ClickHouse に完全にに基づいています。カバーされる内容と別の商用オフリングが必要な内容の完全な内訳については、[Overview](#) ページを参照してください。

第1章 What ClickHouse is for

ClickHouse は、速度が他のすべてを上回るワークロード用に最適化されています。

- データが数秒または数分前に到着し、すぐに照会可能である必要がある大量の分析
- IoT モニタリング、アドテク入札、不正検出、および同様のタイムクリティカルなワークロード
- ライブデータでの数千の同時ユーザーを含むユーザー向けダッシュボード
- イベント、ログ、テレメトリの 1 秒未満のロールアップ
- Postgres の可観測性、大規模な Postgres データベースからのメトリックの保存と照会

1.1 ClickHouse と WarehousePG

ClickHouse と WarehousePG は、EDB Postgres AI Analytics 柱内でさまざまな質問に回答し、相互補完します。

ClickHouse は、「今何が起きているのですか？」に答えます。大量の最近のデータの 1 秒未満の集計、巨大な単一のテーブルにわたる数列のスキャン用に最適化されています。

WarehousePG は、「何が起こったのか？」という質問に答えます。複雑なマルチテーブルの結合、エンリッチメント、データモデリング、および速度よりも精度が重要なバッチ分析。

Apache Iceberg[®]は、それらの間の真実の共有ソースとして機能します。いずれかのエンジンによって書き込まれたデータは、両方からアクセスでき、それらの間にコピーや ETL パイプラインはありません。

第2章 Release notes

EDB Postgres AI for ClickHouse ドキュメントでは、マイナーリリースとパッチを含む ClickHouse の EDB Postgres AI の最新バージョンについて説明しています。これらのリリースノートでは、各リリースの新機能をカバーしています。

Version	Release date
<i>DataLakeCatalog reference</i>	24 June 2026

2.1 EDB Postgres AI for ClickHouse 26.3.13.31 release notes

リリース日 2026 年 6 月 24 日

ClickHouse 26.3.13.31 の EDB Postgres AI は、最初のリリースです。新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

Type	Description
Upstream merge	オープンソースの ClickHouse 26.3.13.31 LTS に基づいています。上流の変更の詳細については、 ClickHouse 26.3 changelog を参照してください。

第3章 Overview

EDB が ClickHouse をパッケージ化して配布する方法、このドキュメントでカバーされている内容、およびその範囲外を理解します。

3.1 EDB が ClickHouse を提供する方法

ClickHouse の EDB Postgres AI はフォークではありません。EDB は、アップストリームリリースと同じ、エンタープライズ Linux ディストリビューション用の RPM、コンテナイメージ、および Kubernetes オペレーターとして再パッケージされ、EDB によってサポートされているオープンソースの ClickHouse バイナリをビルドおよび配布します。

EDB の貢献は次のとおりです。

- 固定された上流リリースでの公式オープンソースの ClickHouse ソースをビルドする
- バイナリを RPM、コンテナイメージ、および Kubernetes オペレーターとしてパッケージし、EDB パッケージリポジトリを介して配布されます。
- WarehousePG および EDB Postgres Lakehouse と並んで、ClickHouse を EDB Postgres AI Analytics プラットフォームに統合する
- EDB サポートの提供

ClickHouse クエリエンジン、SQL 言語、テーブルエンジン、およびパフォーマンス特性は [upstream open-source release](#) と同じです。

EDB パッケージビルドを実行していることを確認します。

```
SELECT value FROM system.build_options WHERE name = VERSION_OFFICIAL;
```

出力には、EDB パッケージリリースの (EDB Build) が含まれます。

3.2 含まれるもの

EDB は、オープンソースの ClickHouse バイナリをパッケージ化してサポートしています。EDB ディストリビューションでは次の機能が利用できます。

Area	What's covered
テーブルエンジン	MergeTree family, integration engines (<i>PostgreSQL</i> , <i>S3</i> , <i>Kafka</i> , <i>IcebergS3</i> , <i>Delta-Lake</i> , <i>MongoDB</i> , <i>Redis</i> , and more), and special engines (<i>Distributed</i> , <i>Materialized-View</i> , <i>Dictionary</i> , and more)
Database engines	<i>Atomic</i> , <i>Replicated</i> , <i>PostgreSQL</i> , <i>MySQL</i> , <i>SQLite</i> , <i>DataLakeCatalog</i>
SQL and query capabilities	Aggregations, window functions, CTEs, subqueries, materialized views, all major data formats (Parquet, JSON, CSV, Avro, ORC, Arrow), dictionaries, projections, and RBAC
Data lake integration	Read access to Apache Iceberg [®] tables via <i>DataLakeCatalog</i> (REST, AWS Glue, Databricks Unity Catalog, Hive, OneLake) and directly via <i>IcebergS3</i> , <i>IcebergAzureBlobStorage</i> , and related engines; Delta Lake and Apache Hudi via dedicated table engines
<i>DataLakeCatalog</i> reference	ClickHouse Keeper, <i>clickhouse-client</i> , <i>clickhouse-local</i> , and Named Collections

3.3 カバーされていないもの

次の機能は、上流の ClickHouse プロジェクトとは別の商用サービスである ClickHouse[®] Cloud 専用です。これらはオープンソースバイナリの一部ではないため、このドキュメントでは説明していません。

Feature	Description
ClickPipes	Kafka、S3、Postgres CDC、MySQL、MongoDB などの管理対象インジェストパイプライン
SharedMergeTree	ClickHouse [®] Cloud の共有ストレージアーキテクチャ用の独自のテーブルエンジン
SQL Console	ダッシュボード、ビジュアライゼーション、および GenAI SQL 支援を備えた Web ベースのクエリエディター
Query Insights	Cloud コンソールに組み込まれたマネージドクエリログ分析
Managed Postgres	ClickHouse と連携するフルマネージドの Postgres サービス
Compute-compute separation	ストレージとは無関係にコンピューティングをスケーリングするためのアーキテクチャ
Auto-scaling	自動垂直および水平スケーリング
Managed backups	外部エクスポートを使用した構成可能な自動バックアップ
BYOC	Bring Your Own Cloud、AWS アカウントに展開されるマネージドサービス
Compliance certifications	HIPAA、SOC2、PCI DSS、ISO 27001 クラウドサービスで管理
SSO / CMEK	Cloud コンソールを介したシングルサインオンと顧客管理の暗号キー

3.3.1 ClickHouse architecture

ClickHouse の EDB Postgres AI は、スタンドアロンインスタンスまたは分散クラスターとして実行され、シャーディング、レプリケーション、またはその両方をサポートしています。

ClickHouse サーバーは、データを保存および処理するデータベースエンジンです。ClickHouse クライアントは、サーバーに接続し、クエリを実行するためのコマンドラインインターフェイスです。

デプロイメントトポロジ

トポロジは、データがどのように保存、複製、および提供される方法と、どの程度の操作の複雑性を引き受けるかを決定します。

- **シングルノード ClickHouse** は、単一のホストでスタンドアロンインスタンスとして実行されます。クラスター調整は必要ありません。開発、テスト、およびシングルテナントの分析ワークロードに適しています。
- **マルチノードクラスター** データは水平スケールのためにシャード全体に分散され、高可用性のために複製されます。マルチノードクラスターは、シャーディングのみ、レプリケーションのみ、または両方を一緒に使用できます。レプリケーションには ClickHouse Keeper が必要です。

テーブルエンジン

ClickHouse では、すべてのテーブルに、データを保存、インデックス付け、およびレプリケートする方法を決定するエンジンがあります。プライマリファミリーは **MergeTree** で、高スループット分析ワークロード用に設計されています。**ReplicatedMergeTree** は、ノード間のビルトインレプリケーションでそれを拡張します。分散クエリの場合、**Distributed** エンジンは、クエリを基になるシャードにルーティングし、結果をマージする仮想テーブルとして機能します。完全なリファレンスについては、ClickHouse ドキュメントの [Table engines](#) を参照してください。

シャーディング

ClickHouse は、シェアードナッシングアーキテクチャを使用します。各ノードは独自のストレージを持ち、独立して動作します。シャーディングは、各ノードが異なる行のサブセットを保持する複数のノードにデータを水平に分割することにより、これを利用します。クエリーはシャード間で並列実行され、その結果はクライアントに返される前にマージされます。このアプローチでは、書き込みスループットとストレージを単一のノードで処理できる量を超えてスケールしますが、高可用性は提供されません。シャードがダウンすると、そのデータは利用できなくなります。

シャーディングには、クラスターコーディネーションは必要ありません。クエリルーティングは、**Distributed** テーブルエンジンによって処理され、すべてのシャードにクエリーを並列送信し、結果を透過的にマージします。詳細は、*ClickHouse がクエリをルーティングする方法* を参照してください。

レプリケーション

レプリケーションは、マルチプルのノードで同じデータの同一のコピーを保持します。1つのノードがダウンすると、別のレプリカがクエリーを提供できます。レプリケーションは高可用性を提供しますが、自分自身でストレージ容量や書き込みスループットを向上させることはありません。

ClickHouse でレプリケーションを有効にするには、テーブルエンジンとして **ReplicatedMergeTree** を選択し、ClickHouse Keeper を展開します。Keeper は、どのデータ部分が存在するかを追跡し、コピーの同期を保つことにより、レプリカを調整します。

ClickHouse Keeper

ClickHouse Keeper は、ClickHouse のビルトイン調整サービスであり、Apache ZooKeeper と同等の役割を持ち、プロトコルレベルでそれと互換性があります。以下を処理します。

- レプリカ全体に存在するデータ部分の追跡
- レプリカの同期を維持するためのインサートの調整
- 分散 DDL の実行 すべてのノードに適用されるスキーマの変更
- レプリケート操作のリーダーの選出

Keeper は Raft コンセンサスアルゴリズムを使用し、機能するにはノードのクォーラムが必要です。最小は 3 つのキーパーノードであり、これにより、クラスターは 1 つのノードの障害を許容できます。5 つの Keeper ノードを使用すると、2 つの障害を許容できます。

Keeper は 2 つの方法で実行できます。コロケーション展開では、Keeper は同じノードで ClickHouse サーバーと並行して実行されます。コロケーションは管理が簡単で、小規模なクラスターに適しています。専用展開では、Keeper はサーバーワークロードのない別のノードで実行されます。これにより、分離が向上し、大規模なクラスターまたは実稼働クラスターに推奨されます。

シャーディングとレプリケーションの組み合わせ

シャーディングとレプリケーションを組み合わせ、ホリゾンタルスケールと高可用性の両方を実現します。一般的な運用トポロジは、それぞれ 2 つのレプリカを持つ 2 つのシャードであり、合計 4 つのサーバーノードになります。

- 2 つのシャードは、2 つの独立したデータセットにわたる水平スケールを提供します。
- シャードごとに 2 つのレプリカが高可用性を提供します。各シャードにはスタンバイコピーがあります。

Keeper は、少なくとも 3 つのノードで実行され、サーバーノード上のコロケーションまたは別の専用ノードで実行されます。2 つの Keeper 展開アプローチに関するガイダンスについては、[展開オプション](#) を参照してください。

</div>

ClickHouse がクエリをルーティングする方法

データが複数のノードにまたがってシャード化されている場合、クエリーはすべてのシャードに到達し、その結果を組み合わせる必要があります。ClickHouse は、Distributed テーブルエンジンでこのルーティングを処理します。

Distributed テーブルは、各シャードの基になる MergeTree または ReplicatedMergeTree テーブルの上部にある仮想テーブルです。任意のノードで Distributed テーブルを照会すると、そのノードはクエリをすべてのシャードに平行にファンアウトし、結果を収集し、マージされた結果を返します。

アプリケーションは、シャーディングトポロジを知る必要がなく、サーバーノードに接続し、Distributed テーブルを照会します。書き込みの場合、Distributed テーブルに挿入すると、定義したシャーディングキー通常は列のハッシュを使用して、各行が正しいシャードにルーティングされます。

シャーディング、レプリケーション、および Keeper トポロジはすべて ClickHouse サーバー構成ファイルで定義されます。詳細については、ClickHouse ドキュメントの [Configuration files](#) を参照してください。

3.3.2 EDB Postgres AI for ClickHouse compatibility

プラットフォームの互換性

x86-64 上の Rocky Linux 10。

システム要件

ノードごとの最小推奨ハードウェア

	Single-node	Cluster node
CPU	4 cores	4 cores
RAM	16 GB	16 GB
Storage	100 GB (NVMe preferred)	100 GB (NVMe preferred)

製品サイズのガイダンスについては、[ClickHouse sizing and hardware recommendations](#) を参照してください。

WarehousePG

- WarehousePG 6.x 以降。

Apache Iceberg[®]

Iceberg format version	Read	Write
v1	Yes	No
v2 (position deletes)	Yes	No
v2 (equality deletes)	Yes	No
v3 (deletion vectors)	No	No

Iceberg の書き込みサポートはまだ利用できません。[Known issues](#) を参照してください。

オブジェクトストレージ

Provider	Table engines
Amazon S3	<i>IcebergS3, DeltaLake, S3</i>
Azure Blob Storage	<i>IcebergAzureBlobStorage, AzureBlobStorage</i>
HDFS	<i>IcebergHDFS, HDFS</i>

次のページに続く

表 5 – 前のページからの続き

Provider	Table engines
Local filesystem	<i>IcebergLocal, File</i>

3.3.3 Known issues

このリリースには、次の既知の問題と制限が含まれています。該当する場合、影響を軽減するために役立つ回避策が含まれています。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。

- PostgreSQL テーブルエンジンは、WarehousePG コーディネーターにのみ接続し、WarehousePG の超並列処理 MPP アーキテクチャには関与しません。大量のデータを移動する場合、`INSERT INTO clickhouse_table SELECT * FROM warehousepg_linked_table` を使用してネイティブの ClickHouse MergeTree テーブルにデータをロードします。
- Iceberg カタログ統合は読み取り専用です。INSERT、UPDATE、および DELETE 操作はサポートされていません。Spark、PyIceberg、または別の Iceberg 互換ライタを使用して、Iceberg テーブルを作成および設定します。

第4章 Installing and configuring EDB Postgres AI for ClickHouse

展開を計画し、インフラストラクチャに ClickHouse の EDB Postgres AI をインストールします。

4.1 展開オプション

トポロジは、データがどのように保存、複製、および提供される方法と、どの程度の操作の複雑性を引き受けるかを決定します。シャーディング、レプリケーション、および ClickHouse Keeper の完全な説明については、*ClickHouse architecture* を参照してください。

4.1.1 シングルノード

ClickHouse は、単一のホストでスタンドアロンインスタンスとして実行されます。クラスター調整は必要ありません。開発、テスト、およびシングルテナントの分析ワークロードに適しています。

4.1.2 マルチノードクラスター

データは水平スケーリングのためにシャード全体に分散され、高可用性のために複製されます。レプリケーションには、少なくとも 3 つのノードで実行されている ClickHouse Keeper が必要です。Keeper は、サーバーノードまたは別の専用ノードでコローションで実行できます。

4.2 インストール方法の選択

EDB は、ClickHouse をインストールする 3 つの方法を提供します。

- *Installing EDB Postgres AI for ClickHouse on Rocky Linux*
- *Installing EDB Postgres AI for ClickHouse with Docker*
- *Installing EDB Postgres AI for ClickHouse on Kubernetes*

4.2.1 Installing EDB Postgres AI for ClickHouse on Rocky Linux

EDB は、x86-64 上の Rocky Linux 10 の RPM パッケージとして ClickHouse を配布しています。次のパッケージが利用できます。

Package	Description	Install on
<i>edb-clickhouse-server</i>	ClickHouse サーバーエンジン、systemd サービス、およびデフォルトの構成ファイル。`clickhouse-keeper.service`が含まれるため、サーバーノードは追加パッケージを使用せずにサーバーと並行して Keeper を実行できます。	Server nodes
<i>edb-clickhouse-client</i>	<i>clickhouse-client</i> 、 <i>clickhouse-local</i> 、 <i>clickhouse-benchmark</i> 、および関連ツール	Server nodes, remote client hosts
<i>edb-clickhouse-common-static</i>	コンパイルされた ClickHouse バイナリ。サーバーおよびクライアントパッケージに必要です。	All server and client nodes
<i>edb-clickhouse-keeper</i>	専用調整ノードのスタンドアロン Keeper バイナリ。別のノードで Keeper を実行するマルチノード展開の場合にのみ必要です。	Dedicated Keeper nodes only

ノードのロールと展開トポロジの説明については、[ClickHouse architecture](#) を参照してください。

前提条件

- Rocky Linux 10
- sudo または root アクセス
- EDB サブスクリプショントークン、[EDB customer portal](#) から入手できます。

単一ノードの展開

ClickHouse は、クラスター調整を必要としないスタンドアロンインスタンスとして実行されます。開発、テスト、およびシングルテナントの分析ワークロードに適しています。

1. パッケージをインストールします。

```
export EDB_SUBSCRIPTION_TOKEN=<your-token>
export EDB_SUBSCRIPTION_PLAN=clickhouse
curl -sSLf "https://downloads.enterprisedb.com/$EDB_SUBSCRIPTION_TOKEN/$EDB_
→SUBSCRIPTION_PLAN/setup.rpm.sh" | sudo -E bash
sudo dnf install -y \
  edb-clickhouse-common-static \
  edb-clickhouse-server \
  edb-clickhouse-client
```

1. サーバーを起動します。

```
sudo systemctl enable clickhouse-server
sudo systemctl start clickhouse-server
```

サーバーが実行されていることを確認します。

```
sudo systemctl status clickhouse-server
```

1. ClickHouse コマンドラインクライアントを使用して接続します。

```
clickhouse-client
```

デフォルトでは、`default` ユーザーにはパスワードがありません。設定するには、接続した後に次を実行します。

```
ALTER USER default IDENTIFIED BY yourpassword;
```

次に、次のように接続します。

```
clickhouse-client --password
```

1. バージョン文字列を確認して、EDB ビルドを検証します。

```
SELECT value FROM system.build_options WHERE name = VERSION_OFFICIAL;
```

出力には、EDB パッケージリリースの (EDB Build) が含まれます。

クラスターの展開

クラスター展開は、シャーディング、レプリケーション、またはその両方を使用して、複数のサーバーノードにデータを分散します。レプリケーションには、少なくとも3つのノードで実行されている ClickHouse Keeper が必要です。シャーディング、レプリケーション、および Keeper ロールの説明については、[ClickHouse architecture](#) を参照してください。

次の手順では、5 ノード展開について説明します。それぞれ2つのレプリカを持つ2つのシャードに4つのサーバーノード。Keeper はノード1とノード2に同じ場所にあり、ノード5で専用サービスとして実行されます。独自のトポロジに合わせて構成を調整します。

Node	Role	Shard	Replica
node-1	ClickHouse server + Keeper	1	1
node-2	ClickHouse server + Keeper	1	2
node-3	ClickHouse server	2	1
node-4	ClickHouse server	2	2
node-5	Keeper		

パッケージのインストール

1. すべてのノードで EDB リポジトリを設定します。

```
export EDB_SUBSCRIPTION_TOKEN=<your-token>
export EDB_SUBSCRIPTION_PLAN=clickhouse
curl -sSLf "https://downloads.enterprisedb.com/$EDB_SUBSCRIPTION_TOKEN/$EDB_
→SUBSCRIPTION_PLAN/setup.rpm.sh" | sudo -E bash
```

1. 各ノードにロールに適切なパッケージをインストールします。

各サーバーノードで

```
sudo dnf install -y \
  edb-clickhouse-common-static \
  edb-clickhouse-server \
  edb-clickhouse-client
```

コロケーション展開の場合、edb-clickhouse-server には既に clickhouse-keeper.service が含まれているため、追加パッケージは必要ありません。

各専用の Keeper ノードで

```
sudo dnf install -y edb-clickhouse-keeper
```

Keeper の設定

サービスを開始する前に、すべての Keeper ノードを構成します。1 つのノードの障害を許容するには、少なくとも 3 つの Keeper ノードを使用します。

1. RPM は、デフォルトの/etc/clickhouse-keeper/keeper_config.xml をインストールします。各 Keeper ノードの内容を次のものに置き換え、<server_id> をノードごとに一意の整数 1、2、3 に設定します。

```
<clickhouse>
  <listen_host>0.0.0.0</listen_host>
  <keeper_server>
    <tcp_port>9181</tcp_port>
    <server_id>1</server_id><!-- Set to 1, 2, or 3 depending on the node -->
    <log_storage_path>/var/lib/clickhouse/coordination/log</log_storage_path>
    <snapshot_storage_path>/var/lib/clickhouse/coordination/snapshots</snapshot_
→storage_path>
    <coordination_settings>
      <operation_timeout_ms>10000</operation_timeout_ms>
      <session_timeout_ms>30000</session_timeout_ms>
      <raft_logs_level>information</raft_logs_level>
    </coordination_settings>
    <raft_configuration>
      <server><id>1</id><hostname><node-1-ip></hostname><port>9234</port></
→server>
      <server><id>2</id><hostname><node-2-ip></hostname><port>9234</port></
```

(次のページに続く)

(前のページからの続き)

```

→server>
    <server><id>3</id><hostname><node-5-ip></hostname><port>9234</port></
→server>
    </raft_configuration>
    </keeper_server>
</clickhouse>

```

サーバーの構成

各サーバーノードには、リモートサーバー、Keeper エンドポイント、およびノード独自のシャードとレプリカ ID を定義するクラスター構成が必要です。

1. 各サーバーノードで/etc/clickhouse-server/config.d/cluster.xml を作成し、各ノードの<macros> セクションの<shard> および<replica> 値を調整します。

Node	<shard>	<replica>
node-1	01	01
node-2	01	02
node-3	02	01
node-4	02	02

```

<clickhouse>
  <listen_host>0.0.0.0</listen_host>
  <remote_servers>
    <my_cluster>
      <shard>
        <internal_replication>true</internal_replication>
        <replica><host><node-1-ip></host><port>9000</port></replica>
        <replica><host><node-2-ip></host><port>9000</port></replica>
      </shard>
      <shard>
        <internal_replication>true</internal_replication>
        <replica><host><node-3-ip></host><port>9000</port></replica>
        <replica><host><node-4-ip></host><port>9000</port></replica>
      </shard>
    </my_cluster>
  </remote_servers>
  <zookeeper>
    <node><host><node-1-ip></host><port>9181</port></node>
    <node><host><node-2-ip></host><port>9181</port></node>
    <node><host><node-5-ip></host><port>9181</port></node>
  </zookeeper>
  <macros>
    <cluster>my_cluster</cluster>

```

(次のページに続く)

(前のページからの続き)

```

<shard>01</shard><!-- Adjust per node, see table above -->
<replica>01</replica><!-- Adjust per node, see table above -->
</macros>
<distributed_ddl>
  <path>/clickhouse/task_queue/ddl</path>
</distributed_ddl>
</clickhouse>

```

サービスの開始

1. すべての Keeper ノードで Keeper を起動します。各 Keeper ノードで

```

sudo systemctl enable clickhouse-keeper
sudo systemctl start clickhouse-keeper

```

1. 各 Keeper ノードが正常であることを確認します。各 Keeper ノードで次を実行します。正常なノードは imok で応答します。

```

clickhouse-keeper-client -h 127.0.0.1 -p 9181 -q "ruok"
__OUTPUT__
imok

```

1. クォーラムが 1 つの Keeper ノードから形成されていることを確認します。2 の値は、2 つのフォロワーがリーダーと同期していることを意味し、3 つのキーパーノードすべてがクォーラムを形成していることを確認します。

```

clickhouse-keeper-client -h 127.0.0.1 -p 9181 -q "mntr" | grep zk_synced_followers
__OUTPUT__
zk_synced_followers 2

```

1. すべてのサーバーノードでサーバーを起動します。

```

sudo systemctl enable clickhouse-server
sudo systemctl start clickhouse-server

```

1. サーバーが各サーバーノードで実行されていることを確認します。

```

sudo systemctl status clickhouse-server

```

クラスターへの接続

1. edb-clickhouse-client がインストールされているホストから任意のサーバーノードに接続します。

```

clickhouse-client -h <server-node-ip>

```

デフォルトでは、default ユーザーにはパスワードがありません。設定するには、接続した後に次を実行します。

```
ALTER USER default IDENTIFIED BY yourpassword;
```

次に、次のように接続します。

```
clickhouse-client -h <server-node-ip> --password
```

1. すべてのサーバーノードが登録されていることを確認します。system.clusters はサーバーノードのみをリストするため、専用の Keeper ノード 5 は表示されません。

```
SELECT cluster, shard_num, replica_num, host_name
FROM system.clusters
WHERE cluster = my_cluster;
```

__OUTPUT__

cluster	shard_num	replica_num	host_name
my_cluster	1	1	<node-1-ip>
my_cluster	1	2	<node-2-ip>
my_cluster	2	1	<node-3-ip>
my_cluster	2	2	<node-4-ip>

レプリケーションとシャーディングのテスト

ローカルの ReplicatedMergeTree テーブルとその上に Distributed テーブルを作成します。いくつかの行を挿入し、両方のテーブルを照会して、各シャード内でレプリケーションが機能していること、および分散テーブルがすべてのシャードのデータを集計することを確認します。

1. ローカルストレージテーブルを作成します。ReplicatedMergeTree エンジン、各ノードにデータを物理的に保存し、シャード内でレプリケートします。

```
CREATE TABLE events ON CLUSTER my_cluster (
    event_id UInt32,
    user_id   UInt32,
    event_type String,
    created_at DateTime
) ENGINE = ReplicatedMergeTree(/clickhouse/tables/{shard}/events, {replica})
ORDER BY event_id;
```

1. events の上に Distributed テーブルを作成します。ローカルテーブルとは異なり、データは保存されません。クエリーをすべてのシャードにわたって events にルーティングし、結果を集計します。

```
CREATE TABLE events_dist ON CLUSTER my_cluster (
    event_id UInt32,
    user_id   UInt32,
    event_type String,
    created_at DateTime
) ENGINE = Distributed(my_cluster, default, events, rand());
```

1. `events_dist` を介していくつかの行を挿入します。 `rand()` シャーディングキーは、各行をシャード全体に分散します。

```
INSERT INTO events_dist VALUES
```

```
(1, 101, login, now()),
(2, 102, purchase, now()),
(3, 103, logout, now()),
(4, 104, login, now());
```

1. 任意のサーバーノードのローカルテーブルを照会します。各ノードは、そのシャードの行のみを結果ます。特定の行は、`rand()` がそれらを分散した方法によって異なります。

```
SELECT * FROM events;
```

```
--OUTPUT--
```

event_id	user_id	event_type	created_at
2	102	purchase	2026-06-10 14:48:15
3	103	logout	2026-06-10 14:48:15

1. 任意のノードから `events_dist` を照会して、すべての行がシャードにわたって返されることを確認します。

```
SELECT * FROM events_dist ORDER BY event_id;
```

```
--OUTPUT--
```

event_id	user_id	event_type	created_at
1	101	login	2026-06-10 14:48:15
2	102	purchase	2026-06-10 14:48:15
3	103	logout	2026-06-10 14:48:15
4	104	login	2026-06-10 14:48:15

4.2.2 Installing EDB Postgres AI for ClickHouse with Docker

EDB は、Linux パッケージと同じ RPM からビルドされたコンテナイメージを公開します。イメージは Rocky Linux 10minimal ベースであり、x86-64 で利用でき、`clickhouse-server`、`clickhouse-keeper`、および `clickhouse-client` が含まれています。

前提条件

- Docker Engine 20.10 以降と `Compose v2 docker compose` コマンド
- EDB サブスクリプショントークン、[EDB customer portal](#) から入手できます。

イメージのプル

EDB コンテナレジストリにログインし、イメージをプルします。

```
export EDB_SUBSCRIPTION_TOKEN=<your-token>
docker login docker.enterprisedb.com \
  --username <your-edb-email> \
  --password "$EDB_SUBSCRIPTION_TOKEN"
docker pull docker.enterprisedb.com/edb-clickhouse-server:26.3.12.3
```

シングルノードサーバーの実行

開発とテストの場合、単一の ClickHouse Server ノードを実行します。

1. サーバーを起動します。

```
docker run -d \
  --name clickhouse-server \
  --ulimit nofile=262144:262144 \
  -p 8123:8123 \
  -p 9000:9000 \
  docker.enterprisedb.com/edb-clickhouse-server:26.3.12.3
```

ポート 8123 は、HTTP インターフェイスを公開します。ポート 9000 は、clickhouse-client およびほとんどのドライバーが使用するネイティブ TCP インターフェイスを公開します。

コンテナを再起動してもデータを持続するには、ボリュームマウントを追加します。

```
docker run -d \
  --name clickhouse-server \
  --ulimit nofile=262144:262144 \
  -p 8123:8123 \
  -p 9000:9000 \
  -v /local/path/clickhouse-data:/var/lib/clickhouse \
  docker.enterprisedb.com/edb-clickhouse-server:26.3.12.3
```

2. デフォルトのサーバー設定をオーバーライドするには、構成ディレクトリをマウントします。

```
docker run -d \
  --name clickhouse-server \
  --ulimit nofile=262144:262144 \
  -p 8123:8123 \
  -p 9000:9000 \
  -v /local/path/config:/etc/clickhouse-server/config.d \
  docker.enterprisedb.com/edb-clickhouse-server:26.3.12.3
```

ClickHouse は、起動時に config.d のすべての.xml ファイルをメイン構成にマージします。

3. ClickHouse クライアントに接続します。

```
docker exec -it clickhouse-server clickhouse-client
```

4. EDB ビルドを確認します。

```
SELECT value FROM system.build_options WHERE name = VERSION_OFFICIAL;
```

出力には、EDB パッケージリリースの (EDB Build) が含まれます。

マルチノードクラスターの実行

Docker Compose を使用して、マルチノードクラスターを展開します。EDB 提供のサーバーイメージには `clickhouse-server` と `clickhouse-keeper` の両方が含まれているため、クラスター内のすべてのノードは単一のイメージから実行されます。

次の例では、3つの専用キーパーノード合計7つのコンテナを使用した2シャード、2レプリカクラスターをセットアップします。シャーディング、レプリケーション、および Keeper ロールの説明については、[ClickHouse architecture](#) を参照してください。

構成ファイルの作成

次のディレクトリ構造を作成します。

```
cluster/
├── docker-compose.yml
├── keeper-01/
│   └── keeper_config.xml
├── keeper-02/
│   └── keeper_config.xml
├── keeper-03/
│   └── keeper_config.xml
├── server-01/
│   ├── config.xml
│   └── users.xml
├── server-02/
│   ├── config.xml
│   └── users.xml
├── server-03/
│   ├── config.xml
│   └── users.xml
└── server-04/
    ├── config.xml
    └── users.xml
```

各ディレクトリに、そのノードロールの構成ファイルを設定します。

キーパーノード

3つの Keeper ノードは、Raft クォーラムを形成します。それぞれに一意の `server_id` があります。各キーパーディレクトリに `keeper_config.xml` を作成し、`server_id` を 1、2、および 3 にそれぞれ設定します。

keeper_config.xml

```

<clickhouse replace="true">
  <logger>
    <level>information</level>
    <console>1</console>
  </logger>
  <listen_host>0.0.0.0</listen_host>
  <keeper_server>
    <tcp_port>9181</tcp_port>
    <server_id>1</server_id>
    <log_storage_path>/var/lib/clickhouse/coordination/log</log_storage_path>
    <snapshot_storage_path>/var/lib/clickhouse/coordination/snapshots</snapshot_
→ storage_path>
    <coordination_settings>
      <operation_timeout_ms>10000</operation_timeout_ms>
      <session_timeout_ms>30000</session_timeout_ms>
      <raft_logs_level>information</raft_logs_level>
    </coordination_settings>
    <raft_configuration>
      <server>
        <id>1</id>
        <hostname>keeper-01</hostname>
        <port>9234</port>
      </server>
      <server>
        <id>2</id>
        <hostname>keeper-02</hostname>
        <port>9234</port>
      </server>
      <server>
        <id>3</id>
        <hostname>keeper-03</hostname>
        <port>9234</port>
      </server>
    </raft_configuration>
  </keeper_server>
</clickhouse>

```

サーバーノード

各サーバーノードには、クラスタトポロジ、Keeper エンドポイント、およびシャード/レプリカマクロが必要です。各サーバーディレクトリに `config.xml` を作成します。ノード間で異なる値は、`<macros>` セクションの `<shard>` および `<replica>` エントリのみです。

Node	<shard>	<replica>
server-01	01	01
server-02	02	01
server-03	01	02
server-04	02	02

config.xml

```
<clickhouse replace="true">
  <logger>
    <level>information</level>
    <console>1</console>
  </logger>
  <display_name>cluster node 01</display_name>
  <listen_host>0.0.0.0</listen_host>
  <http_port>8123</http_port>
  <tcp_port>9000</tcp_port>
  <distributed_ddl>
    <path>/clickhouse/task_queue/ddl</path>
  </distributed_ddl>
  <remote_servers>
    <my_cluster>
      <shard>
        <internal_replication>true</internal_replication>
        <replica>
          <host>server-01</host>
          <port>9000</port>
        </replica>
        <replica>
          <host>server-03</host>
          <port>9000</port>
        </replica>
      </shard>
      <shard>
        <internal_replication>true</internal_replication>
        <replica>
          <host>server-02</host>
          <port>9000</port>
        </replica>
        <replica>
          <host>server-04</host>
          <port>9000</port>
        </replica>
      </shard>
    </my_cluster>
  </remote_servers>
</clickhouse>
```

(次のページに続く)

(前のページからの続き)

```

</remote_servers>
<zookeeper>
  <node>
    <host>keeper-01</host>
    <port>9181</port>
  </node>
  <node>
    <host>keeper-02</host>
    <port>9181</port>
  </node>
  <node>
    <host>keeper-03</host>
    <port>9181</port>
  </node>
</zookeeper>
<macros>
  <shard>01</shard>
  <replica>01</replica>
</macros>
</clickhouse>

```

ユーザー設定

4つのサーバーノードはすべて同じ `users.xml` を使用します。これを各サーバーディレクトリにコピーし、クラスター起動後にパスワードを設定します。

users.xml

```

<clickhouse replace="true">
  <profiles>
    <default>
      <max_memory_usage>10000000000</max_memory_usage>
      <load_balancing>in_order</load_balancing>
      <log_queries>1</log_queries>
    </default>
  </profiles>
  <users>
    <default>
      <profile>default</profile>
      <networks>
        <ip>::/0</ip>
      </networks>
      <quota>default</quota>
      <access_management>1</access_management>
    </default>
  </users>

```

(次のページに続く)

(前のページからの続き)

```

<quotas>
  <default>
    <interval>
      <duration>3600</duration>
      <queries>0</queries>
      <errors>0</errors>
      <result_rows>0</result_rows>
      <read_rows>0</read_rows>
      <execution_time>0</execution_time>
    </interval>
  </default>
</quotas>
</clickhouse>

```

Compose ファイルの作成

cluster/ ディレクトリに docker-compose.yml を作成します。

```

services:
  keeper-01:
    image: docker.enterprisedb.com/edb-clickhouse-server:26.3.12.3
    container_name: keeper-01
    hostname: keeper-01
    entrypoint: ["/usr/bin/clickhouse-keeper", "--config-file=/etc/clickhouse-keeper/
→keeper_config.xml"]
    volumes:
      - ./keeper-01/keeper_config.xml:/etc/clickhouse-keeper/keeper_config.xml

  keeper-02:
    image: docker.enterprisedb.com/edb-clickhouse-server:26.3.12.3
    container_name: keeper-02
    hostname: keeper-02
    entrypoint: ["/usr/bin/clickhouse-keeper", "--config-file=/etc/clickhouse-keeper/
→keeper_config.xml"]
    volumes:
      - ./keeper-02/keeper_config.xml:/etc/clickhouse-keeper/keeper_config.xml

  keeper-03:
    image: docker.enterprisedb.com/edb-clickhouse-server:26.3.12.3
    container_name: keeper-03
    hostname: keeper-03
    entrypoint: ["/usr/bin/clickhouse-keeper", "--config-file=/etc/clickhouse-keeper/
→keeper_config.xml"]
    volumes:
      - ./keeper-03/keeper_config.xml:/etc/clickhouse-keeper/keeper_config.xml

```

(次のページに続く)

(前のページからの続き)

```
server-01:
  image: docker.enterprisedb.com/edb-clickhouse-server:26.3.12.3
  container_name: server-01
  hostname: server-01
  ulimits:
    nofile: { soft: 262144, hard: 262144 }
  ports:
    - "127.0.0.1:18123:8123"
    - "127.0.0.1:19000:9000"
  volumes:
    - ./server-01/config.xml:/etc/clickhouse-server/config.d/config.xml
    - ./server-01/users.xml:/etc/clickhouse-server/users.d/users.xml
  depends_on: [keeper-01, keeper-02, keeper-03]

server-02:
  image: docker.enterprisedb.com/edb-clickhouse-server:26.3.12.3
  container_name: server-02
  hostname: server-02
  ulimits:
    nofile: { soft: 262144, hard: 262144 }
  ports:
    - "127.0.0.1:18124:8123"
    - "127.0.0.1:19001:9000"
  volumes:
    - ./server-02/config.xml:/etc/clickhouse-server/config.d/config.xml
    - ./server-02/users.xml:/etc/clickhouse-server/users.d/users.xml
  depends_on: [keeper-01, keeper-02, keeper-03]

server-03:
  image: docker.enterprisedb.com/edb-clickhouse-server:26.3.12.3
  container_name: server-03
  hostname: server-03
  ulimits:
    nofile: { soft: 262144, hard: 262144 }
  ports:
    - "127.0.0.1:18125:8123"
    - "127.0.0.1:19002:9000"
  volumes:
    - ./server-03/config.xml:/etc/clickhouse-server/config.d/config.xml
    - ./server-03/users.xml:/etc/clickhouse-server/users.d/users.xml
  depends_on: [keeper-01, keeper-02, keeper-03]

server-04:
  image: docker.enterprisedb.com/edb-clickhouse-server:26.3.12.3
```

(次のページに続く)

(前のページからの続き)

```

container_name: server-04
hostname: server-04
ulimits:
  nofile: { soft: 262144, hard: 262144 }
ports:
  - "127.0.0.1:18126:8123"
  - "127.0.0.1:19003:9000"
volumes:
  - ./server-04/config.xml:/etc/clickhouse-server/config.d/config.xml
  - ./server-04/users.xml:/etc/clickhouse-server/users.d/users.xml
depends_on: [keeper-01, keeper-02, keeper-03]

```

クラスターの起動と検証

1. cluster/ ディレクトリから、すべてのコンテナを起動します。

```
docker compose up -d
```

2. server-01 に接続します。

```
docker exec -it server-01 clickhouse-client
```

3. すべてのサーバーノードが登録されていることを確認します。system.clusters はサーバーノードのみをリストするため、専用の Keeper コンテナは表示されません。

```
SELECT cluster, shard_num, replica_num, host_name
```

```
FROM system.clusters
```

```
WHERE cluster = my_cluster;
```

```
__OUTPUT__
```

cluster	shard_num	replica_num	host_name
my_cluster	1	1	server-01
my_cluster	1	2	server-03
my_cluster	2	1	server-02
my_cluster	2	2	server-04

4. すべてのノードにわたって Keeper の接続を確認します。クエリーは clusterAllReplicas を使用してすべてのサーバーノードに対して実行され、それぞれが認識するルート z ノードを結果ます。行を返すすべてのノードは、すべてのサーバーが Keeper に到達できることを確認します。

```
SELECT hostName(), name, path
```

```
FROM clusterAllReplicas(my_cluster, system.zookeeper)
```

```
WHERE path = /;
```

```
__OUTPUT__
```

hostName()	name	path
server-01	keeper	/

(次のページに続く)

(前のページからの続き)

server-01	clickhouse	/
server-02	keeper	/
server-02	clickhouse	/
server-03	keeper	/
server-03	clickhouse	/
server-04	keeper	/
server-04	clickhouse	/

レプリケーションとシャーディングのテスト

ローカルの ReplicatedMergeTree テーブルとその上に Distributed テーブルを作成します。いくつかの行を挿入し、両方のテーブルを照会して、各シャード内でレプリケーションが機能していること、および分散テーブルがすべてのシャードのデータを集計することを確認します。

1. ローカルストレージテーブルを作成します。ReplicatedMergeTree エンジン、各ノードにデータを物理的に保存し、シャード内でレプリケートします。

```
CREATE TABLE events ON CLUSTER my_cluster (
    event_id    UInt32,
    user_id     UInt32,
    event_type  String,
    created_at  DateTime
) ENGINE = ReplicatedMergeTree(/clickhouse/tables/{shard}/events, {replica})
ORDER BY event_id;
```

1. events の上に Distributed テーブルを作成します。ローカルテーブルとは異なり、データは保存されません。クエリーをすべてのシャードにわたって events にルーティングし、結果を集計します。

```
CREATE TABLE events_dist ON CLUSTER my_cluster (
    event_id    UInt32,
    user_id     UInt32,
    event_type  String,
    created_at  DateTime
) ENGINE = Distributed(my_cluster, default, events, rand());
```

1. events_dist を介していくつかの行を挿入します。rand() シャーディングキーは、各行をシャード全体に分散します。

```
INSERT INTO events_dist VALUES
(1, 101, login,    now()),
(2, 102, purchase, now()),
(3, 103, logout,   now()),
(4, 104, login,    now());
```

1. 任意のサーバーノードのローカルテーブルを照会します。各ノードは、そのシャードの行のみを結果ます。特定の行は、rand() がそれらを分散した方法によって異なります。

```
SELECT * FROM events;
```

```
--OUTPUT--
```

event_id	user_id	event_type	created_at
2	102	purchase	2026-06-10 14:48:15
3	103	logout	2026-06-10 14:48:15

1. 任意のノードから `events_dist` を照会して、すべての行がシャードにわたって返されることを確認します。

```
SELECT * FROM events_dist ORDER BY event_id;
```

```
--OUTPUT--
```

event_id	user_id	event_type	created_at
1	101	login	2026-06-10 14:48:15
2	102	purchase	2026-06-10 14:48:15
3	103	logout	2026-06-10 14:48:15
4	104	login	2026-06-10 14:48:15

4.2.3 Installing EDB Postgres AI for ClickHouse on Kubernetes

EDB ClickHouse Kubernetes オペレーターは、Kubernetes での ClickHouse 展開を管理し、クラスターのプロビジョニング、構成、アップグレード、およびスケーリングを処理します。

Helm チャートを介してインストールされると、以下を作成します。

- 4つのカスタムリソース定義 CRD
- `clickhouseinstallations` 略称 `chi`
- `clickhouseinstallationtemplates`
- `clickhouseoperatorconfigurations`
- `clickhousekeeperinstallations` 略称 `chk`
- オペレーターおよびメトリックエクスポーターサイドカー 2/2 `Ready` を実行する展開。
- オペレーターがクラスター全体で ClickHouse リソースを管理するために必要なサービスアカウント、ロール、クラスターロール、およびバインディング。

前提条件

- 実行中の Kubernetes クラスター 1.26 以降
- `kubectl` はクラスターと通信するように構成
- Helm 3.8 以降
- `clickhouse` リポジトリへの読み取りアクセスを持つ EDB Cloudsmith アカウント。そこから 2つの資格情報を使用します。

- **資格トークン** — Helm チャートリポジトリの URL に埋め込まれています。 <https://www.enterprisedb.com/accounts/login> にある EDB アカウントから取得します。
- **Cloudsmith API キー** — Kubernetes が `docker.enterprisedb.com` からイメージをプルするときに Docker パスワードとして使用されます。 <https://cloudsmith.io/user/settings/api/> から取得します EDB SSO を介してログインします。

イメージプルシークレットの作成

Kubernetes が `docker.enterprisedb.com` から EDB コンテナイメージをプルするには資格情報が必要です。オペレーターをインストールする名前空間に `docker-registry` シークレットを作成します。

```
kubectl create namespace clickhouse
kubectl create secret docker-registry cs-pull \
  --namespace=clickhouse \
  --docker-server=docker.enterprisedb.com \
  --docker-username=<your-cloudsmith-username> \
  --docker-password=<your-cloudsmith-api-key>
```

Helm インストールでオペレーターポッドが独自のイメージをプルできるように、すべての ClickHouseInstallation ClickHouse ポッドが `edb-clickhouse-server` をプルできるように、の両方で `cs-pull` を参照します。

オペレーターのインストール

1. EDB Helm リポジトリを追加します。 `<entitlement-token>` を EDB アカウントのトークンに置き換え、 `<cs-repo>` をリリースノートにリストされている EDB Cloudsmith リポジトリ名に置き換えます。

```
helm repo add edb \
  "https://dl.cloudsmith.io/<entitlement-token>/enterprisedb/<cs-repo>/helm/charts/"
helm repo update edb
```

1. チャートをインストールします。 EDB リリースサフィックス `-NedbM` は SemVer プレリリースタグであり、 `helm search` がデフォルトでこれらをフィルタリングするため、 `--version` を明示的に指定します。

```
helm upgrade --install clickhouse-operator \
  edb/edb-clickhouse-operator-chart \
  --version <operator-version> \
  --namespace clickhouse \
  --set imagePullSecrets[0].name=cs-pull
```

1. オペレーターが実行されていることを確認します。

```
kubectl get pods -n clickhouse
```

オペレーターポッドは、約 30 秒以内に 2/2 Ready を報告します。2つのコンテナは、オペレーターとメトリックエクスポートサイドカーです。

1. 登録された CRD を確認します。

```
kubectl get crds | grep altinity
```

上記の 4 つの CRD すべてが出力に表示されます。

ClickHouse クラスターの作成

オペレーターは ClickHouseInstallation リソースを監視し、実行中の ClickHouse ポッドにそれらを調整します。clickhouse-installation.yaml という名前のファイルを作成します。

```
apiVersion: clickhouse.altinity.com/v1
kind: ClickHouseInstallation
metadata:
  name: my-clickhouse
  namespace: clickhouse
spec:
  configuration:
    clusters:
      - name: my-cluster
        templates:
          podTemplate: edb-clickhouse
        layout:
          shardsCount: 1
          replicasCount: 1
  templates:
    podTemplates:
      - name: edb-clickhouse
        spec:
          imagePullSecrets:
            - name: cs-pull
          containers:
            - name: clickhouse
              image: docker.enterprisedb.com/<cs-repo>/edb-clickhouse-server:<server-
↪version>
              ports:
                - name: http
                  containerPort: 8123
                - name: client
                  containerPort: 9000
```

imagePullSecrets はポッドテンプレートの内部にあることに注意してください。これは、オペレーターポッド自分自身ではなく、オペレーターが作成する ClickHouse ポッドに適用されます。どちらもそれが必要です。

<server-version> は、edb-clickhouse-server の Docker イメージタグです。Docker タグには、RPM パッケージが使用する -NedbM サフィックスがありません。上流バージョンのみ 26.3.12.1 など。オペレーターバージョンと互換性のあるバージョンについては、リリースノートを確認してください。

リソースを適用し、クラスターが起動するのを確認します。

```
kubectl apply -f clickhouse-installation.yaml
kubectl get chi -n clickhouse -w
```

STATUS が Completed を表示するまで待ちます。これには約 60～90 秒かかります。

ClickHouse に接続する

オペレーターは、クラスターごとにヘッドレスサービスを作成します。ポート転送を使用してサービス名を介して接続します。

```
kubectl port-forward -n clickhouse svc/clickhouse-my-clickhouse 9000:9000
```

次に、別のターミナルで

```
clickhouse-client --host 127.0.0.1 --port 9000
```

または、クラスター内でワンショットクエリーを実行します。

```
kubectl exec -it -n clickhouse \
  "$(kubectl get pods -n clickhouse -l clickhouse.altinity.com/chi=my-clickhouse -o_
  ↪name | head -1)" \
  -- clickhouse-client --query "SELECT version()"
```

返されたバージョンには (EDB Build) サフィックスが含まれ、EDB ディストリビューションを実行していることを確認します。

アンインストール

```
kubectl delete chi my-clickhouse -n clickhouse           # operator deletes pods and_
  ↪services
helm uninstall clickhouse-operator -n clickhouse         # removes the operator_
  ↪Deployment; CRDs survive
kubectl delete secret cs-pull -n clickhouse               # optional
```

CRD は、helm uninstall によって意図的に削除されません。CRD を削除すると、他の名前空間のものも含む、すべての依存 ClickHouseInstallation が削除されます。クラスター内に他のオペレーターインスタンスがない場合にのみそれらを削除します。

```
kubectl delete crd \
  clickhouseinstallations.clickhouse.altinity.com \
  clickhouseinstallationtemplates.clickhouse.altinity.com \
  clickhouseoperatorconfigurations.clickhouse.altinity.com \
  clickhousekeeperinstallations.clickhouse-keeper.altinity.com
```


第5章 Getting started with ClickHouse

ClickHouse インスタンスに接続し、EDB 固有の機能を探索し、さらに進むためのリソースを見つけます。

5.1 ClickHouse に接続する

ClickHouse がインストールして実行されたら、`clickhouse-client` を使用して接続します。

```
clickhouse-client
```

EDB パッケージビルドを実行していることを確認します。

```
SELECT value FROM system.build_options WHERE name = VERSION_OFFICIAL;
```

5.2 統合

- *Connecting ClickHouse to WarehousePG* PostgreSQL テーブルエンジンを使用して WarehousePG テーブルの読み取りおよび書き込み
- *Integrating with Iceberg catalogs* Amazon S3 または Azure Blob Storage に保存されている Iceberg テーブルを照会します

5.3 関連項目

ここで説明している内容を超える SQL リファレンス、構成、および操作ガイダンスについては、上流の ClickHouse ドキュメントを参照してください。

- [Creating tables](#)
- [Inserting data](#)
- [Writing queries](#)
- [SQL reference](#)
- [Table engines](#)
- [Server configuration](#)
- [Best practices](#)

第6章 Connecting ClickHouse to WarehousePG

ETL パイプラインやデータの複製を使用せずに、ClickHouse から WarehousePG データを照会し、WarehousePG にデータを書き込みます。ClickHouse は、PostgreSQL テーブルエンジンを使用して WarehousePG に接続します。これは、WarehousePG テーブルを ClickHouse テーブル定義にマップし、標準の Postgres ワイヤプロトコルを介して読み取りおよび書き込みを処理します。

注釈

PostgreSQL エンジンは WarehousePG コーディネーターにのみ接続します。WarehousePG の超並列処理 MPP アーキテクチャを採用していないため、クエリはシングルノードの Postgres クエリとしてコーディネーターを介して実行されます。

6.1 ClickHouse からの接続を受け入れるように WarehousePG を構成する

1. コーディネーターホストで、`$COORDINATOR_DATA_DIRECTORY/postgresql.conf` の `listen_addresses` が '*' またはネットワーク範囲に設定されていることを確認します。まだである場合は、更新します。

```
listen_addresses = *
```

1. ClickHouse ホストからの接続を許可するエントリーを `pg_hba.conf` に追加します。
<clickhouse-host-network> を ClickHouse ホストのネットワーク範囲に置き換えます。

```
echo "host    <database>    <user>    <clickhouse-host-network>/24    md5" >>
→$COORDINATOR_DATA_DIRECTORY/pg_hba.conf
```

1. コーディネーター構成をリロードします。

```
gpstop -u
```

1. ClickHouse ホストから、WarehousePG コーディネーターに到達可能であることを確認します。

```
psql -h <coordinator-ip> -p <port> -d <database> -U <user>
```

6.2 ClickHouse でリンクテーブルを作成する

PostgreSQL テーブルエンジンを使用して、WarehousePG テーブルを ClickHouse テーブル定義にマップします。ClickHouse テーブルに対するクエリは、作成されると、リアルタイムで WarehousePG に直接送信され、ローカルにコピーまたはキャッシュされるデータはありません。

1. WarehousePG で、ClickHouse に公開するテーブルを作成します。例

```
CREATE TABLE analytics_data (
  id          integer primary key,
  event_type  varchar(50),
  event_time  timestamp,
  value       numeric(12, 4)
);

INSERT INTO analytics_data (id, event_type, event_time, value)
VALUES
  (1, page_view, 2025-01-01 10:00:00, 1.0),
  (2, click,    2025-01-01 10:01:00, 2.5);
```

1. ClickHouse サーバーから、clickhouse-client セッションを開きます。

```
clickhouse-client
```

1. ENGINE = PostgreSQL を使用して WarehousePG テーブルにマッピングする ClickHouse でデータベースとテーブルを作成します。

```
CREATE DATABASE whpg_connect;

CREATE TABLE whpg_connect.analytics_data
(
  id          UInt64,
  event_type  String,
  event_time  DateTime,
  value       Float64
)
ENGINE = PostgreSQL(
  <coordinator-ip>:<port>, -- host:port of the WarehousePG coordinator
  <database>,              -- WarehousePG database name
  analytics_data,          -- table name in WarehousePG
  <user>,                  -- WarehousePG user
  <password>               -- password (stored encrypted in ClickHouse)
);
```

ENGINE = PostgreSQL 構文は、host:port、データベース、テーブル、ユーザー、パスワードの5つの位置パラメーターを受け入れます。オプションの6番目のパラメーターは、スキーマ名デフォルト public を指定します。WarehousePG が trust 認証で構成されている場合、パスワードに空の文字列'' を渡します。

1. 定義を検査して、テーブルが正しくリンクされていることを確認します。

```
DESCRIBE TABLE whpg_connect.analytics_data;
```

または、CREATE TABLE ステートメントを完全に表示します。

```
SHOW CREATE TABLE whpg_connect.analytics_data;
```

パスワードは出力に [HIDDEN] と表示されます。

6.3 ClickHouse から WarehousePG データを読み取る

ClickHouse テーブルを照会します。ClickHouse は WarehousePG に接続し、COPY ... TO STDOUT ステートメントを実行し、行指向ストリームを到着時に列指向形式に変換します。

```
SELECT * FROM whpg_connect.analytics_data;
```

__OUTPUT__

id	event_type	event_time	value
1	page_view	2025-01-01 10:00:00	1
2	click	2025-01-01 10:01:00	2.5

ClickHouse テーブルの作成後に WarehousePG に挿入されたデータは、次の SELECT にすぐに表示されます。PostgreSQL エンジンを使用する場合、ClickHouse は WarehousePG データをローカルにキャッシュしません。

6.4 ClickHouse から WarehousePG へのデータの書き込み

ClickHouse テーブルで INSERT を使用して、WarehousePG にデータを書き込みます。ClickHouse は、WarehousePG 側で挿入を COPY ... FROM STDIN ステートメントに変換します。

-- Run in ClickHouse

```
INSERT INTO whpg_connect.analytics_data (id, event_type, event_time, value)
VALUES (3, purchase, 2025-01-01 10:05:00, 99.99);
```

行が WarehousePG に表示されることを確認します。

-- Run in WarehousePG

```
SELECT * FROM analytics_data;
```

6.5 クエリフローの理解

ClickHouse で EXPLAIN を使用して、システム間でデータがどのように移動するかを確認します。

```
EXPLAIN SELECT * FROM whpg_connect.analytics_data;
```

__OUTPUT__

explain

(次のページに続く)

(前のページからの続き)

```
| Expression ((Project names + (Projection + Change column names to column_
→identifiers)))
| ReadFromPostgreSQL
|
```

ReadFromPostgreSQL ステップは、コーディネーターへの TCP 接続を確立し、SQL クエリーを送信し、受信行指向ストリームを ClickHouse の列指向形式に変換します。データが Expression ステップに到達するまでに、データは既に列状形式になっています。

第7章 Integrating with Iceberg catalogs

ClickHouse を外部の Iceberg カタログに接続して、データを移動せずにデータレイクで高速分析クエリを実行します。Iceberg カタログは、どのテーブルが存在するか、データがオブジェクトストレージのどこにあるか、および誰がアクセスしているかを追跡するガバナンスとメタデータレイヤーです。

Spark ジョブ、Databricks、dbt などのさまざまなパイプラインとツールは、オブジェクトストレージにデータを書き込むことができ、カタログはそのすべての単一の一貫したビューを提供します。ClickHouse、Spark、Trino など、Iceberg プロトコルを実装するクエリエンジンは、同じカタログに接続し、同じデータを個別に照会できます。

注釈

ClickHouse の Iceberg カタログ統合は読み取り専用です。 *INSERT*、*UPDATE*、および *DELETE* 操作はサポートされていません。詳細は、[Known issues](#) を参照してください。

7.1 カタログへの接続

DataLakeCatalog データベースエンジンを使用して、ClickHouse を外部の Iceberg カタログに接続します。エンジンは実験的なものであり、使用前にオプトインが必要です。

1. カタログタイプの実験設定を有効にします。必要な設定はカタログによって異なります。完全なリストについては、[DataLakeCatalog reference](#) を参照してください。REST カタログの場合

```
SET allow_experimental_database_iceberg = 1;
```

1. 外部カタログにマッピングするデータベースを作成します。例

```
CREATE DATABASE my_catalog
ENGINE = DataLakeCatalog(https://catalog.example.com/api/iceberg/prj_abc123)
SETTINGS
  catalog_type = rest,
  warehouse = my-warehouse,
  auth_header = Authorization: Bearer eyJhbGciOiJSUzI1NiJ9.example;
```

そこで

- `catalog_type` はカタログタイプです。有効な値については、[DataLakeCatalog reference](#) を参照してください。 - `warehouse` は、カタログ内のウェアハウスまたはプロジェクトの識別子です。 - `auth_header` は、ベアラートークン認証の HTTP ヘッダーです。

ほとんどの REST カタログでは、上記のように、`auth_header` を介して静的ベアラートークンを使用して認証します。OAuth2 クライアント資格情報をサポートするカタログの場合、代わりに `catalog_credential` を使用します。

```
CREATE DATABASE my_catalog
ENGINE = DataLakeCatalog(https://catalog.example.com/api/iceberg/prj_abc123)
SETTINGS
    catalog_type = rest,
    warehouse = my-warehouse,
    catalog_credential = my-client-id:my-client-secret,
    oauth_server_uri = https://auth.example.com/oauth/token,
    auth_scope = catalog:read;
```

そこで

- `catalog_credential` は、`client_id:client_secret` 形式の OAuth2 クライアント資格情報です。
- `oauth_server_uri` は、OAuth2 承認サーバー URI です。 - `auth_scope` は、OAuth2 スcope です。
- 指定しない場合、デフォルトの `PRINCIPAL_ROLE:ALL`。

7.2 データの照会

データベースが作成されたら、使用可能なすべてのテーブルをリストします。

```
SHOW TABLES FROM my_catalog;
```

__OUTPUT__

name
public.calls
public.notes
analytics.events
duckdb_tpcds_sf_1.catalog_sales

テーブルは `namespace.table_name` 形式でリストされます。バッククォームを使用してそれらを参照します。

```
SELECT * FROM my_catalog.`public.calls` LIMIT 5;
```

__OUTPUT__

id	ts	data	category
1	2025-05-01 00:01:00.000000	data 1	bread
2	2025-05-01 00:02:00.000000	data 2	fruit
3	2025-05-01 00:03:00.000000	data 3	juice

集計、フィルタ、結合、ウィンドウファンクションを含む標準 SQL が適用されます。

```
SELECT category, count() AS total
FROM my_catalog.`public.calls`
```

(次のページに続く)

(前のページからの続き)

```
GROUP BY category  
ORDER BY total DESC;
```

ClickHouse は、パーティションプルーニングと述語プッシュダウンに Iceberg メタデータを使用して、クエリに必要な Parquet データファイルのみを読み取ります。