

TiDB

RDBMSのスケール課題を解消するNewSQL

長谷川 誠 [著]

Makoto Hasegawa

本多 康夫 [監修]

Yasuo Honda

PingCAP株式会社

実践入門

書き込み用
エンドポイントを
冗長化可能

バージョン
アップが
容易

タイムトラベル
クエリが可能

シャーディングが
不要

無停止で
効率よく
スケール可能

運用に
役立つ!

逆引き
コマンド
リファレンス
付き

アーキテクチャ、
様々な環境へのインストール、
各環境における操作方法を
詳しく解説

高可用性&高拡張可能性で
現代のアプリケーションの
ニーズに応える

この1冊でTiDBの
基本的な仕組みと
動かし方が
しっかり理解できる

本書に記載された内容は情報の提供のみを目的としています。したがって、本書の記述に従った運用は、必ずお客様自身の責任と判断によって行ってください。これらの情報の運用の結果について、技術評論社および著者はいかなる責任も負いません。

本書記載の情報は2025年2月時点のもので、ご利用時には変更されている場合もあります。

本書に掲載されているサンプルプログラムや実行結果、画面図などは、特定の設定に基づいた環境で再現される一例です。

ソフトウェアに関する記述は、本文に記載してあるバージョンをもとにしています。ソフトウェアはバージョンアップされる場合があります、本書での説明とは機能内容や画面図などが異なってしまうこともあります。本書ご購入の前に、必ずバージョン番号をご確認ください。

以上の注意事項をご承諾いただいたうえで、本書をご利用願います。注意事項をお読みいただくずにお問い合わせいただいても、技術評論社および著者は対処しかねます。あらかじめ、ご承知おきください。

本書に登場する製品名などは、一般に各社の商標または登録商標です。なお、本書中に™、©、®などのマークは記載しておりません。



はじめに



MySQLは長年にわたり、信頼性の高いデータベースとして多くのエンジニアに支持されています。そのシンプルさ、性能、そしてオープンソースであることの魅力により、数々のプロジェクトに採用されてきました。筆者自身も数多くのMySQLサーバーを構築、運用してきたエンジニアの一人です。Webアプリケーションから基幹業務システムまで、さまざまな用途でMySQLの安定性を実感する一方で、データ量とアクセス数の増加に伴う運用課題も痛感してきました。MySQLの運用をもっと楽しみたい、性能を柔軟に増減させたい、障害対応を簡素化したいという思いを筆者なりに日々感じていました。

そんな中出会ったのがTiDBでした。筆者がTiDBの存在を知ったのは、TiDBのストレージ層で使用されているTiKVがCloud Native Computing Foundationによって、十分に成熟したプロジェクトであることを表すGraduated Projectとして、2020年に認定されたことがきっかけでした。もともとTiKV自体に技術的な興味を持っていたこともあり、このTiKVをストレージとして活用し、MySQL互換のインターフェースでデータにアクセスできるTiDBというソフトウェアが存在するのを知りました。そして、その革新的な設計思想と内部アーキテクチャーについて調べ始めたことで、TiDBへの興味がどんどん膨らんでいきました。当時、分散システムとリレーショナルデータベースの良さを兼ね備えた非常に興味深いアーキテクチャーだと感じたことは、今でも鮮明に記憶しています。

TiDBを実際に触ってから今日まで、MySQLでの課題が解消していく喜びと、分散データベース特有のノウハウを学んでいく必要性を感じています。コミュニティ活動にも積極的に参加するようになり、イベント登壇や勉強会などでTiDBの魅力を伝える活動を始めました。振り返ってみると単なる技術的興味から始まり、知見の共有へと発展してきたTiDBとの関わりは、筆者自身のエンジニアとしての成長にも大きく貢献してくれたと思っています。このようなコミュニティ活動や情報発信を続ける中で、ご縁があってこのようなTiDBの書籍を執筆する機会をいただき、皆さんに本書をお届けするに至りました。

本書を通じて、TiDBの技術的な素晴らしさだけでなく、実際のデータベース運用における課題解決の可能性や将来性も含めた魅力をたくさんの人に伝えたいと思っています。そして読者の皆さんがTiDBを知ることで、データベース運用の新たな選択肢を得ていただければ幸いです。

TiDBの世界に飛び込む準備はできましたか？ 本書を読み終わった時にはきっとTiDBの魅力を感じていただけるはずです。ぜひTiDBを実際に触ってみて、その魅力を体感してください。従来のデータベースとは異なる新しい可能性が見えてくることでしょう。

本書から学べること

本書はTiDBの基本的なアーキテクチャーをはじめ、さまざまな環境へのTiDBのインストール手順について解説します。インストールが終われば、次に必要なことはTiDBに対する各種操作方法を理解することです。各環境におけるTiDBに対する操作方法をインストール手順の後に詳細に解説していきます。また、マネージドサービスであるTiDB Cloudを利用する際の手順も解説しています。

加えて、データベースの運用には欠かせないデータのバックアップやリストア、移行を効率的に行うための方法についても解説し、TiDBを運用していく上で役に立つさまざまなテクニックも紹介しています。最後に付録として、よく使うコマンド操作を逆引きリファレンスとして紹介していますので、ぜひ活用してください。

TiDBは現代の多様なインフラニーズに対応するデータベースです。この本を通じてTiDBを使用したモダンなデータベース運用の実践方法を理解し、自身のシステムに応用、実践できる知識を身につけていただけることを期待しています。

- **第1章のポイント**

TiDBが生まれた背景にあるデータベースの歴史に触れながら、TiDBの特徴と利用形態を紹介します

- **第2章のポイント**

分散システムとして設計されているTiDBのアーキテクチャーについて解説します。構成を理解することで、拡張性、可用性、パフォーマンスの仕組みを知るための基礎とします

- **第3章のポイント**

物理サーバーまたは仮想サーバーにTiDBを構築したいならここを読んでください。強力なコマンドラインツールであるTiUPを詳しく解説します

- **第4章のポイント**

Kubernetes環境にTiDBを構築したいならここを読んでください。Kubernetesで構築・運用・管理するためのツールTiDB Operatorを詳しく解説します

- **第5章のポイント**

運用管理の負担を軽くしたいならここを読んでください。TiDBのマネージドサービスであるTiDB Cloudを詳しく解説します

- **第6章のポイント**

TiDB稼働後に役立つデータ運用ツールを紹介します。データのバックアップとリストアは重要ですし、データの移動や同期によって既存データベースの移行や共存を助けます

- **第7章のポイント**

TiDBの機能を最大限に活用するためのSQLや設定などのノウハウを紹介します

本書を読むべき人

本書はMySQLをはじめとする従来のRDBMSの運用面や性能面の限界に直面しているエンジニアや、スケーラブルで高可用なデータベースソリューションを求めているシステム管理者、開発者に最適です。また、クラウドネイティブな環境でのデータベース運用に興味がある方や、最新のデータベース技術を学びたいと考えている技術者にも役立つ内容となっています。

この本の表記

TiDB

本書内でTiDBと表記している場合はソフトウェアとしてのTiDB、およびTiDBを構成しているコンポーネントすべてを含んだクラスター全体のことを示しています。

TiDBクラスター

TiDB クラスターとはTiDBのドキュメントのTiDB Architecture^{注1}に従い、アーキテクチャ図内のTiDBを構成するコンポーネントの一部であるTiDB Clusterのことを示します。

データベースの役割としてのソース

従来のデータベースの役割としてマスターという表記を使用する代わりに本書ではソース、またはソースサーバーという表記を使用します。これはMySQL 8.0.21のリリースノート^{注2}内で表記の変更が言及されましたので、それに準拠するようにしたためです。

コマンドラインツールの呼び名

本書内ではさまざまなコマンドの使い方、実行結果を記載しています。その際、コマンドラインツール(コマンドを入力するツール)の一般名称を「端末」と表記しています。端末は、たとえばmacOSのターミナル、Windowsのコマンドプロンプト、Linuxのターミナルなどのことを指します。

注1 <https://docs.pingcap.com/tidb/stable/tidb-architecture/>

注2 <https://dev.mysql.com/doc/relnotes/mysql/8.0/en/news-8-0-21.html>

端末の出力結果の省略

本書内では、端末に表示されるコマンドラインの出力結果やメッセージを一部省略して記載している場合があります。省略された部分は「...」や「(省略)」と表記されていますのでご了承ください。

折り返し記号

本書内では、本来は1行で書く必要があるコマンドやコードであってもページ幅の都合で折り返して記載しています。行末に🔗の記号がある箇所は折り返されていることを表しています。改行ではないことにご注意ください。

単位

本書内では、データ量の単位としてKiB(キビバイト)、MiB(メビバイト)、GiB(ギビバイト)を使用しています。これは、TiDBのドキュメント内で使用されている単位に準拠するためです。KiBは 2^{10} (1024)バイト、MiBは 2^{20} (1048576)バイト、GiBは 2^{30} (1073741824)バイトを意味しています。



本書での実行環境



本書ではLinux環境での動作環境としてRocky Linux 9を使用しています。Kubernetes環境はDockerを使用したkindで構築したものを使用し、Kubernetesのバージョンは1.31.4を使用しています。本書内でも案内していますが、kindを使用する場合はDockerまたはPodmanを別途インストールする必要があります。また、本書は2025年2月時点での情報をもとに記述しています。



本書のサンプルコードについて



本書のサンプルコードは、下記のGitHubで公開しています。

<https://github.com/makocchi-git/tidb-practical-book-sample>

本書の正誤表や追加情報について

本書の正誤表や追加情報は、下記の本書サポートページをご参照ください。



<https://gihyo.jp/book/2025/978-4-297-14832-4>

謝辞

本書を執筆するにあたり、監修として全体的な文章のレビューおよび技術的なサポートにご協力いただいた本多様(PingCAP 株式会社)に感謝の意を表させていただきます。また、普段活動しているTiDBコミュニティの運営メンバーの皆様、そしてTiDBの開発の関わる皆様にも感謝の意を表します。最後に、このようなTiDBの書籍を執筆する機会をくださった菊池さん(株式会社技術評論社)に改めて感謝の意を表します。

はじめに	iii

第 1 章

TiDBが必要とされるデータベースのいま

データベースの変遷とTiDBの特徴	1

I-1 TiDBとは	2

column OSS Insightを支えるTiDB	2

TiDBのリリースおよびバージョン表記	2

TiDBの歴史	3

column Spannerを参考に生まれてきた他のソフトウェア	4

PingCAPについて	4

I-2 NewSQLとは	4

RDBMSの特徴	5

column シャーディングとは	5

NoSQLの特徴	5

なぜNewSQLが必要とされているのか	6

NewSQLの特徴	6

column ACID特性とは	8

Distributed SQL (分散SQL)	8

I-3 TiDBの特徴	8

コマンドラインツールTiUPによるオペレーションが強力	9

書き込み用のエンドポイントの冗長化が容易	9

シャーディングが不要	10

バージョンアップが容易	10

効率の良いスケールが可能	10

データ反映の遅延を気にしなくて良い	12

タイムトラベルクエリが可能	12

column MVCCとは	13

モニタリング環境および組み込みダッシュボードが提供されている	13

レイテンシーの増加傾向を抑える機能が提供されている	14

HTAPをサポートしている	15

リクエストユニットによるリソース制御が可能	15
TTLによるデータの自動削除が可能	16
MySQLと高い互換性がある	16
1-4 TiDB の提供形態	17
TiDB Cloud	17
TiDB Self-Managed	17
TiDB Community	17
column エンタープライズ版とオープンソース版の機能の違い	17
利用目的での比較	18
1-5 まとめ	18

第 2 章

TiDB を構成するコンポーネント

可用性と拡張性が高い分散システム

19

2-1 TiDB の構成	20
2-2 TiDB クラスター	21
TiKVにアクセスする際のSQLの変換	21
TiFlash への振り分け	25
column EXPLAIN 時に表示される task の意味	27
2-3 PD クラスター	27
データの配置管理	27
TSO の発行	27
column TiDB における分散トランザクションを支える技術	28
TiDB クラスターからの TSO 発行リクエストの並列化	28
TSO を時刻へ変換	29
時刻から TSO を生成	30
PD クラスターの死活監視	31
column スプリットブレインとは	32
ストレージクラスターの死活監視	32
PD クラスターのパフォーマンス	33
PD マイクロサービスモード	33

Active PD Follower 機能	33
2-4 ストレージクラスター	34
TiKV	34
column CNCF について	35
リージョンと RocksDB	35
RocksDB の Titan プラグイン	36
TiFlash	37
column TiFlash で採用されている技術	38
TiFlash を有効にした場合はディスク容量に注意	38
プッシュダウンとコプロセッサ	38
TiKV のコプロセッサ	39
2-5 TiProxy コンポーネント	40
TiProxy の仕組み	41
効果的に TiProxy を使用するために必要なこと	41
graceful-wait-before-shutdown の設定	41
systemd の TimeoutStopSec の設定	42
TiUP 操作時の待機時間の設定	42
Kubernetes 環境における terminationGracePeriodSeconds の設定	42
自己証明書の作成	42
TiDB 全体のバージョンアップ時に TiProxy を除外しておく	42
TiProxy のパフォーマンス	43
TiProxy の制約事項	43
2-6 まとめ	43

第 3 章

仮想／物理サーバーへの TiDB の構築

TiUP を使った構築と操作	45

3-1 TiUP の基本	46
TiUP をインストールする	46
column インストール手順で使用したコマンドの説明	48
シェルの補完機能を使用する	48
TiUP をバージョンアップする	49

使用するTiUPのバージョンを管理する	49
TiUPのデータ保存先を変更する	51
TiUPが使用する環境変数を確認する	52
3-2 TiDB Playgroundの使い方	52
TiDB Playgroundを起動する	52
起動するTiDBバージョンを指定する	55
タグを指定してTiDB Playgroundを起動する	56
タグを利用したデータの永続化	58
タグを使用しても前回起動時に指定した構成情報は引き継がれない	60
column 自動生成されるタグはどのように生成されているのか	60
特定の設定を読み込ませてTiDB Playgroundを起動する	60
TiDB Playgroundで起動するコンポーネントを調整する	62
モニタリングコンポーネントの起動の抑制	62
各種コンポーネントのスケールアウトとスケールイン	63
各種コンポーネントの起動数の指定	64
TiProxyを使用した接続	65
TiDB Playgroundのデータをクリーンアップする	68
TiKVのPlaygroundを起動する	69
3-3 TiUPを使用した仮想サーバーまたは物理サーバーへのTiDBの構築	70
サーバー環境を準備する	70
サーバー環境を元にトポロジーファイルを作成する	70
TiUP実行時のsshログイン方法を指定する	73
トポロジーファイルを元に構成のチェックをする	73
トポロジーファイルを元にTiDBを構築する	74
データベースのrootパスワードを紛失した場合	77
TiDB構築時における特定の設定の反映	79
現在のトポロジー状態をファイルに保存する	79
TiUPの同時実行数を変更する	80
3-4 各種コンポーネントのスケールアウトとスケールイン	80
各種コンポーネントをスケールアウトする	80
各種コンポーネントをスケールインする	83
3-5 TiDB Dashboardの構築	84
TiDB Dashboardに接続する	84

TiDB Dashboardを無効化する	85
稼働しているTiDBおよびTiDB Playground	85
TiDB構築時およびTiDB Playground起動時	86
TiDB Dashboardを担当しているPDサーバーを切り替える	86
3-6 モニタリング環境の構築	87
PrometheusおよびGrafanaのバージョンを確認する	89
TiDB Playgroundでモニタリング環境の起動を抑制する	89
3-7 TiDBのライフサイクル操作	90
TiDBを停止する	90
TiDBを再起動する	91
TiDBを削除する	91
削除時のデータやログの保持	92
OS起動時のプロセス自動起動の有効／無効を切り替える	92
コンポーネントやノードを指定して操作する	93
操作の履歴および操作内容の詳細を確認する	94
操作履歴の内容を再現する	95
操作履歴を削除する	95
3-8 TiDBのバージョンアップ	96
バージョンアップ前に確認するべきこと	96
オンラインでバージョンアップする	96
オフラインでバージョンアップする	98
3-9 クラスターメタ情報のバックアップ	99
3-10 デモ用データセットの作成	100
3-11 まとめ	104

第 4 章

Kubernetes 環境へのTiDBの構築

TiDB Operatorを使った構築と操作

105

4-1 Kubernetes環境の準備	106
kubectiをインストールする	107

kindを使用してKubernetes環境を準備する	107
minikubeを使用してKubernetes環境を準備する	110
4-2 TiDB Operatorのインストール	112
カスタムリソース定義をインストールする	112
column TiDBクラスターオートスケーラーについて	114
Helmを使用してTiDB Operatorをインストールする	114
TiDB Operatorのインストール時に設定可能なオプション	116
4-3 TiDB Operatorを使用したTiDBの構築	117
マニフェストファイルを作成して適用する	117
構築したTiDBへ接続する	119
Kubernetesクラスター内からの接続	119
Kubernetesクラスター外からの接続	121
ポートフォワーディングを使用した接続	124
各種コンポーネントをスケールアウト、スケールインする	125
各種コンポーネントをスケールアップ、スケールダウンする	128
TiProxyを構築する	130
TiFlashを構築する	132
TiDB Dashboardを構築する	133
モニタリング環境を構築する	135
NgMonitoringを有効化する	138
バージョンアップする	139
TiDBのバージョンアップ	140
TiDB Operatorのバージョンアップ	141
4-4 ログの確認	142
コンポーネントのログを確認する	142
スローログを確認する	144
4-5 初期パスワードとイニシャライズ	145
4-6 TiDBの削除	146
4-7 TiDB Operatorのアンインストール	147
4-8 まとめ	148

第 5 章

TiDB Cloud の利用

TiDB のマネージドサービスを利用して運用負荷を軽減

149

5-1 TiDB Cloud Serverless	150
高可用性を選択する	151
column パブリッククラウドのリージョンとアベイラビリティゾーン	152
TiDB Cloud Serverless Branching を使用して データやスキーマを別々に管理する	152
ベクトル検索機能を使用する	152
TiDB Cloud Serverless のアーキテクチャーを理解する	152
5-2 TiDB Cloud Dedicated	153
ビルトインアラート機能	154
ノードグループ機能	154
5-3 TiDB Cloud の始め方	155
サインインして初期クラスターを確認する	155
組織とプロジェクトを確認する	158
タイムゾーンの設定	158
ユーザーとロールの設定	159
請求情報の設定	160
アカウントを設定する	161
クラスター概要画面を表示する	162
SQL Editor を使用する	163
ブランチを使用してデータとスキーマを別々に管理する	168
column コピーオンライトとは	168
データをバックアップおよび復元する	170
TiDB Cloud Serverless の場合	170
TiDB Cloud Dedicated の場合	171
ビルトインアラート機能を使用する	171
サードパーティメトリクスと連携する	173
クラスターを削除する	174
5-4 TiDB Cloud の稼働状況の確認	175

5-5 TiDB Cloud CLI の使い方	175
インストールする	175
インストールスクリプトを使用してインストール	176
TiUPを使用してインストール	176
プロファイルを作成する	177
プロファイルを直接編集する	178
OAuthによる認証を使用する	178
APIキーによる認証を使用する	180
プロファイルを切り替える	182
プロファイルを更新する	182
プロファイルを削除する	183
TiDB Cloud Serverlessのクラスターを操作する	183
作成	183
削除	184
クラスターへの接続	185
ブランチの操作	185
SQLユーザーの操作	187
月額限度額の設定	188
GitHub ActionsでTiDB Cloud CLIを使用する	189
5-6 まとめ	190

第 6 章

TiDB のデータ運用ツール

バックアップとリストア 同期、移行	191

6-1 バックアップとリストア	192
BRを使用する	193
BR 実行時のパフォーマンスへの影響	193
BR によるバックアップの仕組み	193
BR のインストール	194
スナップショットバックアップ	194
column SST ファイル名について	197
ログバックアップ	197

スナップショットからのリストア	200
ポイントインタイムリストア	202
Kubernetes環境でBRを使用して	
バックアップとリストアをする	203
Backup リソースの定義	205
BackupSchedule リソースの定義	210
Restore リソースの定義	211
Dumplingを使用してデータをエクスポートする	213
Dumpling のインストール	213
TiDB または MySQL からのエクスポート	213
外部ストレージにエクスポートする	218
TiDB Lightningを使用してデータをインポートする	219
TiDB Lightning のインストール	219
IMPORT INTO 構文と TiDB Lightning	220
物理インポートモード	221
論理インポートモード	222
設定ファイルを使用した実行	222
Web インターフェース経由での実行	223
6-2 TiCDC による TiDB 外へのデータ同期	225
TiCDC クライアントをインストールする	225
TiDB Playground で TiCDC を起動する	226
TiUP を使用して TiCDC を構築する	230
外部ストレージへデータを同期する	231
Apache Kafka、Apache Pulsar へデータを同期する	232
設定ファイルを使用してチェンジフィードを作成する	232
双方向レプリケーションを有効にして	
複数台アクティブな TiDB を構築する	232
BDR モードを有効にしたチェンジフィードの作成	233
BDR ロールの操作	234
column DDL と DML	237
データ同期の確認	237
3 台環境での BDR モード	238
TiDB Cloud でチェンジフィードを作成する	239
Kubernetes 環境へ TiCDC をデプロイする	241
6-3 DM によるデータ移行	243
dmctl をインストールする	243

MySQLを起動して移行用データを用意する	244
TiDB PlaygroundでDMクラスターを作成する	245
DMクラスターにデータソースを追加する	247
TiDB PlaygroundのDMクラスターで パスワードの暗号化を使用する場合	249
DMクラスターに移行タスクを作成する	250
データの移行を確認する	251
Webコンソールを使用してDMクラスターを操作する	252
TiDB Operatorを使用したDMクラスターの構築	253
6-4 sync-diff-inspector によるデータ同期の差分確認	256
column sync-diff-inspectorとsync_diff_inspector	256
sync_diff_inspector コマンドをインストールする	256
sync_diff_inspector コマンドを使用して データの差分を確認する	257
6-5 まとめ	261

第 7 章

TiDB を便利に使うためのテクニック

リソース制御、バクトル検索、監視、パフォーマンス測定	263

7-1 SQL によるバックアップとリストア	264
BACKUP	264
RESTORE	266
LOAD DATA	267
IMPORT INTO	268
7-2 ガベージコレクションの設定	270
保持期間の変更	271
TiKV MVCC In-Memory Engine	272
ガベージコレクションの実行間隔の設定	273
ガベージコレクションの停止	274
7-3 リソース制御によるマルチテナント環境の構築	274

リソースグループの確認	275
リソースグループの管理	276
ランナウェイクエリの制御	277
バックグラウンドタスクの制御	279
column reorg DDL とは	279
リソースグループのバインド	280
ユーザーレベルで行う場合	280
セッションレベルで行う場合	281
クエリレベルで行う場合	282
リクエストユニットの計測	282
リソースグループ制御の無効化	283
7-4 メタデータロック	283
7-5 TTL を使用したデータの自動削除	286
TTL の設定	286
TTL のバックグラウンドジョブの設定	287
7-6 ホットスポットの回避	288
TiDB の AUTO_INCREMENT	289
次に採番される値の確認	291
再起動すると払い出される範囲が変更される	291
MySQL と同じ挙動の AUTO_INCREMENT	292
AUTO_RANDOM の使用	293
AUTO_RANDOM の値について	294
非クラスター化テーブルにおける SHARD_ROW_ID_BITS の使用	296
7-7 ベクトル検索機能	298
ベクトル検索が注目されている背景	298
TiDB におけるベクトル検索	298
7-8 Terraform による TiDB Cloud の管理	301
TiDB Cloud の API キーの取得	301
Terraform の定義	302
7-9 Advanced StatefulSet を使用した TiDB の構築	305
Advanced StatefulSet の有効化	306
Advanced StatefulSet の使用	306

インデックスの操作	307
7-10 PrometheusやGrafanaのカスタマイズ	308
Prometheusのカスタマイズ	308
Grafanaのカスタマイズ	310
Alertmanagerのカスタマイズ	312
7-II TiDB Dashboardの活用	313
TiDB Dashboardへのログインを制御する	313
SSOを使用したログイン	314
TiDB Dashboardのページ詳細	316
Overview	317
Cluster Info	317
Top SQL	317
SQL Statements	318
Slow Queries	320
Key Visualizer	321
Cluster Diagnostic	322
Monitoring	322
Search Logs	323
Resource Manager	323
Advanced Debugging	323
7-12 TiUPによるデータベースパフォーマンス測定	324
TPC-C	324
TPC-H	326
CH	329
YCSB	330
TiDBクラスターに対するベンチマーク	330
ストレージクラスターに対するベンチマーク	332
RawSQL	333
より詳細な使い方	333
7-13 TiDB AI Assistantによるナレッジの検索	334
7-14 まとめ	334

付 録

逆引きコマンドリファレンス

335

TiUP	336
コンポーネントの操作	336
シェル補完の操作	338
操作履歴の操作	338
TiDB の操作	339
起動と停止	340
スケールアウトとスケールイン	341
削除	341
バージョンアップ	342
情報の表示	342
監査ログの操作	342
メタデータのバックアップと復元	343
TiDB Playground の操作	343
デモ用データの操作	344
ベンチマークの実行	344
クライアントツールの実行	346
TiDB Cloud CLI	346
シェル補完の操作	347
プロファイル操作	347
OAuth 認証	348
TiDB Cloud Serverless の操作	348
ブランチの操作	349
SQL ユーザーの操作	350
月額限度額の操作	351
データインポートの操作	351
データエクスポートの操作	352
プロジェクトの確認	353
TiDB Bot に問い合わせる	353
おわりに	354
索引/逆引きコマンドリファレンス 索引	355 / 359
著者/監修者プロフィール	363

第

1

章

TiDBが必要とされる データベースのいま

データベースの変遷とTiDBの特徴

本章ではTiDBについて概要を説明し、これから本書を通じて学ぶさまざまな知識の基礎となる部分を学習します。TiDBの生まれた歴史的な背景に触れながら、TiDBを語る上で外すことはできないNewSQLについても説明します。また、TiDBはさまざまな利用形態が提供されており、どのようなシーンでどの利用形態が適しているかについても説明します。



TiDBとは

TiDBは日本語で“タイデービー”と発音します。TiDBの「Ti」はチタン(Titan)の元素記号“Ti”を意味しており、その名の通りチタンのように堅牢なデータベースという意味が込められています。後の章で詳しく説明しますが、TiDBは分散システムで構成されており単一障害点(Single Point Of Failure)が無いシステムです。そのため、耐障害性に優れた堅牢なソフトウェアと言えるでしょう。

TiDBはオープンソースのソフトウェアであり、全世界のコントリビューターによって開発されています。ライセンスはApache License 2.0^{注1}です。執筆時点でのTiDBのGitHubリポジトリ^{注2}のステータスによるとStar数は37,000を超え、コントリビューター数は約880人に達しています。OSS Insight^{注3}で公開されているオープンソースのデータベースランキング^{注4}によると、2018年から2024年までElasticsearchに次いで2位に位置しており、人気の高さがうかがえます。

OSS Insightを支えるTiDB

column

OSS InsightはGitHubのデータを定期的に取り込み、どのリポジトリが最も人気があるか、どのリポジトリが最も多くのPull RequestsやIssuesが開かれているかなどを表示できます。執筆時点ではOSS Insightには80億以上のGitHubイベントが蓄積されており、この膨大なデータに対してさまざまなレポートを表示するために分析クエリを発行できます。また、自然言語で質問をしてSQLクエリを自動生成する機能もあります。OSS InsightのデータベースにはTiDB Cloud Serverlessが採用されています。

TiDBのリリースおよびバージョン表記

TiDBは2つのリリース形態を取っており、長期サポートリリースのLTS(Long Term Support)と開発マイルストーンリリースのDMR(Development Milestone Release)の形態があります。LTSは新機能が追加され、ライフサイクルではパッチリリースとして不具合修正とセキュリティ脆弱性^{ぜいじやく}修正が提供さ

注1 <https://www.apache.org/licenses/LICENSE-2.0>

注2 <https://github.com/pingcap/tidb>

注3 <https://ossinsight.io/>

注4 <https://ossinsight.io/collections/open-source-database/>

れます。パッチリリースでは新機能は提供されません。DMRも同様に新機能が追加されますが、パッチリリースが提供されず不具合は後続のDMRまたはLTSで修正されます。また、DMRはPingCAPによるサポートも対象外になります。LTSにおけるサポート期間についてはドキュメントのTiDB Release Support Policy^{注5}を参照してください。

TiDBのバージョンの表記は「x.y.z」となります。x.yはリリースシリーズを表し、zはリリース初期では0です(例: 8.5.0)。zはパッチリリースと共に増加していきます(例: 8.5.1)。DMRのバージョンは「-DMR」が付与されます(例: 8.4.0-DMR)。

TiDBの歴史

製品が十分に成熟し、一般的に公開される段階をGA(General Availability)と呼びます。TiDBが生まれた日を最初にGAとしてリリースされた日と定義するならば、2017年10月16日が記念すべき第一歩を踏み出した日ということになります。TiDBのリリースノート^{注6}によれば、最初のGAリリースとして1.0.0が2017年10月16日にリリースされています。

TiDBはなぜ生まれてきたのでしょうか。PingCAP創業者の一人であるEd Huang氏によると、当時MySQLの構築、運用に関わっていた際に会社の成長に伴いMySQLの環境もシャーディングでスケールさせる必要があり、サービスを止めずにスケールさせていくことがとても大変であったと語っています^{注7}。その際に氏が出会ったのが、Googleが公開したNewSQLと呼ばれる分散データベースのSpannerの論文でした。従来のMySQLのようなRDBMS(Relational Database Management System)とNoSQL(Not Only SQL)の良い部分を取り入れたNewSQLこそが未来のデータベースであると確信した氏は、Google Cloudでしか使うことのできないSpannerと同様なソフトウェアをオープンソースとして作ろうと考え、TiDBを生み出しました。

2017年に最初のGAリリースをして以来、TiDBはさまざまな機能を追加しながらリリースを重ね、2022年にはマネージドサービスであるTiDB CloudのサービスがGAとしてリリースされました。TiDB CloudはTiDBに関わる運用部分をすべてマネージド側で担保してくれることにより、大幅な運用コストの低減を見込めます。TiDB Cloudについては本書の第5章で詳しく記載します。

注5 <https://www.pingcap.com/tidb-release-support-policy/>

注6 <https://docs.pingcap.com/tidb/stable/release-notes>

注7 <https://pingcap.co.jp/press-release/tidb-anticipated-as-no-1-database-2022>

Spannerを参考に生まれてきた他のソフトウェア

column

Spannerを参考に生まれてきたソフトウェアはTiDB以外にもいくつか存在します。代表的なソフトウェアをいくつか挙げるとYugabyteDB^{注a}やCockroachDB^{注b}になります。TiDBはMySQLプロトコルとの互換性を持っていますが、YugabyteDBとCockroachDBはPostgreSQLプロトコルとの互換性を持っており、それぞれ特徴が異なります。

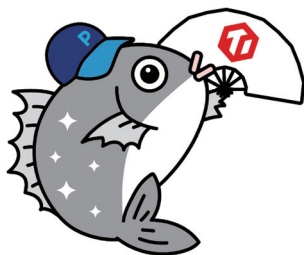
注a <https://www.yugabyte.com/jp/yugabytedb/>

注b <https://www.cockroachlabs.com/product/overview/>

PingCAPについて

PingCAPは2015年に創業された企業です。TiDBの主要な開発企業であり、TiDBのマネージドサービスであるTiDB Cloudというサービスを提供している企業です。PingCAPの社名は、ネットワークの疎通を確認するために使用されるコマンド「Ping」とCAP定理の「CAP」の2つの単語を組み合わせて生まれています。3つのうち2つを選ばなければならないとされるCAP定理の一貫性(Consistency)、可用性(Availability)、ネットワーク分断への耐性(Partition Tolerance)ですが、この3つのすべてに接続したい(Ping)という思いが込められています。PingCAPの日本法人は2021年に設立され、日本法人独自の公式キャラクターピンちゃんも存在します(画像1.1)。

画像1.1 ピンちゃん



1-2

NewSQLとは

TiDBを理解するためには、まずNewSQLという概念を押さえておくことが重要です。NewSQLとは一言で言えば、従来型のRDBMSとNoSQLの良いところを組み合わせた新しいデータベースのアーキテクチャーです。この概念を深く理解するためには、まずRDBMSとNoSQLの特性をそれぞれ理解することが重要です。その上でNewSQLがなぜ必要とされるようになったのかについても説明します。

RDBMSの特徴

長い間システムの根幹を支えてきたデータベースであるRDBMSは、その歴史的な背景から設計思想が単一のサーバー上で稼働することを前提としていることが多く、スケールアウトが苦手であるという特徴を持っています。この特徴により、システムの性能向上が必要な場合にはサーバーのハードウェア性能を向上させるスケールアップが一般的な対応策として取られます。しかし、スケールアップにはコストがかかるだけでなく性能向上の限界があるため、その限界を超えるニーズに対応するための追加手段を検討する必要があります。

他の解決策として、データを分割して複数のサーバーに分散するシャーディングという手法があります。この方法は大規模システムにおいてスケーラビリティを提供しますが、適切に運用するためにはデータ分割のルール設計や負荷分散の調整が求められ、運用の手間が増えることがあります。また、RDBMSではソースサーバーの冗長化を実現する際に他の仕組みやツールを組み合わせる必要がある場合が多く、システム全体の設計と管理において工夫が求められます。

RDBMSはトランザクション処理やデータ整合性を強く保証する特徴を持っている一方で、大量の同時書き込み要求が発生する環境ではロックの発生や待機時間の増加が発生することがあり、性能の低下につながることも多くあります。

RDBMSの代表的なソフトウェアとしてはMySQL^{注8}、PostgreSQL^{注9}、Microsoft SQL Server^{注10}が挙げられます。

シャーディングとは

column

シャーディングとは大量のデータを効率的に管理するための方法で、データをシャード(データセット)として分割して複数のサーバーやデータベースに分散させる仕組みです。シャーディングを導入することで1つのサーバーに負担が集中するのを防ぎ、全体のスケーラビリティを向上させることができますが、運用上の負荷が増加するという大きな課題もあります。データが複数のサーバーに分散されるため、新しいデータの追加や更新時にどのサーバーに保存すべきかを決定するロジックが必要になります。また、シャード間でデータの整合性を保つ仕組みや、障害発生時の復旧計画もより高度なものが求められます。

データを分割した結果、特定のサーバーに処理が偏るとそのサーバーだけが過負荷になる問題(ホットスポット)が発生する可能性があります。ホットスポットを防ぐために、適切な分割基準を設定して定期的にシャードの再配置や調整をする必要があります。そのため運用の手間が増えることを考慮しなければなりません。

NoSQLの特徴

NoSQLはRDBMSが抱える課題に対応するために登場したデータベースです。その最大の特徴は、高

注8 <https://www.mysql.com/>

注9 <https://www.postgresql.org/>

注10 <https://learn.microsoft.com/ja-jp/sql/sql-server/what-is-sql-server>

いスケーラビリティを持ち、水平にスケールアウトすることが容易である点です。このため、大量のデータや高トラフィックを処理する必要がある特定のユースケースで優れた性能を発揮します。また、NoSQLはスキーマレスなデータモデルを採用しており、柔軟なデータ構造を提供します。データ形式としてはキーバリュー形式やドキュメント形式などがあり、これにより異なる種類のデータを効率的に扱うことが可能です。

NoSQLは一貫性やトランザクション管理をある程度犠牲にすることで、可用性や性能を優先する設計を採用していることが多いのが特徴です。このため、特に複雑なデータ処理やビジネスロジックを必要とするアプリケーションでは、開発者がデータのー貫性を確保する仕組みをアプリケーション側で実装しなければならない場合があります。これらの特性は、NoSQLが特定のユースケースで優れた性能を発揮する一方で、利用場面によっては慎重な選択と設計が求められる要因となっています。

NoSQLの代表的なソフトウェアとしては、Apache Cassandra^{注11}やMongoDB^{注12}が挙げられます。これらはそれぞれ異なる特性を持ち、多様なユースケースに対応しています。

なぜNewSQLが必要とされているのか

インターネットやクラウド技術の普及により、ECサイトやオンラインゲームのような大規模データ処理を必要とするアプリケーションが増加し、それらのバックエンドを支えるデータベースに対する要求も急速に高まっています。従来のRDBMSはトランザクション処理やデータ整合性の保証に優れている一方で、スケーラビリティの面で課題を抱えており、このような環境ではその制約が顕著になっています。一方でNoSQLは高いスケーラビリティを持っていますが、その代償として一貫性やトランザクション管理を犠牲にしており、適用が難しい場合もあります。

NewSQLはまさにこれらの課題を解決するために登場したデータベースであると言えるでしょう。一貫性やトランザクション管理を犠牲にすることなく、高いスケーラビリティを実現する設計思想を持つNewSQLは、まさに現代のアプリケーションを支える最適なデータベースです。

NewSQLの特徴

NewSQLは従来のRDBMSとNoSQLの両方の利点を持つデータベースです(表1.1)。その最大の特徴は高いスケーラビリティとデータのー貫性を同時に実現する点にあります。従来のRDBMSが苦手とする水平スケールアウトを可能としながら、分散環境においてもACID特性を保持できるためトランザクション管理やデータ整合性を犠牲にする必要がありません。

注11 <https://cassandra.apache.org/>

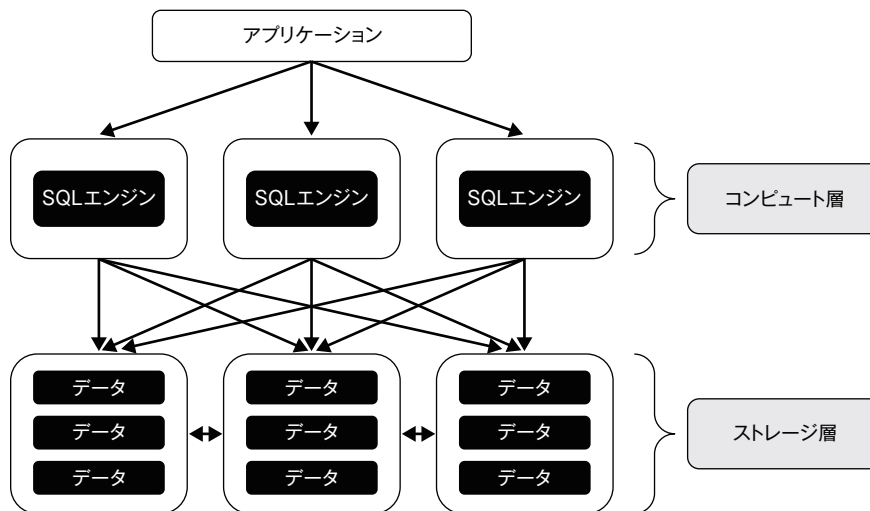
注12 <https://www.mongodb.com/>

表1.1 データベースの特徴比較

特徴	RDBMS	NoSQL	NewSQL
SQLインターフェース	有り	対応していない場合が多い	有り
トランザクション処理	有り	堅牢なトランザクションはサポートしていない場合が多い	RDBMSと同等
スケーラビリティ(読み取り性能)	リードレプリカで対応可能	高い	高い
スケーラビリティ(書き込み性能)	シャーディングが必要	高い	高い

さらにNewSQLはSQLインターフェースを採用しており、既存のRDBMSと同様の操作が可能です。これにより開発者や運用担当者が新しい技術を学ぶ負担を最小限に抑えつつNewSQLを採用できます。また、分散トランザクション処理やデータの自動シャーディングが内部で効率的に行われるため、開発者は複雑な運用設計をする必要が無くなり開発に集中できます。

NewSQLのアーキテクチャーは一般的にはSQLを受け付けるコンピュート層とデータを格納するストレージ層に分かれており、それぞれが独立してスケールアウトできる構造になっています(図1.1)。TiDBのアーキテクチャーもコンピュート層とストレージ層が分かれています。TiDBを構成するコンポーネントの詳細については第2章で詳しく説明します。

図1.1 NewSQLのアーキテクチャー

ACID特性とは

column

ACID特性とはトランザクション処理における重要な4つの特性を指します。それぞれの頭文字を取ってACIDと呼ばれています。

- **Atomicity(原子性)**

すべて成功するかすべて失敗するかのどちらかであることを保証します。途中でエラーが発生した場合、トランザクション内の全操作は元の状態にロールバックされます

- **Consistency(一貫性)**

トランザクション実行前後でデータの整合性が保たれることを保証します

- **Isolation(独立性)**

複数のトランザクションが同時に実行された場合でも、トランザクション間で干渉が発生しないことを保証します

- **Durability(永続性)**

トランザクションが成功してコミットされた場合、その変更が永続的に保存されることを保証します

▶ Distributed SQL (分散SQL)

近年はNewSQLという言葉ではなく、Distributed SQLという言葉が使われることが増えてきています。Distributed SQLの特徴は、複数のサーバー間でデータを複製する単一のリレーショナルデータベースであり、データセンター間や地理的に分散した環境においても一貫性を保つことができる点とされています。これは本節で説明したNewSQLの特徴と一致しています。

ではTiDBはNewSQLなのでしょうか。それともDistributed SQLなのでしょうか。執筆者なりの視点で言えば、TiDBはNewSQLでもありDistributed SQLでもある、と考えます。NewSQLとDistributed SQLの違いを厳密に区別することは難しいため、本書ではTiDBをNewSQLとして説明しますが、Distributed SQLとしても認識していただいて問題ありません。

I-3

TiDBの特徴

TiDBはMySQLをフォークして作成されたわけではなく、独自の実装でMySQLとのプロトコル互換性を実現しています。そのため、その内部では全く異なるアーキテクチャーが採用されており、MySQLとは少し違う特徴を持っています。本節ではいくつかMySQLとは違う特徴を挙げ、従来のデータベースの運用で課題になりがちな部分をTiDBはどのように解決しているのかを説明します。

■ コマンドラインツールTiUPによるオペレーションが強力

まず最初に紹介したい特徴は、なんといってもTiUPです。TiUPはTiDBの構築、バージョンアップ、スケールイン、スケールアウトの操作、さらには設定変更など、データベース運用に必要なさまざまなオペレーションを簡単に実行できる統合ツールです。MySQLではバージョンアップやスケールイン、スケールアウトを実現する場合、複数のノードにまたがってさまざまな操作をする必要があります。この作業は非常に手間がかかり、専門知識が求められるものでした。一方で、TiUPはノードの追加や削除が直感的な操作で完了します。また、バージョンアップ作業においてもTiUPはプロセスが自動化されており、短時間で信頼性の高いバージョンアップを可能にしています。TiUPのおかげで、従来のデータベース運用では複雑になりがちなオペレーションが効率化され、運用者の負担が大幅に軽減される点が魅力です。TiUPの詳細な使い方は第3章で紹介します。

TiUPは物理サーバーや仮想サーバーにTiDBを構築する場合に使用するツールであり、Kubernetes^{注13}環境でのTiDBの管理には対応していません。Kubernetes環境ではTiDB Operator^{注14}を使用することでTiDB運用を効率的に行えます。TiDB Operatorを使用したKubernetes環境でのTiDB管理については第4章で詳しく紹介します。

■ 書き込み用のエンドポイントの冗長化が容易

MySQLでは書き込みを担当するソースサーバーを冗長化する仕組みや、ソースサーバーに障害が発生した際にレプリカサーバーをソースサーバーへ昇格させる仕組みを構築する際に、さまざまな外部ツールを組み合わせる必要があります。外部ツールの設定や挙動を細かく管理する必要があり、MySQL自体の運用に加えてこれらの外部ツールの管理が運用負荷を大幅に増加させる原因となります。

TiDBでは接続を受け付けるTiDBコンポーネントがエンドポイントとなり、読み取りと書き込みの両方のSQLに対応できるため、MySQLのように読み取りと書き込みのエンドポイント分離を意識して設定を分ける必要がありません。さらにTiDBはTiDBコンポーネントを水平にスケールアウトさせることで、容易に負荷分散やエンドポイントの冗長化を実現できます。MySQLのように外部ツールを使用する必要がなく、運用者はTiDBの一元化された環境での運用に集中できるため、トラブルシューティングやメンテナンス作業の効率も大幅に向上します。TiDBを構成するコンポーネントについては第2章で詳しく説明します。

注13 <https://kubernetes.io/>

注14 <https://docs.pingcap.com/tidb-in-kubernetes/stable/tidb-operator-overview>

シャーディングが不要

MySQLではデータが増大するにつれて特に書き込み性能が劣化しやすく、この問題を回避するためにデータをシャーディングする必要が発生する場合があります。前節の「RDBMSの特徴」でも説明した通り、この作業には多大な運用コストが伴います。

一方、TiDBではこのような問題を根本的に解決するための仕組みが組み込まれています。TiDBではデータが自動的に「リージョン」と呼ばれる単位で分割され、複数のノードにまたがって分散して格納される設計が採用されています。この仕組みによって、データがどれだけ増大してもストレージ層で適切に分散処理が行われるため、運用者が意識的にシャーディングを行う必要がありません。さらに、アプリケーション側はデータの物理的な配置を意識する必要がなく、単一の論理データベースとしてTiDBを利用できます。場合によっては、TiDBにデータの分割方法をまかせるのではなくカスタマイズしたいシナリオもあるかもしれません。このようなニーズにもTiDBは柔軟に対応しており、SQLのSPLIT句を使用することで明示的にリージョン分割を行えます。

バージョンアップが容易

新機能の実装や不具合の修正を目的として、ソフトウェアのバージョンアップは定期的に行う必要があります。特にデータベースの場合はこの作業には慎重なアプローチが求められます。一般的に、稼働中のシステムに対するバージョンアップはメンテナンス時間を確保してシステムを停止し、作業を進める必要があります。この停止期間中、サービスを利用するユーザーに影響が出るためメンテナンスのスケジュール調整やダウンタイムの短縮は重要な課題です。こうした事情から、システムを停止せずにバージョンアップを実施できる仕組みは、多くの運用者にとって大きな魅力となります。

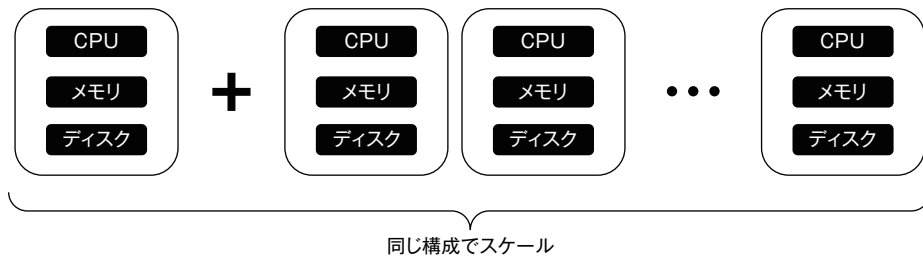
TiDBは分散システムとして単一障害点が存在しないという特徴を持っており、この利点を活かしたオンラインでのバージョンアップが可能です。具体的には、TiDBを構成する複数の内部コンポーネントを順次バージョンアップするアプローチを採用しており、TiDB全体の停止を必要としません(コンポーネントを単一サーバーで稼働させている場合は別です)。また、バージョンアップ作業はTiUPまたはTiDB Operatorによって自動化されていますので、作業の負担が軽減されています。

効率の良いスケールが可能

MySQLでは読み取り性能を向上させるために、レプリカサーバーを追加するのが一般的です。その際には、追加するレプリカサーバーは既存のレプリカサーバーと同じCPU、メモリ、ディスク構成を持つ必要があります(図1.2)。この制約により、たとえ既存のレプリカサーバーにディスク容量の余裕があったとしても、新たに追加するレプリカサーバーにもディスクを含めたフルスペックのリソースを準備する必要があり、無駄なコストの増加につながります。スケールアウトだけでなく、個々のレ

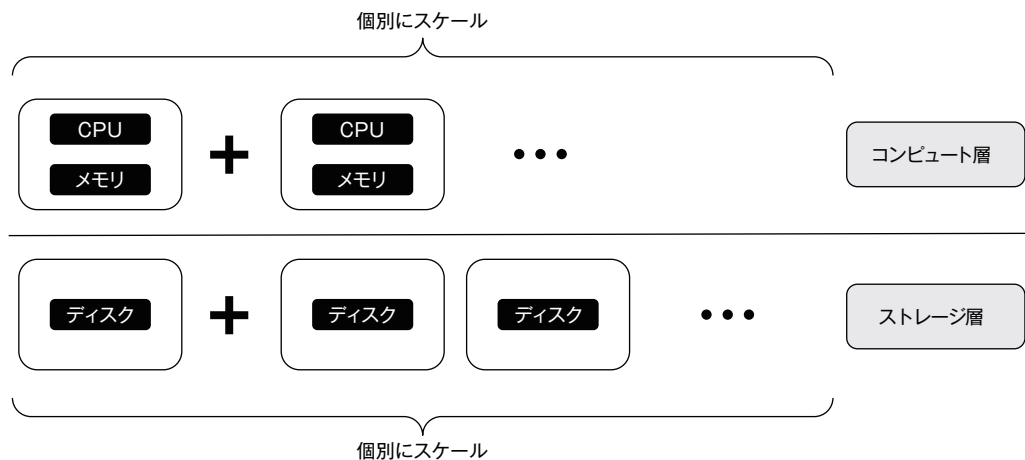
プリカサーバーやソースサーバーのCPUやメモリを増やすスケールアップの手法もありますが、これにも限界があります。ハードウェアの物理的な制約により、CPUやメモリの増設可能な上限が存在し、やがて性能向上の頭打ちに直面することになります。特にソースサーバーにおける書き込み性能のスケールアップは難しく、書き込み負荷が高まるとシステム全体の性能が低下するリスクがあります。

図1.2 従来のRDBMSのスケール方式



TiDBはそのアーキテクチャーの柔軟性により、性能向上のためのアプローチがMySQLと大きく異なります。TiDBではコンポーネント単位でのスケールアウトが可能であり、たとえば読み取りや書き込み性能を向上させたい場合にはコンピュート層のコンポーネントを、ディスク容量を増やしたい場合にはストレージ層のコンポーネントを、それぞれ必要な分だけ水平にスケールできます(図1.3)。そのため、不要なリソースを増やさずに効率的なシステム運用が可能となります。

図1.3 TiDBのスケール方式



さらにTiDBではスケールアウトだけでなくスケールインも容易に実現できる点が大きなメリットです。たとえば、システムのピーク時に合わせてコンポーネントを増やし、負荷が低下したら不要なリソースを削減することで、リソースの無駄を最小限に抑えることが可能です。この動的なスケールリングは、MySQLでは困難だった書き込み性能のスケールにも対応しており、大規模なデータ処理が求められるシステムでも高いパフォーマンスを維持できます。

データ反映の遅延を気にしないで良い

MySQLを運用する際には、ソースサーバーとレプリカサーバー間でのデータ同期が正常に行われているかどうかを常に監視しなければなりません。データ同期に遅延が発生した場合、特にリアルタイム性が求められるシステムにおいてはデータの整合性や信頼性に大きな影響を及ぼし、場合によっては致命的な障害となるからです。そのため、運用者はデータ遅延を最小限に抑えるために、同期方法やネットワークの状態を適切に管理する必要があります。このプロセスは多くのリソースと労力を要し、MySQLの運用における課題の一つとなっています。

これに対して、TiDBはその分散システム設計によって、データの反映遅延を本質的に排除しています。TiDBではクライアントからの書き込みリクエストを受け取ると、データはストレージ層に送られ分散配置される過程で複数のコピーが作成されます。まだこの段階ではクライアントに対して書き込み成功の応答を返しません。ここで重要なのは、必要なコピー数の過半数が正常に書き込まれた時点で、その書き込みが成功したとみなされることです。この段階でTiDBはクライアントに対して書き込みの成功を応答します。この仕組みにより、クライアントから見たときには書き込みが完了した時点でデータがすでにストレージ層内で十分に分散配置されていることになります。よってデータの反映遅延は気にする必要がありません。

タイムトラベルクエリが可能

TiDBはストレージ層がMVCC(Multi-Version Concurrency Control)に対応しており、データが削除や更新されたとしても元のデータが上書きされることはなく、新しいバージョンがタイムスタンプとともに追記されて記録されます。この特徴を活かし、特定の時点のデータを参照できるようになっています。

デフォルトでは10分前のデータまで保持されており、この保持期間は設定変更によって延長可能ですが、保持期間を長くするとその分ストレージ容量を消費するため、適切なバランスを考慮する必要があります。過去のデータを参照するには、SQLでAS OF TIMESTAMPを使用する(リスト1.1)か、システム変数のtidb_snapshotを使用します(リスト1.2)。このように明示的に過去の状態を指定することで、システムのトラブルシューティングやデータ監査、履歴分析などに活用できます。

注意点として、執筆時点では過去のデータをSELECTした結果をそのままINSERT INTOできません(リ

スト1.3)。その場合は一度ダンプしてから再度INSERTする必要があります。

リスト1.1 タイムトラベルクエリの例1

```
SELECT * FROM t AS OF TIMESTAMP '2024-12-01 12:00:00';
```

リスト1.2 タイムトラベルクエリの例2

```
set @@tidb_snapshot='2024-12-01 12:00:00';  
SELECT * FROM t;
```

リスト1.3 サポートされていないタイムトラベルクエリ

```
INSERT INTO t2 SELECT * FROM t AS OF TIMESTAMP '2024-12-01 12:00:00';
```

さらにTiDBではFLASHBACKも備わっており、これを使うことで誤って削除したテーブルやデータベース、さらにはTiDB全体を復元できます。たとえばDROPやTRUNCATEによって削除されたデータであっても、指定された時点まで巻き戻して復元できます。ただし、TRUNCATEされたテーブルを復元する場合には、元のテーブル名ではなく新しいテーブル名を指定する必要がある点に注意が必要です。

MySQLではこのような過去のデータを直接参照する仕組みは標準的には存在しません。過去のデータを確認するためには、事前にスナップショットを取得しておくか、ログファイルから手動で復元作業を行う必要があります。通常このような作業は手間がかかり、迅速な復元や参照が求められるシナリオには適していません。

MVCCとは

column

MVCC (Multi-Version Concurrency Control) は、データベースで複数のユーザーが同時にデータを使う時の一貫性を保証する仕組みです。MVCCはデータをバージョンとして管理し、履歴を保持します。

たとえばあるデータに対してユーザーAが書き込みをしているとします。その一方で、ユーザーBが同じデータを読みたい場合、MVCCではユーザーAの書き込みが完了するのを待たずに、ユーザーAによる変更前のバージョンをユーザーBに提供します。ユーザーAの書き込みが完了すればその新しいバージョンが他のユーザーに提供されるようになります。

こうした仕組みにより、MVCCはデータの整合性を守りながら複数のユーザーが同時にデータを使う場合でも高いパフォーマンスを実現します。MySQLのInnoDBストレージエンジンなど、多くのデータベースで採用されている技術です。

■ モニタリング環境および組み込みダッシュボードが提供されている

データベースの稼働状況を常に把握し、CPUやメモリのリソースが十分かどうか、現在処理されているクエリ数、さらにはスロークエリの発生有無を監視することは、安定した運用を維持するために非常に重要です。これらの情報を適切に把握することで、リソース不足やクエリのパフォーマンス低

下の原因となる問題を未然に防ぐことができます。

TiDBではこれらの運用監視を効率化するために、組み込みのダッシュボードが提供されています。このダッシュボードはWebインターフェースを通じて利用でき、視覚的で分かりやすい形でデータベースの稼働状況を確認できます。ダッシュボードではクエリの処理速度を示すQPS(Queries Per Second)、クエリのレイテンシー、CPUやメモリの使用量などの重要な指標をリアルタイムで確認できます。加えて、CPU使用率が高いクエリを表示したり、スロークエリのリストを表示したりする機能もあります。この機能はパフォーマンスチューニングやクエリ最適化を行う際に非常に役立ちます。

MySQLでは視覚的なダッシュボードが標準で提供されていないため、さまざまなデータを統合して視覚的に見やすくするためには追加の外部ツールが必要になります。

レイテンシーの増加傾向を抑える機能が提供されている

TiDBは分散システムとして設計されているため、クエリを処理する際にコンピュート層やストレージ層など、複数のコンポーネント間でデータやリクエストがやり取りされます。この設計により高いスケーラビリティや冗長性が実現されていますが、その一方でこれらの通信に伴うネットワークレイテンシーが発生するため、MySQLのような単一ノード型データベースに比べてレイテンシーが増加する傾向があります。このレイテンシー増加はリアルタイム性が特に重要なシステムでは性能上の課題となる場合があります。

TiDBはレイテンシーの増加を軽減するためのいくつかの機能を提供しています。その一つがコプロセッサキャッシュです。TiDBではストレージ層でデータの件数のカウントやフィルタリングなどの処理を行うコプロセッサと呼ばれる機能が実装されています。コンピュート層でコプロセッサの結果をキャッシュ可能で、クエリの処理を高速化できます。コプロセッサの詳細については第2章の「ストレージクラスター」で説明します。

次にコンピュート層における実行計画のキャッシュ機能です。この機能ではよく利用される実行計画をキャッシュとして保持することで、同じクエリを再実行する際の計算コストを削減します。これによりクエリの応答時間が短縮され、全体的なレイテンシーの低減が期待できます。ただし、このキャッシュ機能は動的なクエリや頻繁に実行計画が変更される場合には効果が限定的になります。また、実行計画のキャッシュはセッションレベルですが、TiDBのバージョン8.4.0から同じTiDBサーバー内のすべてのセッションでキャッシュを共有できるようになる機能(`tidb_enable_instance_plan_cache`)が実験的に実装されています。

もう一つの機能がメモリにテーブルを格納する機能(キャッシュテーブル)です。この機能を利用すると、特定のテーブルをSQLを処理するコンピュート層のメモリ上に配置でき、ディスクやストレージ層へのアクセスが省略されるため、データ取得の速度が大幅に向上します。この機能は、頻繁に参照されるが更新頻度が低いテーブルに対して効果を発揮します。キャッシュできるテーブルのデータは64MiBまでとなります。その他にもいくつか使用する際の制約がありますので、詳細はドキュメン

トの Cached Tables^{注15}を参照してください。

HTAPをサポートしている

TiDBはHTAP(Hybrid Transactional/Analytical Processing)をサポートしているデータベースです。オンラインでのトランザクション処理(OLTP)とオンラインでの分析処理(OLAP)を両方処理できる柔軟性を持っています。OLTPとOLAPはそれぞれ異なる性質を持つため、それぞれ専用のデータベースを用意するのが一般的です。これは、OLTPが多数のトランザクションを短時間で効率的に処理することに特化している一方で、OLAPは大量のデータを対象にした集計や分析処理を得意とし、内部のデータ構造が大きく異なるためです。

TiDBは通常OLTP処理に適している行指向ストレージが使用されますが、追加コンポーネントであるTiFlashを導入することでOLAPの処理能力を組み込むことが可能になります。TiFlashは列指向ストレージを採用しており、分析クエリに適したデータアクセスが可能になります。TiFlashを利用することでOLAPクエリを実行する際のパフォーマンスが大幅に向上し、トランザクション処理と分析処理を同一のデータセット上で実現できるようになります。TiFlashについては第2章で詳しく説明します。

MySQLはOLTP処理に特化しているため、OLAP用途に対応するためには別途専用のデータウェアハウスや分析用データベースを用意する必要があります。その場合はデータの同期に時間がかかることもあり、リアルタイム分析には適さない場合があります。また、データ移行のためのETL(Extract、Transform、Load)プロセスを設計し管理する運用コストも増大することになります。

TiDBのHTAPサポートは、たとえば取引データの即時分析を可能にすることで顧客対応の迅速化やリスクの軽減が重要な要件となるeコマースプラットフォームや金融システムにおいて特に有効です。eコマースプラットフォームや金融システムにおけるTiDBの採用事例については楽天株式会社の事例^{注16}やSBペイメントサービスの事例^{注17}が公開されています。

リクエストユニットによるリソース制御が可能

TiDBにはマルチテナント環境(同一のシステムやサービスを複数のユーザーやアプリケーションが利用する環境)での使用を想定したリソース制御の仕組みが実装されています。この仕組みは、複数のアプリケーションが同じTiDBを共有している状況で非常に有効です。マルチテナント環境では特定のアプリケーションが大量のリソースを消費する要求をした場合、他のアプリケーションの処理が遅延したり停止したりするノイジーネイバー問題が発生するリスクがあります。

TiDBではリソースをリクエストユニット(RU)という単位で管理します。このリクエストユニットは

注15 <https://docs.pingcap.com/tidb/stable/cached-tables>

注16 <https://pingcap.co.jp/case-study/rakuten/>

注17 <https://pingcap.co.jp/case-study/sbps/>

CPU消費量やストレージI/O帯域などのリソース消費量を元に計算されます。MySQLでは標準でこうした細かいリソース制御機能は提供されていません。マルチテナント環境でMySQLを使用する場合、ノイジーネイバー問題に対応するためには、アプリケーションごとに独立したMySQLインスタンスを設定するか、外部ツールを使用してリソース管理をする必要があります。TiDBのリソース制御については第7章で詳しく説明します。

TTLによるデータの自動削除が可能

TiDBではデータを格納した後に一定の時間が経つと自動的にデータを削除する機能があります。データの保持期間はTTL(Time to Live)の設定によって制御されます。TTLを適切に設定することで、不要になった古いデータを効率的にクリーンアップし、ストレージの有効活用を図ることができます。たとえば、一時的なデータを格納するテーブルや、一定期間を過ぎたログデータを自動的に削除したい場合に役立ちます。

不要なデータを定期的に削除する場合、通常はアプリケーション側で削除ロジックを実装するか、定期的なバッチ処理やスクリプトを使用する必要があります。そのため運用コストが高くなりがちですが、TiDBは標準としてTTL機能が実装されているため、運用の工数を抑えることが可能です。

TTLはテーブルを作成する際に設定するか、TTLが未設定の既存のテーブルに対して後から設定できます。また、すでに定義されているTTL設定の削除もできます。TTLの設定方法や管理方法は第7章で詳しく説明します。

MySQLと高い互換性がある

TiDBはMySQLと完全な互換性を持つわけではなく、いくつかの点で制約や差異が存在します。TiDBの導入や運用を検討する際には、これらの制約や差異を事前に理解しておくことが重要です。

まず、執筆時点ではTiDBのMySQL互換性の対象となるMySQLのバージョンは5.7および8.0です。次に機能面での違いとして、TiDBはストアドプロシージャやトリガー、UDF(User-defined Functions)といった機能には対応していません。また、MySQLのレプリケーション機能を使用してTiDBをソースサーバーとしたレプリカサーバーを作成できません。ただし、TiDBとMySQLの間でデータを同期する必要がある場合には、TiCDCを利用することで対応が可能です。TiCDCは、TiDBからMySQLやApache Kafkaなどへのデータ変更をリアルタイムでストリームするツールであり、異なるデータベース間の同期を効率的に実現します。TiCDCについては第6章で詳しく説明します。MySQLとの互換性についてはドキュメントのMySQL Compatibility^{注18}を参照してください。

注18 <https://docs.pingcap.com/tidb/stable/mysql-compatibility>

1-4 TiDBの提供形態

TiDBは執筆時点でPingCAPより3つの形態で提供されています。どの形態が適しているのかを検討しやすいように、それぞれの提供形態の詳細を説明します。

TiDB Cloud

TiDB Cloudはパブリッククラウド上でTiDBをマネージドサービスとして利用できるサービスです。TiDBは分散システムであるがゆえに動作させるべきコンポーネントが多くなり、それだけに運用にかかる工数も少なくないのですが、その運用をマネージド側で吸収してくれる点は大きな魅力と言えるでしょう。

TiDB Cloudには2つのサービス形態があり、TiDB Cloud DedicatedおよびTiDB Cloud Serverlessが利用できます。それぞれTiDBを使えることに変わりはないのですが、サービス内容や特徴は少々異なります。それぞれの特徴については第5章で詳しく説明します。

TiDB Self-Managed

TiDB Self-ManagedはオンプレミスのサーバーやKubernetesクラスターにTiDBを自身で構築する提供形態です。TiDB Cloudでは実現できない高度なカスタマイズが必要な場合や、複数のクラウド環境にまたがるシステム要件がある場合はこちらの提供形態を検討してください。TiDB Self-Managedでは自身でTiDBを構築した場合でもPingCAPより技術サポートを受けることが可能です。

TiDB Community

TiDB Communityはオープンソースのソフトウェアです。TiDBの運用や構築を自身で行い、PingCAPによる技術サポートを受けなくてよい場合はこちらの提供形態を検討してください。物理サーバーや仮想サーバー上に構築する手順は第3章、Kubernetesクラスター上に構築する手順は第4章で説明します。

エンタープライズ版とオープンソース版の機能の違い

column

エンタープライズ版であるTiDB CloudおよびTiDB Self-Managedとオープンソース版であるTiDB Communityには一部機能の差があります。エンタープライズ版はオープンソース版に加えて、監査機能やサポートサービスが提供されています。

利用目的での比較

利用目的に合わせたTiDBの提供形態を表1.2にまとめます。

表1.2 利用目的に合わせたTiDBの提供形態

	TiDB Cloud	TiDB Self-Managed	TiDB Community
マネージドサービスで利用したい	○		
PingCAPによるサポートを受けたい	○	○	
監査機能が必要	○	○	
パブリッククラウドで利用したい	○	○	○
オンプレミスで利用したい		○	○
Kubernetesクラスター上に構築したい		○	○

1-5

まとめ

本章ではTiDBの概要や歴史に触れながら、TiDBにどのような特徴があるのかを説明しました。TiDBの特徴をしっかりと理解し、どの提供形態が自分たちに適しているかを把握することは、これからTiDBを実際に触っていく上でとても重要なことです。次章ではTiDBの内部のアーキテクチャーがどうなっているのかを説明していきます。

第 2 章

TiDBを構成する コンポーネント

可用性と拡張性が高い分散システム

TiDBは分散システムで構成されています。大まかに分類するとコンピュート層とストレージ層に分かれますが、内部ではもう少し細かい単位で役割に応じたコンポーネントが複数存在し、それぞれが協調して動作することでTiDBは高いパフォーマンスと可用性を実現しています。本章ではTiDBを支えている各種コンポーネントを深掘りしていきます。

2-1

TiDBの構成

TiDBを構成する上で必須なコンポーネントはTiDBクラスター、PDクラスター、ストレージクラスターの3つになります(図2.1)。それ以外にもオプションとしてさまざまな機能を持ったコンポーネントがありますので、要件に応じて選択してください(表2.1)。

図2.1 TiDBのアーキテクチャー

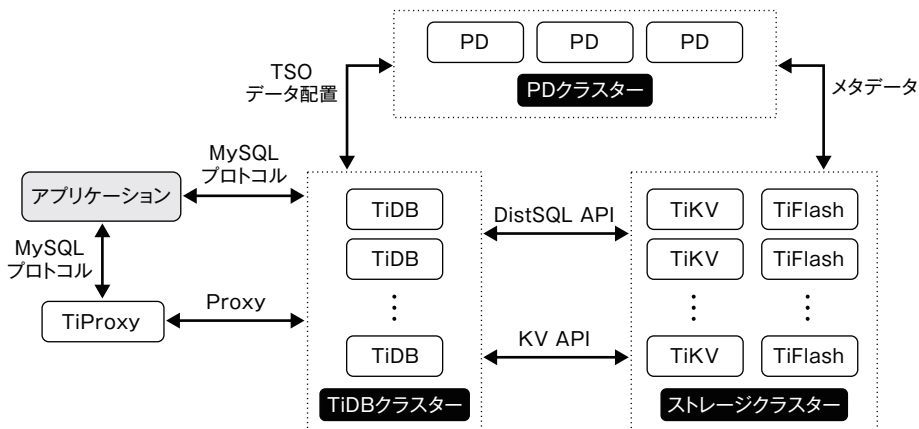


表2.1 TiDBを構成するコンポーネント

コンポーネント	概要
TiDBクラスター	SQLを処理するコンピュート層を担当するコンポーネント
PDクラスター	データ配置の管理、メタデータ管理を担当するコンポーネント
ストレージクラスター (TiKV)	ストレージ層を担当するコンポーネント (行指向データ形式)
ストレージクラスター (TiFlash)	ストレージ層を担当するコンポーネント (列指向データ形式)
TiProxy	TiDBクラスターの負荷分散、コネクションハンドリングを担当するコンポーネント
TiCDC	変更データをリアルタイムに外部へ転送するコンポーネント
Monitoring	メトリクス収集のためのPrometheusコンポーネント
Grafana	収集したメトリクスを可視化するためのダッシュボードを表示するコンポーネント
Alertmanager	Prometheusで定義されたアラートの通知を管理するコンポーネント

2-2

TiDB クラスター

2

TiDB クラスターはコンピュータ層にあたるコンポーネントで、アプリケーションからの接続を受け付けて SQL を処理し、ストレージクラスターと通信してデータの読み取りや書き込みを行う役割を担当します。TiDB クラスターは複数の TiDB サーバーで構成され、各 TiDB サーバーはステートレスであるためスケールアウトおよびスケールインが容易です。前段にロードバランサーや後述する TiProxy を使用することで、複数の TiDB サーバーにトラフィックを分散させて可用性と SQL の処理性能を向上させることができます。

TiDB クラスターは MySQL のオプティマイザーのような役割を持っており、ストレージクラスターと通信してデータを処理する際に TiKV または TiFlash のどちらが効率のかを判断し、通信先を選択します。TiKV と通信してデータを読み書きする場合、TiDB クラスターは SQL を解析してキーバリュー形式に変換してデータを読み書きします。これは TiKV がキーバリュー形式でデータを保存しているためです。

TiKV にアクセスする際の SQL の変換

実際にテーブル内のデータがどのように変換されて TiKV に保存されているかを実際のデータを元に説明します。

TiDB はクラスター化テーブルと非クラスター化テーブルで格納する際のキーが変わります。クラスター化テーブルとはテーブルの行が主キーと同じ順序で格納されているテーブルを意味します。非クラスター化テーブルは行の格納順が主キーと一致していないテーブルを意味します。なお、この文脈におけるクラスター化という用語は、連携して動作するデータベースサーバーのグループではなく、データの格納方法の構成を指しています。

クラスター化テーブルは主キーでアクセスした場合に行レコードを直接取得できますが、非クラスター化テーブルでは保存されている主キーから行の ID に相当する RowID を取得し、その RowID を元に行レコードを取得するためテーブルに対する操作が1つ増えることになります。

クラスター化テーブルのデータはリスト 2.1 の形式、非クラスター化テーブルのデータはリスト 2.2 の形式になります。tablePrefix は t、recordPrefixSep は r と固定値になります。{tableID} は TiDB によってテーブルごとに割り当てられるユニークな ID (テーブル ID) になります。

リスト 2.1 クラスター化テーブルのキーバリュー

キー: tablePrefix{tableID}_recordPrefixSep{主キーの値}
バリュー: [カラム2, カラム3, ...] (主キーのカラム以外のカラム)

リスト2.2 非クラスター化テーブルのキーバリュー

キー: tablePrefix{tableID}_recordPrefixSep{RowID}

バリュー: [カラム1、カラム2, ...]

たとえば次のようなテーブルt1があるとします。データとして3件登録しておきます。

テーブルの作成とデータの登録

```
mysql> USE test;
Database changed

mysql> CREATE TABLE t1 (
  -> id BIGINT PRIMARY KEY,
  -> name VARCHAR(256),
  -> age INT,
  -> UNIQUE (name),
  -> INDEX (age)
  -> );
Query OK, 0 rows affected (0.09 sec)

mysql> INSERT INTO t1 VALUES (100, 'Alice', 20), (200, 'Bob', 30), (300, 'Charlie', 40);
Query OK, 3 rows affected (0.01 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM t1 ORDER BY id;
+-----+-----+-----+
| id   | name  | age  |
+-----+-----+-----+
| 100  | Alice | 20   |
| 200  | Bob   | 30   |
| 300  | Charlie | 40  |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

テーブルIDを調べるにはINFORMATION_SCHEMA内のTABLESを参照します。

テーブルIDの確認

```
mysql> SELECT TIDB_TABLE_ID FROM INFORMATION_SCHEMA.TABLES
  -> WHERE table_schema='test' AND table_name='t1';
+-----+
| TIDB_TABLE_ID |
+-----+
|          112  |
+-----+
1 row in set (0.00 sec)
```

t1テーブルには112のテーブルIDが割り当てられたことが分かります。次にt1がクラスター化テーブルか非クラスター化テーブルかを調べます。同じようにTABLESを参照します。TIDB_PK_TYPEカラム

に情報が格納されています。

```

クラスター化の確認
mysql> SELECT TIDB_PK_TYPE FROM INFORMATION_SCHEMA.TABLES
-> WHERE table_schema='test' AND table_name='t1';
+-----+
| TIDB_PK_TYPE |
+-----+
| CLUSTERED    |
+-----+
1 row in set (0.01 sec)

```

CLUSTEREDであればクラスター化テーブルとなります。NONCLUSTEREDであれば非クラスター化テーブルです。TiDBでは主キーを指定する際に、CLUSTEREDまたはNONCLUSTEREDを指定できます。指定しなかった場合はCLUSTEREDとして主キーが作成されます。

```

非クラスター化テーブルの作成1
mysql> USE test;
Database changed

mysql> CREATE TABLE t2 (
->   id BIGINT PRIMARY KEY NONCLUSTERED,
->   name VARCHAR(256),
->   age INT,
->   UNIQUE (name),
->   INDEX (age)
-> );

```

SQLのコメント構文も使用できます。

```

非クラスター化テーブルの作成2
mysql> CREATE TABLE t2 (
->   id BIGINT PRIMARY KEY /*T![clustered_index] NONCLUSTERED */,
...
(省略)
...

```

t1テーブルはクラスター化テーブルですので、実際のt1テーブル内のデータはリスト2.1の形式で次のように変換されてTiKVに格納されます。

```

t1テーブルのキーバリュー変換
t112_r100 -> ["Alice", 20]
t112_r200 -> ["Bob", 30]
t112_r300 -> ["Charlie", 40]

```

非クラスター化テーブルの例を示します。t2テーブルに次のようにデータを挿入します。

t2テーブルの作成とデータの登録

```
mysql> INSERT INTO test.t2 VALUES (100, 'Alice', 20), (200, 'Bob', 30), (300, 'Charlie', 40);
Query OK, 3 rows affected (0.01 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

RowIDは `_tidb_rowid` を SELECT に指定することで調べられます。SELECT 時に `*` を指定しても表示できませんので注意してください。なお、`TIDB_PK_TYPE` が `CLUSTERED` のテーブルでは `_tidb_rowid` は存在しないため SELECT できません。

RowIDの確認

```
mysql> SELECT _tidb_rowid, id, name, age FROM test.t2 ORDER BY id;
+-----+-----+-----+-----+
| _tidb_rowid | id | name | age |
+-----+-----+-----+-----+
|          1 | 100 | Alice | 20 |
|          2 | 200 | Bob   | 30 |
|          3 | 300 | Charlie | 40 |
+-----+-----+-----+-----+
3 rows in set (0.01 sec)
```

t2 テーブル内のデータはリスト 2.2 の形式で TiKV に格納されます。t2 のテーブル ID が 119 である場合、次のようになります。

t2 テーブルのキーバリュー変換

```
t119_r1 -> [100, "Alice", 20]
t119_r2 -> [200, "Bob", 30]
t119_r3 -> [300, "Charlie", 40]
```

インデックスも同様に変換されてキーバリュー形式で保存されます。ユニークインデックスであればリスト 2.3 の形式、非ユニークインデックスであればリスト 2.4 の形式でキーの値が変わります。`indexPrefixSep` は `i` と固定値になります。

リスト2.3 ユニークインデックスのキーバリュー

```
キー: tablePrefix{tableID}_indexPrefixSep{indexID}_indexedColumnsValue
バリュー: _tidb_rowid
```

リスト2.4 非ユニークインデックスのキーバリュー

```
キー: tablePrefix{tableID}_indexPrefixSep{indexID}_indexedColumnsValue_{RowID}
バリュー: NULL
```

{indexID} は TiDB によってインデックスごとに割り当てられるユニークな ID (インデックス ID) になります。{indexID} は `INFORMATION_SCHEMA` 内の `TIDB_INDEXES` を参照して調べることができます。

インデックスの確認

```
mysql> SELECT INDEX_ID, COLUMN_NAME FROM INFORMATION_SCHEMA.TIDB_INDEXES
```



```

-> WHERE table_schema='test' AND table_name='t1';
+-----+-----+
| INDEX_ID | COLUMN_NAME |
+-----+-----+
|      0 | id          |
|      1 | name        |
|      2 | age         |
+-----+-----+
3 rows in set (0.00 sec)

```

t1テーブルはインデックスが3つあり、インデックスID0は主キー、インデックスID1はnameカラムのユニークインデックス、インデックスID2はageカラムの非ユニークインデックスです。t1テーブルのインデックスは次のように変換されます。

t1テーブルのインデックスのキーバリュー変換

```

t112_i0_100 -> 1
t112_i0_200 -> 2
t112_i0_300 -> 3

t112_i1_Alice -> 1
t112_i1_Bob   -> 2
t112_i1_Charlie -> 3

t112_i2_20_1 -> NULL
t112_i2_30_2 -> NULL
t112_i2_40_3 -> NULL

```

非ユニークインデックスはバリューがNULLになりますが、これはインデックスの値(indexedColumns Value)が重複する可能性があるため、キーの値をユニークにするためにRowIDをキーの名前として付与しているからになります。このようなキーにすることで、キーのプリフィックス(tablePrefix{tableID}_indexPrefixSep{indexID}_indexedColumnsValue_)を指定してインデックスを検索してRowIDを取得できます。そのため、バリューには何も入れる必要がありません。

TiFlashへの振り分け

TiFlash コンポーネントを使用している場合、SQLの解析結果によってTiFlashとTiKVどちらからデータを取得した方が効率が良いかをTiDBクラスターは自動で判断します。また、TiDBクラスターに任せずに手動でデータの取得先を指定可能です。その場合はSQLのヒントでREAD_FROM_STORAGEを使用するか、セッション内でtidb_isolation_read_enginesを指定します。

次の例は、何も指定しなかった場合にTiFlashからデータを取得するSQLです。taskの内容を確認するとmpp[tiflash]となっていることが分かります。

TiFlashへの振り分け

```
mysql> EXPLAIN SELECT COUNT(*) FROM sample;
+-----+-----+-----+ ...
| id          | estRows | task      | ...
+-----+-----+-----+ ...
| HashAgg_22   | 1.00    | root      | ...
|   └─TableReader_24 | 1.00    | root      | ...
|     └─ExchangeSender_23 | 1.00    | mpp[tiflash] | ...
|       └─HashAgg_9 | 1.00    | mpp[tiflash] | ...
|         └─TableFullScan_21 | 20000.00 | mpp[tiflash] | ...
+-----+-----+-----+ ...
5 rows in set (0.00 sec)
```

ヒントを加えることでTiFlashから取得する計画だったものがTiKVから取得する計画に変わります。READ_FROM_STORAGEはTIKVまたはTIFLASHを指定します。[]内はテーブル名を指定します。taskの内容がcop[tikv]となっていることが分かります。

TiKVへの振り分け

```
mysql> EXPLAIN SELECT /*+ READ_FROM_STORAGE(TIKV[sample]) */ COUNT(*) FROM sample;
+-----+-----+-----+ ...
| id          | estRows | task      | ...
+-----+-----+-----+ ...
| HashAgg_13   | 1.00    | root      | ...
|   └─TableReader_14 | 1.00    | root      | ...
|     └─HashAgg_6 | 1.00    | cop[tikv] | ...
|       └─TableFullScan_12 | 20000.00 | cop[tikv] | ...
+-----+-----+-----+ ...
4 rows in set (0.01 sec)
```

システム変数のtidb_isolation_read_enginesを変更することでもデータの取得先を変更できます。tidb_isolation_read_enginesはデフォルトではtikv,tiflash,tidbが設定されています。たとえば次のようにtidb_isolation_read_enginesをtikv,tidbのみに設定することで、TiFlashからデータを取得しないようにできます。

データ取得先の変更

```
mysql> SET @@session.tidb_isolation_read_engines = 'tikv,tidb';
Query OK, 0 rows affected (0.00 sec)

mysql> EXPLAIN SELECT COUNT(*) FROM sample;
+-----+-----+-----+ ...
| id          | estRows | task      | ...
+-----+-----+-----+ ...
| HashAgg_13   | 1.00    | root      | ...
|   └─TableReader_14 | 1.00    | root      | ...
|     └─HashAgg_6 | 1.00    | cop[tikv] | ...
+-----+-----+-----+ ...
```

```
|      └─TableFullScan_12 | 20000.00 | cop[tikv] | ...
+-----+-----+-----+-----+
4 rows in set (0.01 sec)
```

TiFlashについては2-4節で詳しく説明します。

EXPLAIN時に表示されるtaskの意味

column

rootはTiDBサーバーによる処理を意味しています。cop[tikv]はTiKVのコプロセッサによる処理を意味しています。TiKVのコプロセッサについては2-4節内の「TiKVのコプロセッサ」を参照してください。mpp[tiflash]はTiFlashのMPP(Massively Parallel Processing)による処理を意味しています。MPPについてはドキュメントのUse TiFlash MPP Mode^{注a}を参照してください。MPPはデフォルトで有効になっています。

注a <https://docs.pingcap.com/tidbcloud/use-tiflash-mpp-mode>

2-3

PDクラスター

PD(Placement Driver)クラスターはTiDB全体の言わば頭脳のような存在です。PDクラスターの主な役割は大きく分けてデータの配置管理とTSO(TimeStamp Oracle)の発行です。これらの機能を通じて、PDクラスターはTiDBの効率的なデータ管理と分散トランザクション処理を支えています。本節ではPDクラスターの各役割について説明します。

データの配置管理

TiDBはデータをリージョンという単位に分割して管理し、ストレージクラスターに分散して保存します。PDクラスターはこの時、リージョンをストレージクラスターのどのノードに配置するか決定します。また、特定のリージョンに対してアクセスが集中し、ホットスポットが発生した場合はそのリージョンをストレージクラスター間で移動させてリバランシングを行います。さらに、サイズが小さくなったリージョン同士の統合や、サイズが大きくなりすぎたリージョンの分割といったタスクも担当しており、これによりストレージの効率的な利用が可能になりTiDB全体のパフォーマンスが最適化されます。

TSOの発行

PDクラスターは分散トランザクションの管理のためにTSO(TimeStamp Oracle)を発行します。この

TSOは454623537343758337のような18桁の数字で表示されます。TSOはタイムスタンプとして単調な増加を保証し、常に一意です。TiDBにおける分散トランザクション処理を実現するための基盤となる仕組みで、PDクラスターはTiDBクラスターに対してトランザクションの開始時とコミット時にTSOを発行します。これによりトランザクションは順序を持つことができ、分散環境においても一貫性の担保が可能になります。

TiDBにおける分散トランザクションを支える技術

column

TiDBでは分散システムにおけるトランザクションを実現するために、さまざまな技術が使われています。TSOもその技術の一つです。その他に使われている技術として2PC(2 Phase Commit)や、Googleが開発したPercolatorなどの技術があります。

TiDBはトランザクションを開始する際に開始タイムスタンプとしてTSOを取得し、トランザクション対象のリージョンすべてに対して開始タイムスタンプと共に事前書き込みを要求し、対象データをロックします。すべての事前書き込みに成功したら、コミットタイムスタンプとしてTSOを再度PDクラスターから取得し、コミットタイムスタンプと共に再度コミットしてロックを解除します。その後トランザクションの成功を返します。この一連の処理は2PCと呼ばれています。

Percolatorは、Google内部で使用される分散データストレージシステムの一部として設計されました。その主な特徴は大規模なデータセットに対して効率的なトランザクション処理を提供する点にあります。従来の2PCはロックの長時間保持やパフォーマンスの低下といった課題がありましたが、Percolatorではこれらを改善するための工夫が施されています。Percolatorについての詳細はPercolatorの論文^{注a}を参照してください。

注a https://www.usenix.org/legacy/event/osdi10/tech/full_papers/Peng.pdf

TiDBクラスターからのTSO発行リクエストの並列化

TSOの発行処理に時間がかかっているにもかかわらず、PDクラスターのCPUやネットワーク帯域に余裕がある場合は、TiDBクラスターからのTSO発行リクエストを並列化することでTiDBのパフォーマンスを向上できる可能性があります。TSO発行のパフォーマンスを確認するには、モニタリング環境を有効にして「PD TSO RPC Duration」や「PD server TSO handle duration」のGrafanaダッシュボードを確認してください。モニタリング環境については第3章内の「モニタリング環境の構築」を参照してください。

通常TiDBクラスターはPDクラスターに対してTSOの発行要求をする際に、シーケンシャルにリクエストを送信します。この挙動はtidb_tso_client_rpc_modeの設定で変更できます。デフォルトはDEFAULTが設定されていて並列化はされません。

```
tidb_tso_client_rpc_modeの確認
mysql> SELECT @@tidb_tso_client_rpc_mode;
+-----+
| @@tidb_tso_client_rpc_mode |
+-----+
| DEFAULT                     |
+
```

```
+-----+
1 row in set (0.00 sec)
```

tidb_tso_client_rpc_modeをPARALLELに設定することでTiDBクラスターからのTSO発行リクエストを2並列にできます。また、PARALLEL-FASTに設定すると4並列にできます。

```
tidb_tso_client_rpc_modeの変更
mysql> SET GLOBAL tidb_tso_client_rpc_mode = 'PARALLEL';
Query OK, 0 rows affected (0.03 sec)
```

PARALLELおよびPARALLEL-FASTを設定した場合、PDクラスターへの負荷が上がりますので設定する際には十分注意してください。

▶ TSOを時刻へ変換

TSOは先頭46bitに物理タイムスタンプの情報が格納され、続く18bitに論理タイムスタンプが格納されています。

```
TSOの構成
mysql> SELECT
  -> BIN(454310362294255617) AS TSO,
  -> BIN(454310362294255617 >> 18) AS PHY_TS,
  -> LPAD(BIN(454310362294255617 & 0x3FFFF), 18, 0) AS LOG_TS \G
***** 1. row *****
TSO: 1100100111000001000110111100001001001010100000000000000001
PHY_TS: 11001001110000010001101111000010010010101
LOG_TS: 0000000000000000001
1 row in set (0.00 sec)
```

物理タイムスタンプはミリ秒単位のUNIXタイムスタンプの値になります。FROM_UNIXTIME()を使用してUNIXタイムスタンプを日付に変換できます。ミリ秒での数値のため、1000で割ってから変換します。

```
物理タイムスタンプのUNIXタイムスタンプ
mysql> SELECT FROM_UNIXTIME((454310362294255617 >> 18) / 1000);
+-----+
| FROM_UNIXTIME((454310362294255617 >> 18) / 1000) |
+-----+
| 2024-12-01 12:34:56.7890 |
+-----+
1 row in set (0.01 sec)
```

論理タイムスタンプは同じミリ秒内に複数のタイムスタンプが必要なシナリオや、特定のイベントによってクロックの進行が逆転する可能性があるシナリオで使用される増分カウンターになります。

TSOの値から日付に変換する場合は便利な関数を用意されています。TIDB_PARSE_TSO()は物理タイムスタンプを表示する関数です。論理タイムスタンプを表示する場合はTIDB_PARSE_TSO_LOGICAL()を使用します。

SQLでTSOの変換

```
mysql> SELECT
  -> TIDB_PARSE_TSO(454310362294255617),
  -> TIDB_PARSE_TSO_LOGICAL(454310362294255617) \G
***** 1. row *****
      TIDB_PARSE_TSO(454310362294255617): 2024-12-01 12:34:56.789000
TIDB_PARSE_TSO_LOGICAL(454310362294255617): 1
1 row in set (0.00 sec)
```

SQLによる変換以外に、コマンドラインツールであるTiUPを使用してTSOから物理タイムスタンプと論理タイムスタンプに変換できます。変換する場合は `tiup ctl:<version> pd tso` コマンドを使用します。

コマンドラインでTSOの変換

```
$ tiup ctl:v8.5.0 pd tso 454310362294255617
Starting component ctl: ...
system: 2024-12-01 12:34:56.789 +0000 UTC
logic: 1
```

TiUPについては第3章で説明します。

▶ 時刻からTSOを生成

先ほど変換した時刻2024-12-01 12:34:56.789000からTSOを生成してみましょう。論理タイムスタンプは1とします。まず物理タイムスタンプの文字列をUNIXタイムスタンプに変換します。TSOではミリ秒までのUNIXタイムスタンプを使用するため、値を1000倍して表示します。

物理タイムスタンプのミリ秒UNIXタイムスタンプ

```
mysql> SELECT UNIX_TIMESTAMP('2024-12-01 12:34:56.789000') * 1000;
+-----+
| UNIX_TIMESTAMP('2024-12-01 12:34:56.789000') * 1000 |
+-----+
| 1733056496789.000000 |
+-----+
1 row in set (0.00 sec)
```

1000倍したタイムスタンプは2進数に変換します。

物理タイムスタンプの2進数変換

```
mysql> SELECT BIN(1733056496789.000000);
+-----+
| BIN(1733056496789.000000) |
+-----+
| 11001001110000010001101111000010010010101 |
+-----+
1 row in set (0.00 sec)
```

次に論理タイムスタンプ部分です。1を2進数で表示し、18bitで表現します。

```

論理タイムスタンプの2進数変換
mysql> SELECT LPAD(BIN(1), 18, 0);
+-----+
| LPAD(BIN(1), 18, 0) |
+-----+
| 000000000000000001 |
+-----+
1 row in set (0.00 sec)

```

これでTSOを構成する物理タイムスタンプと論理タイムスタンプの情報がそろいました。物理タイムスタンプと論理タイムスタンプを結合してTSOを生成します。

```

TSOの生成
mysql> SELECT CONV(
->   CONCAT(
->     "11001001110000010001101111000010010010101",
->     "000000000000000001"
->   ), 2, 10) AS TSO;
+-----+
| TSO          |
+-----+
| 454310362294255617 |
+-----+
1 row in set (0.00 sec)

```

2024-12-01 12:34:56.789000からTSOとして454310362294255617が生成されました。現在時刻を元にTSOを生成する場合は、NOW(3)を使用してUNIXタイムスタンプを取得し、同様の手順でTSOを生成できます。

PDクラスターの死活監視

PDクラスターは単一ノードで稼働できますが、通常は複数のPDノードで構成されており、PDノードの1台がリーダーとして選出され処理を担当します。その際他のノードはフォロワーとして動作します。リーダーノードに障害が発生した場合はフォロワーの中からリーダーが選出され、リーダーが切り替わります。リーダー選出はRaft^{注1}によるコンセンサスアルゴリズムが使用されています。Raftにおける役割の説明を表2.2に示します。

注1 <https://raft.github.io/>

表2.2 Raftグループの役割の説明

役割	詳細
リーダー	グループ内で常に1つで、クライアントからの処理を担当する
フォロワー	グループ内で投票権を持つ（リーダー候補）
ラーナー	グループ内で投票権を持たない（リーダー候補ではない）

Raftではスプリットブレインを防ぐため、クォーラム（クラスターの過半数）が同意することで決定が承認されます。そのため、PDクラスターは3台以上の奇数台で構成されることが推奨されます。

スプリットブレインとは

column

スプリットブレインは分散システムにおいて本来はリーダーが1つであるべきなのに、複数のリーダーが存在してしまう状態を指します。たとえば2台のノードで構成されているクラスターがあり、ネットワークの分断により1台ずつに分かれてしまったとしましょう。この場合、お互いは相手がダウンしたと検知してリーダーになろうとします。結果として、2つのグループがそれぞれリーダーとして稼働することになり、データの整合性が失われることになります。

これを防ぐためには、奇数台でクラスターを構成して、分断された場合でもどちらか一方のみがクォーラムを満たせるようにすることが重要になります。

リーダーはPDクラスター内の他のフォロワーに対して定期的にハートビートを送信します。このハートビートはリーダーが健在であることを知らせ、フォロワーたちがリーダーの存続を確認する手段となります。リーダー障害時にはフォロワーはハートビートを受信しなくなるため、タイムアウト期間が経過するとフォロワーはリーダーが不在であると判断します。リーダーが不在と判断されたフォロワーは候補者となり、新たなリーダーになるために選挙を開始します。候補者は他のPDノードに対して投票を依頼し、過半数の票を獲得した場合に新たなリーダーとなることができます。新しいリーダーが選出された後、TSOの発行やデータ配置のスケジューリングなどはすぐに新しいリーダーによって引き継がれます。このプロセスが迅速に行われるため、TiDB全体に対する影響を抑えつつ、高可用性を維持しています。

ストレージクラスターの死活監視

PDクラスターはストレージクラスターのノードから定期的に2種類のハートビート、ストアハートビートとリージョンハートビートを受信しています。各ストレージクラスターのノードは、ディスク使用率や負荷状況などの自分の状態をストアハートビートとして、リージョンに関する情報をリージョンハートビートとして、PDクラスターへ定期的に送信します。PDクラスターはこの2種類のハートビートを受信することで、どのリージョンがどのノードに配置されているのか、各ノードがどれくらいのリソースを使っているのかを把握できますので、効率的なデータ配置が可能になります。

PDクラスターのパフォーマンス

PDクラスターはTiDBの頭脳のような存在であり、PDクラスターが高負荷になるとTiDB全体のパフォーマンスに影響を及ぼす可能性があります。たとえば、高負荷時にはTSOの発行が遅延し、トランザクション処理が遅延することになります。また、ストレージノード間の負荷分散がうまくいかなくなることで、特定のストレージノードに負荷が集中して過負荷のノードが生まれ、これがボトルネックとなりクラスター全体のパフォーマンスの低下を引き起こす可能性があります。

そのため、PDクラスターがボトルネックにならないように、十分なりソースを割り当てておくことはTiDBを運用する上で非常に重要となります。ストレージクラスターのノードの数を増やす時は特に注意してください。ストレージクラスターからのハートビートが増えてPDクラスターが高負荷になる可能性があるためです。その場合はPDクラスターのメトリクスを確認し、必要に応じてチューニングのために各種パラメーターの調整をしてください。モニタリングを有効にしている場合は、Grafanaダッシュボードの「TiDB - PD server TSO handle time」および「Heartbeat - TiKV side heartbeat statistics」グラフが参考になります。

PDマイクロサービスモード

TiDBのバージョン8.0.0から実験的な機能として、TSOの発行機能とデータの負荷分散を行うスケジュール機能をPDクラスター外のコンポーネントに分離するPDマイクロサービスモードが導入されました。PDマイクロサービスはPDクラスターのパフォーマンスのボトルネックを解決し、TSO発行のレイテンシー増加やスケジューリング障害によるTiDB全体のサービス停止といったリスクを軽減します。

PDマイクロサービスを有効にすると、PDクラスターとは別にtsoサービスとschedulingサービスがそれぞれ起動します。tsoサービスはTSOの発行を担当し、schedulingサービスはデータの配置管理を担当します。tsoサービスやschedulingサービスはそれぞれ冗長化のために複数起動可能です。1台がプライマリーとして動作し、障害が発生するとセカンダリーがプライマリーに昇格してサービスを継続します。PDマイクロサービスを有効にした場合、PDクラスターはTSOを発行しなくなります。そのためtsoサービスは冗長化しておくことをおすすめします。schedulingサービスはデフォルトでenable-scheduling-fallbackが有効になっており、schedulingサービスがダウンした場合はPDクラスターがスケジュール機能を引き継ぐようになっています。

Active PD Follower機能

PDクラスターが扱うリージョンの数が大量にある場合や、TiDBサーバーの数が多く場合、PDクラスター内のリーダーのCPU使用率が高くなる可能性があります。通常PDクラスター内のリーダーだけがTiDBサーバーからのリージョン情報のリクエストを処理しますが、pd_enable_follower_handle_regionを有効にすると、TiDBサーバーはPDクラスター内のリーダー以外からもリージョン情報のリ

クエストを取得できるようになります。この機能によりPDクラスターのリーダーの負荷を分散させることができます。

Active PD Followerの確認

```
mysql> SELECT @@global.pd_enable_follower_handle_region;
+-----+
| @@global.pd_enable_follower_handle_region |
+-----+
|                                           0 |
+-----+
1 row in set (0.00 sec)
```

デフォルトはOFF(0)でActive PD Followerは無効です。有効にするにはON(1)に設定します。

Active PD Follower機能の有効化

```
mysql> SET GLOBAL pd_enable_follower_handle_region = ON;
```

2-4

ストレージクラスター

ストレージクラスターは実際のデータを保存するコンポーネントであり、TiKVおよびTiFlashがこの役割を担います。TiKVは分散キーバリューストアであり、かつトランザクション処理にも対応しています。一貫性を保ちながら水平にスケール可能で大規模データを扱うことができます。TiFlashはデータを列指向で保存するコンポーネントであり、分析クエリに特化しています。

TiKV

TiKVはTiDBと同じくオープンソースソフトウェアとして公開^{注2}されています。また、堅牢性とパフォーマンスに優れた開発言語であるRustが採用されている点も特徴のソフトウェアです。TiKVはCNCF (Cloud Native Computing Foundation)のプロジェクトであり、十分に成熟しているとされているGraduatedプロジェクトとして認定^{注3}されています。ライセンスはApache License 2.0^{注4}です。

注2 <https://github.com/tikv/tikv>

注3 <https://www.cncf.io/projects/tikv/>

注4 <https://www.apache.org/licenses/LICENSE-2.0>

CNCFについて

column

CNCFはLinux Foundationのプロジェクトの一つであり、コンテナ技術の推進やその進化を取り巻くテクノロジー業界の足並みをそろえるために設立された財団です。CNCFではプロジェクトのライフサイクルが定義^{注a}されており、成熟度に応じてSandbox、Incubation、Graduatedと遷移します。CNCFがどのようなプロジェクトを管理していて、どのようなライフサイクルが定義されているのかはランドスケープ^{注b}より確認できます。CNCFはクラウドネイティブの定義も行っており、GitHub上から日本語で確認^{注c}できます。日本では2023年11月にCNCF公認のCNCJ(Cloud Native Community Japan)が発足しました。

注a <https://github.com/cncf/toc/blob/main/process/README.md>

注b <https://landscape.cncf.io/>

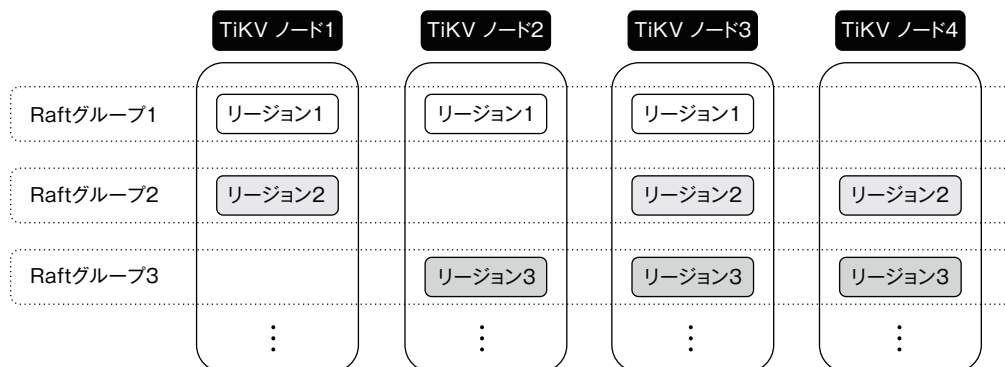
注c <https://github.com/cncf/toc/blob/main/DEFINITION.md#日本語版>

▶ リージョンとRocksDB

TiKVはデータを保存する際にRocksDBに保存します。RocksDBはMeta(旧Facebook)が開発したストレージエンジンであり、データ構造にLSM ツリー(Log Structured Merge-Tree)が使われています。RocksDBの詳細はここでは触れませんが、詳細を知りたい方はTiDBのドキュメント内のRocksDBの概要^{注5}を参照してください。

リージョンはデフォルトでは異なるTiKVノード間で3つにコピーされます(図2.2)。リージョンごとにRaftグループが作成され、3つのリージョンのうち1つがリーダーとなります。リージョンに対する書き込みはリーダーのみに対して行われ、残りのノードはフォロワーとなります。リージョンのリーダーにデータが書き込まれた際、TiKVはすぐにクライアントに書き込み成功を返さずに必要なコピー数の過半数がコピー完了してから成功を返します。

図2.2 TiKVのアーキテクチャ



注5 <https://docs.pingcap.com/tidb/stable/rocksdb-overview>

リージョンは次の条件のどちらかを満たすと2つ以上のリージョンに分割されます。リージョンのサイズはデフォルトで256MiBになります。デフォルトのサイズは設定で変更可能ですが、1GiBを超えるサイズは推奨されません。

- リージョンのサイズがしきい値を超える(デフォルトはリージョンのサイズの1.5倍)
- リージョン内のキーの数がしきい値を超える(デフォルトは2560000)

また、リージョンは次の条件を両方満たしている時間が一定時間(デフォルトは1時間)続いた場合に1つのリージョンに結合されます。

- リージョンのサイズがしきい値を下回る(デフォルトは54MiB)
- リージョン内のキーの数がしきい値を下回る(デフォルトは540000)

リージョンをRocksDBに格納する際に、次の4種類のカラムファミリーにデータが格納されます。カラムファミリーはRocksDBの機能で、データを論理的に分割して管理するために使用されます。カラムファミリーの詳細はGitHubのWiki内のColumn Families^{注6}を参照してください。

- raft
- lock
- write
- default

raftはリージョンのメタデータを格納します。サイズはごくわずかで通常は気にする必要ありません。lockは悲観的トランザクションの悲観的ロックと分散トランザクションの事前書き込みロックを格納します。トランザクションがコミットされると削除されます。そのため、lockのデータが肥大している場合は待機中のトランザクションが増大しているということになります。writeは実際に書き込まれたデータおよびデータが属するトランザクションの開始タイムスタンプとコミットタイムスタンプが格納されます。データが255byte未満であればwriteに保存され、255byteを超えるデータであればdefaultに保存されます。

RocksDBのTitanプラグイン

TiDBのバージョン7.6.0よりRocksDBのプラグインであるTitanがデフォルトで有効になりました。Titanを有効にすると列のデータが大きい場合(JSONなど)に読み書きのパフォーマンスが向上します。一方でディスク容量はRocksDB単体の時よりも消費が大きくなります。また、範囲クエリのパフォーマンスがRocksDB単体の時よりも低下します。

min-blob-sizeの設定値を超えるデータはTitanを使用して保存され、それを下回るサイズのデータ

注6 <https://github.com/facebook/rocksdb/wiki/Column-Families>

はRocksDBのネイティブ形式で保存されます。min-blob-sizeのデフォルトはTiDBのバージョン7.6.0より前のバージョンでは1KiB、7.6.0以降のバージョンでは32KiBです。

TiDBのバージョン7.6.0より前のバージョンから7.6.0以降にバージョンアップした場合はTitanの有効／無効の設定(rocksd.titan.enabled)とmin-blob-sizeの設定はそのまま変わらず引き継がれます。

RocksDBのTitanプラグインについての詳細はドキュメントのTitan Overview^{注7}を参照してください。

TiFlash

TiFlash^{注8}はTiDBにおけるOLAP(Online Analytical Processing)クエリを高速化するためのコンポーネントです。データの構造がTiKVとは異なり、TiKVがキーバリュー形式であるのに対してTiFlashは列指向のデータ形式です。列指向のデータ形式は大規模なデータのスキャンに適しており、集計や分析クエリのパフォーマンスが向上します。

TiFlashコンポーネントはTiDB構築時にはオプション扱いとなっていますので、TiFlashを使う場合は有効化してTiDBを構築してください。既存のTiDBに対して後からTiFlashコンポーネントを有効化、無効化することももちろん可能です。TiFlashを有効にする方法は3-4節または4-3節で説明します。

TiFlashに格納されるリージョンはTiKVノードからコピーされます。コピーする際には同様にRaftコンセンサスアルゴリズムが使用されますが、TiKVの時とは違いリーダー候補にならず、書き込み成功の判断に参加しないノードとしてRaftグループに参加します(ラーナー)。そのためTiFlash側へのデータの同期は非同期で行われます。TiDBへの書き込み後にTiFlash側にデータが反映される際に時間がかかる場合がありますので注意してください。

TiFlashを有効化しただけではTiFlashへデータはコピーされません。TiFlashへのデータコピーを有効化するためには、テーブルまたはデータベースに対してALTERステートメントを実行してTiFlashのレプリカ数を1以上に指定する必要があります。TiFlashへのデータコピーを無効化する場合はレプリカ数を0に指定します。

TiFlashへのデータコピーを有効化する

```
mysql> ALTER TABLE sample SET TIFLASH REPLICAS 1;  
mysql> ALTER DATABASE test SET TIFLASH REPLICAS 1;
```

TiFlashへのデータコピーを無効化する

```
mysql> ALTER TABLE sample SET TIFLASH REPLICAS 0;  
mysql> ALTER DATABASE test SET TIFLASH REPLICAS 0;
```

TiFlashへのデータ同期の状況はINFORMATION_SCHEMAのTIFLASH_REPLICAより取得できます。AVAILABLEが1であれば同期が完了しており、TiFlashを利用できます。完了していない場合は0が表示

注7 <https://docs.pingcap.com/tidb/stable/titan-overview>

注8 <https://github.com/pingcap/tiflash>

されます。PROGRESSは0から1の範囲の小数点でデータ同期の進捗状況を表示します。

TiFlashのデータ同期状況

```
mysql> SELECT TABLE_SCHEMA, TABLE_NAME, AVAILABLE, PROGRESS
-> FROM INFORMATION_SCHEMA.TIFLASH_REPLICA
-> WHERE TABLE_SCHEMA = 'test' and TABLE_NAME = 'sample';
```

TABLE_SCHEMA	TABLE_NAME	AVAILABLE	PROGRESS
test	sample	1	1

TiFlashで採用されている技術

column

TiFlash内部で使用されているランタイムやストレージインターフェースは、ClickHouse^{注a}がベースとなっています。ただし、TiFlashとClickHouseではフォーカスしているシナリオが全く異なり、TiFlashはトランザクションデータのリアルタイム分析に重点を置いているのに対し、ClickHouseは非トランザクションデータに重点を置いています。

注a <https://github.com/ClickHouse/ClickHouse>

▶ TiFlashを有効にした場合はディスク容量に注意

TiDBではデータはリージョン単位に分割され、分散配置される時にデフォルトでは3つにコピーされます。これは実データ量が保存時には3倍に膨れることを意味します。データはストレージクラスターに格納される時に圧縮されるため、純粹に3倍になるわけではないのですがTiDBを利用する際には考慮しておくポイントです。さらにTiFlashコンポーネントを有効にした場合、データはTiKVとは別に追加でコピーされることになります。TiFlashのレプリカ数を増やすと、それだけ追加でディスク容量が必要になるので注意してください。

■ プッシュダウンとコプロセッサー

TiDBではクエリの処理を効率化するために、特定の関数や演算子の計算処理をストレージクラスター側にプッシュダウンします。ストレージクラスターでは、コプロセッサーと呼ばれる機能を使ってプッシュダウンされた計算処理を行い、その結果をTiDBサーバーに返却します。

プッシュダウンされる関数や演算子は、たとえばMIN()やMAX()、COUNT()などの関数や、ANDやORなどの演算子です。プッシュダウンに対応している関数や演算子の一覧はドキュメントを参照してください。TiKVの場合はList of Expressions for Pushdown^{注9}、TiFlashの場合はPush-down Calculations

注9 <https://docs.pingcap.com/tidb/stable/expressions-pushed-down>

Supported by TiFlash^{注10}を参照してください。

▶ TiKVのコプロセッサ

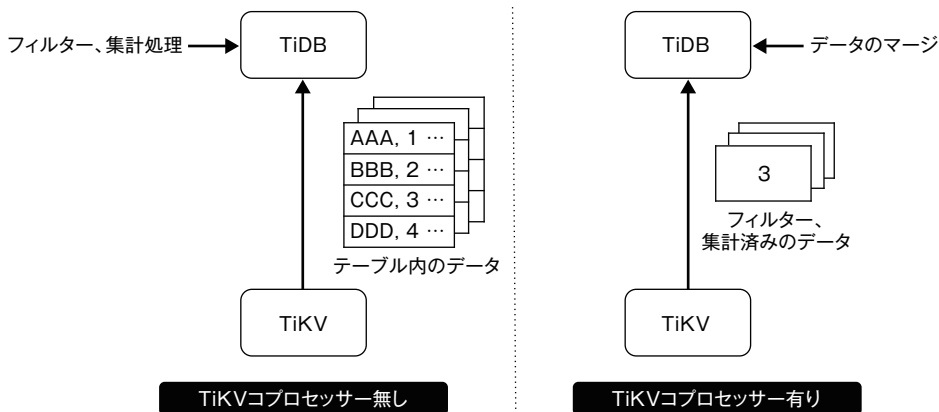
本項ではTiKVを例にコプロセッサについて説明します。たとえば次のような集計のSQLを実行した場合を考えます。

```
SELECT COUNT(ID) FROM t WHERE ID > 100;
```

コプロセッサを使用しない場合、TiDBサーバーはTiKVからすべてのデータを取得したあとに自身でフィルタリングと集計をしなければなりません。コプロセッサを使用した場合、TiDBサーバーはTiKV側でフィルタリングと集計をしたデータの結果のみを受け取ります。このようにすることでTiDBサーバー側では結果をマージするだけで済むため、TiDBサーバーの負荷を軽減できます。また、TiDBサーバーとTiKVとの間のデータトラフィック量も削減できます。

図2.3はTiKVのコプロセッサを使用しない場合(左)と、使用する場合(右)の違いを示しています。

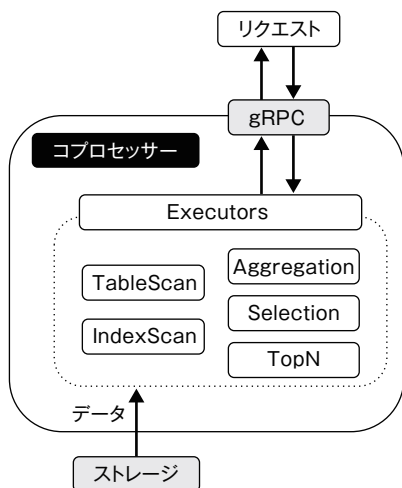
図2.3 TiKVのコプロセッサ



コプロセッサの内部にはExecutorがあり、それぞれが役割を持っています(図2.4)。それぞれのExecutorが連鎖するように処理を行い、最終的な結果を返します。

注10 <https://docs.pingcap.com/tidb/stable/tiflash-supported-pushdown-calculations>

図2.4 TiKVのコプロセッサ内部



各Executorの役割は次の通りです。

- **TableScan**
主キーを元にデータを取得します
- **IndexScan**
インデックスデータを取得します
- **Selection**
WHERE 句の条件に一致するデータを取得します
- **Aggregation**
COUNTやSUMなどの集計処理を行います
- **TopN**
ORDER BY xxxx LIMIT Nのような場合にデータをソートして上位N件を取得します

2-5 TiProxyコンポーネント

TiProxy^{注11}はTiDBクラスターに対する接続を管理し、ロードバランシングやプロキシ機能を提供するコンポーネントです。

なお、TiProxyコンポーネントはTiDB構築時にはオプション扱いです。TiProxyを有効にする方法は

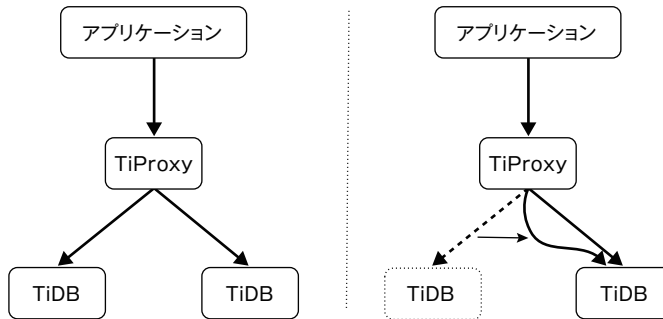
注11 <https://github.com/pingcap/tiproxy>

3-4節または4-3節で説明します。

TiProxyの仕組み

TiProxyはクライアントとTiDBクラスターの間でリバースプロキシとして動作します。TiProxyは利用可能なTiDBサーバーの情報をPDクラスターから取得して動的に管理しますので、クライアント側で再接続しなくてもTiDBサーバーへの接続を透過的に切り替えてくれます。各TiDBサーバーへの接続が偏らないように調整すると同時に、TiDBサーバーがスケールアウトした場合には新規TiDBサーバーに対して接続を割り振ります。TiDBサーバーのスケールイン時には縮退対象のTiDBサーバーへの接続の割り振りを停止し、既存の接続を他のTiDBサーバーへ切り替えます(図2.5)。

図2.5 TiProxyの動作



効果的にTiProxyを使用するために必要なこと

効果的にTiProxyを使用するためには、いくつか考慮しておくポイントがあります。

▶ graceful-wait-before-shutdownの設定

TiDBクラスターの設定項目graceful-wait-before-shutdownの値がデフォルトでは0になっており、TiDBサーバーを停止する前の猶予時間がありません。そのため、graceful-wait-before-shutdownを適切に設定しておくことが重要です。停止するまでの猶予時間を設定することで、TiProxyが接続を振り分けるための時間を確保できます。

適切な値を知るために、ダッシュボード機能を有効にしてトランザクションにかかっている時間を把握しておくことも大切です。最長のトランザクション時間より短い値を設定してしまうと、クライアントが切斷されてしまう可能性があります。ダッシュボード機能については3-5節を参照してください。

graceful-wait-before-shutdownはTiProxyにも設定可能です。TiProxyの場合もデフォルトは0にな

ています。TiProxyの前端にロードバランサーを使用する場合、適切な値に設定してください。

▶ systemdのTimeoutStopSecの設定

graceful-wait-before-shutdownの値を設定したとしても、場合によっては想定した挙動にならない場合もあります。たとえばsystemd経由でプロセスを起動していた場合は、デフォルトでは停止時に90秒間のタイムアウトが設定されます。つまりgraceful-wait-before-shutdownを90秒以上の長さに設定したとしても、systemdによって90秒後に強制停止されてしまいます。

これを避けるためにsystemdの設定でTimeoutStopSec^{注12}を適切な値に設定してください。

▶ TiUP操作時の待機時間の設定

TiUPでの操作時にも注意が必要です。デフォルトでは操作プロセスの各ステップの最大待機時間は120秒に設定されています。この最大待機時間を超えると操作は失敗として扱われます。最大待機時間は--wait-timeoutオプションで変更可能ですので、適切な値を指定してTiUPを実行してください。

▶ Kubernetes環境におけるterminationGracePeriodSecondsの設定

Kubernetes環境でTiProxyを起動している場合はterminationGracePeriodSeconds^{注13}のデフォルトが30秒となり、30秒を超えると強制的にプロセスが終了します。こちらも環境に合わせて適切な値にすることが重要です。

▶ 自己証明書の作成

透過的な再接続を実現するためには、TiDBサーバーとTiProxy間の通信をTLSで暗号化する必要があります。TiUPのバージョン1.16.0でTiDBサーバーとTiProxy間の通信を暗号化するための証明書が自動生成されるようになりました。そのため、バージョン1.16.0以降のTiUPを使用して構築することをおすすめします。手動で証明書を作成して配布する手順はドキュメントのGenerate Self-Signed Certificates^{注14}を参照してください。

▶ TiDB全体のバージョンアップ時にTiProxyを除外しておく

TiDBのバージョンアップを実行する際にTiProxyもアップグレードされてしまうと、クライアント側で予期せぬ切断が起こる可能性があります。TiProxyのバージョンを指定しておき、TiProxyのバージョンアップを回避しておくとういでしょう。TiProxyのバージョン体系はTiDBとは別に管理されています。TiProxyのバージョン設定はTiUPに読み込ませるトポロジーファイル内のcomponent_

注12 <https://www.freedesktop.org/software/systemd/man/latest/systemd.service.html#TimeoutStopSec=>

注13 <https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#lifecycle>

注14 <https://docs.pingcap.com/tidb/stable/generate-self-signed-certificates>

versions^{注15}で行います。TiUPによるTiDBの操作は第3章で説明します。

TiProxyのバージョン固定の設定例

```
component_versions:  
  tiproxy: "v1.3.0"
```

2

TiProxyのパフォーマンス

TiProxyのパフォーマンスについてHAProxy^{注16}との比較情報が公開されています^{注17}。sysbench^{注18}の基本ワークロードのシナリオではHAProxyに比べるとQPSが25%程度低下すると記載されています。パフォーマンスは環境やワークロードによって大きく変わってくるため、可能であれば実際の利用環境に即したワークロードのシナリオでパフォーマンスの確認をしておくとい良いでしょう。

TiProxyの制約事項

TiProxyにはさまざまな制約事項^{注19}があります。TiProxyを利用する際には制約事項をよく確認してください。

2-6

まとめ

本章ではTiDBを構成しているコンポーネントについて説明しました。各コンポーネントはマイクロサービスのように動作し、それぞれがスケールしやすい構成になっています。モノリシックな構成のMySQLと比較すると、TiDBの運用は少々複雑に感じるかもしれませんが、ですがTiDBは運用時に役立つダッシュボードが備わっており、またモニタリング環境のセットアップが容易であるという点で、複雑になりがちな運用をカバーできる仕組みが備わっています。

次章からは実際にTiDBを構築する方法を学んでいきます。

注15 https://docs.pingcap.com/tidb/stable/tiup-cluster-topology-reference#component_versions

注16 <https://www.haproxy.org/>

注17 <https://docs.pingcap.com/tidb/stable/tiproxy-performance-test>

注18 <https://github.com/akopytov/sysbench>

注19 <https://docs.pingcap.com/tidb/stable/tiproxy-overview#limitations>

第 3 章

仮想／物理サーバー へのTiDBの構築

TiUPを使った構築と操作

TiUPはTiDBを管理するためのコマンドラインツールです(タイアップと読まれることが多いです)。このツールはTiDBの構築、管理、スケーリング、バージョンアップといった操作を簡素化し、効率的に行うことを可能にします。TiDBの運用には複数のコンポーネントが関わるため、それらを手動で管理するのは時間や労力がかかりますが、TiUPを使用することでこれらのプロセスを統一的に管理できます。

TiUPは物理サーバーまたは仮想サーバー環境におけるTiDB Self-ManagedおよびTiDB Communityの構築、管理をサポートしています。Kubernetes環境におけるTiDBの構築、管理はTiUPではなくTiDB Operatorを使用します。Kubernetes環境でのTiDB構築については第4章で説明します。また、TiDB CloudはTiUPで構築、管理できません。TiDB Cloudの場合はTiDB Cloud CLIが提供されていますので、そちらを使用してください。TiDB CloudおよびTiDB Cloud CLIについては第5章で説明します。

物理サーバーまたは仮想サーバー環境を準備する際は、TiDBがサポートしているOSおよびCPUアーキテクチャーを確認してください。ドキュメントのTiDB Software and Hardware Requirements^{注1}から確認できます。

本章ではTiUPのインストール方法をはじめ、簡易的な構成でTiDBを起動できるTiDB Playgroundの使い方、複数サーバーを使用したTiDBの構築方法を説明します。

本章で使用するTiDBのバージョンは執筆時点における最新のバージョンである8.5.0を使用します。TiUPは1.16.1を使用します。また、サーバーの動作確認はx86_64環境のRocky Linux 9^{注2}で行っています。

3-1 TiUPの基本

本節ではTiUPをインストールして、実行環境を整える手順を説明します。

TiUPをインストールする

TiUPを使用するために、最初にTiUPのインストールを行います。インストール用のスクリプト(<https://tiup-mirrors.pingcap.com/install.sh>)が用意されていますので、スクリプトを使用してインストールします。

注1 <https://docs.pingcap.com/tidb/stable/hardware-and-software-requirements>

注2 <https://rockylinux.org/>

TiUPのインストール

```
$ curl --proto 'https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/install.sh | sh
```

スクリプトを実行すると、デフォルトでは実行ユーザーのホームディレクトリ(たとえば/home/tidb/)内の.tiup/binディレクトリ以下にTiUPがインストールされます。TiUPを実行しやすくするため、TiUPの実行PATH(たとえば/home/tidb/.tiup/bin)を環境変数PATHに追加しておくことをおすすめします。

先ほど実行したinstall.shは、実行シェルの設定ファイルに対して自動的にPATHを追加します。インストール時にBashを使用していた場合は、ホームディレクトリ内の.bash_profileファイルにPATHの追加が行われていることを確認してください。設定ファイルを読み直すか、ログインし直すことでPATHの設定が反映されます。

~/.bash_profileに追加されたPATH

```
...  
(省略)  
...  
export PATH=/home/tidb/.tiup/bin:$PATH
```

設定ファイルの読み直し

```
$ source ~/.bash_profile
```

インストール時の実行シェルと違うシェルを使用する場合は、使用するシェルの設定ファイルで同様のPATHの追加をしてください。Z shellの場合は次のようになります。

Z shellの設定ファイルにPATHを追加

```
$ echo 'export PATH=/home/tidb/.tiup/bin:$PATH' >> ~/.zshrc
```

TiUPの実体はtiupコマンドになります。tiupコマンドが実行できることを確認します。

TiUPの実行確認

```
$ tiup --version  
1.16.1 tiup  
Go Version: go1.21.13  
Git Ref: v1.16.1  
GitHash: 775384c48a95f7047c016cbf2905eec1e267f629
```

インストール手順で使用したコマンドの説明

column

インストール手順で使用したコマンドはドキュメントのTiUP Overview^{注a}に記載されているコマンドになります。

```
$ curl --proto 'https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/install.sh | sh
```

curl コマンドはURL からデータを取得するコマンドです。--proto オプションでプロトコルを指定し、--tlsv1.2 オプションでTLS1.2を使用するように指定しています。

-sSf オプションは-s オプション、-S オプション、-f オプションを一度にまとめて指定しています。-s オプションは--silent オプションの短縮形で、URL からデータを取得する際の経過情報を表示しないようにするためのオプションです。-S オプションは--show-error オプションの短縮形で、エラーが発生した場合にエラーメッセージを表示するためのオプションです。-f オプションは--fail オプションの短縮形で、HTTPのエラー(ステータス404など)が発生した場合にHTMLを出力せずに終了するためのオプションです。

| はパイプと呼ばれる記号で、左側のコマンドの標準出力を右側のコマンドの標準入力に渡すための記号です。curl コマンドは取得したデータをデフォルトでは標準出力に表示しますので、取得したデータをshコマンドに標準入力に渡して実行しています。

curl コマンドは必要に応じてインストールしてください。Rocky Linux 9の場合は次のようにインストールします。

curlコマンドのインストール

```
$ sudo dnf install curl
```

注a <https://docs.pingcap.com/tidb/stable/tiup-overview>

■ シェルの補完機能を使用する

TiUPは各シェルの補完機能に対応しています。補完機能を有効にすることで、**[Tab]**キーを使用したさまざまな補完が可能になり、TiUPを使用する時に便利になります。補完機能を有効にするために必要な設定はtiup completion <対象シェル名>コマンドで出力されますので、その内容を読み込ませて有効化します。

次の例はBashの補完機能を有効にする手順です。

Bashの補完機能の設定:都度有効にする場合

```
$ source <(tiup completion bash)
```

補完機能を常時有効にしておく場合は/etc/bash_completion.d/tiupを作成してください。次回以降にBashを起動した際は自動的に有効になります。

Bashの補完機能の設定:常時有効にしておく場合

```
$ tiup completion bash | sudo tee -a /etc/bash_completion.d/tiup
```


シェルの補完機能を有効にする場合は、追加パッケージのインストールが必要になる場合があります(Rocky Linux 9の場合はbash-completionパッケージが必要です)。必要に応じてインストールしてください。

tiup completion コマンドはBash以外にZ shell、fish、PowerShellに対応しています。各シェルにおける設定の仕方は--help オプションを実行して確認してください。

各シェルの補完機能のヘルプ表示

```
$ tiup completion zsh --help
$ tiup completion fish --help
$ tiup completion powershell --help
```

TiUPをバージョンアップする

TiUPはTiUP自身のバージョンアップも可能です。tiup update --self コマンドを実行して最新のTiUPを使用できます。

1.16.0から1.16.1にバージョンアップ

```
$ tiup --version
1.16.0 tiup
Go Version: go1.21.11
Git Ref: v1.16.0
GitHash: e543145831d44a863366ff8c379d25980730bbd1

$ tiup update --self
Updated successfully!

$ tiup --version
1.16.1 tiup
Go Version: go1.21.13
Git Ref: v1.16.1
GitHash: 775384c48a95f7047c016cbf2905eec1e267f629
```

「Updated successfully!」が表示されればバージョンアップ成功です。

使用するTiUPのバージョンを管理する

TiUPをインストールする際にスクリプト(<https://tiup-mirrors.pingcap.com/install.sh>)を使用すると、自動的に最新のTiUPがインストールされます。通常であれば最新版で問題無いのですが、最新よりも古いバージョンを使用したい場合はtiup install コマンドで古いバージョンをインストールした後にtiup link コマンドを使用してください。

バージョン1.15.2のTiUPを使用したい場合の実行例になります。

v1.15.2をインストール後に切り替える手順

```
$ tiup install tiup:v1.15.2

$ tiup link tiup:v1.15.2

$ tiup --version
1.15.2 tiup
Go Version: go1.21.10
Git Ref: v1.15.2
GitHash: cbcc2ce967981221380a7307bf7843fec2ea5283
```

tiup install コマンドは実行環境のディレクトリ (/home/<ユーザー>/.tiup/components/) 以下にバイナリーをダウンロードしてバージョンごとに保存します。tiup link コマンドは実行バイナリーをダウンロードされたバイナリーに切り替えるコマンドです。現在どのTiUPのバージョンがインストールされているかは tiup list tiup --installed コマンドで確認できます。なお、tiup install コマンドを使用してインストールしている場合のみ Installed が YES になります。install.sh スクリプトを使用してインストールされたTiUPはYESとして表示されません。

インストールされているバージョン一覧

```
$ tiup list tiup --installed
Available versions for tiup:
Version  Installed  Release           Platforms
-----  -
v1.15.2  YES        2024-05-28T10:58:47Z darwin/amd64,darwin/arm64,linux/amd64,linux/arm64
v1.16.0  YES        2024-07-02T08:59:33Z darwin/amd64,darwin/arm64,linux/amd64,linux/arm64
v1.16.1  YES        2024-10-31T05:59:12Z darwin/amd64,darwin/arm64,linux/amd64,linux/arm64
```

--installed オプションを省略した場合はダウンロード可能なバージョンが表示され、すでにインストールされているバージョンは Installed の部分に YES も表示されます。

ダウンロード可能なバージョン一覧

```
$ tiup list tiup
Available versions for tiup:
Version          Installed  Release           Platforms
-----  -
...
(省略)
...
v1.15.0          YES        2024-04-01T03:38:27Z darwin/amd64, ...
v1.15.1          YES        2024-04-11T11:07:53Z darwin/amd64, ...
v1.15.2          YES        2024-05-28T10:58:47Z darwin/amd64, ...
v1.16.0          YES        2024-07-02T08:59:33Z darwin/amd64, ...
v1.16.1-nightly-6  YES        2024-11-12T07:24:35Z darwin/amd64, ...
v1.16.1          YES        2024-10-31T05:59:12Z darwin/amd64, ...
```

必要なくなったバージョンは tiup uninstall コマンドで削除できます。

不要なバージョンの削除

```
$ tiup uninstall tiup:v1.16.0
/home/tidb/.tiup/components/tiup/v1.16.0
Uninstalled component `tiup:v1.16.0` successfully!
```

tiup list コマンドは他のコンポーネントのインストール状況も確認できます。

PDコンポーネントのインストール状況の例

```
$ tiup list pd --installed
Available versions for pd:
Version  Installed  Release                Platforms
-----  -
v8.5.0   YES           2024-12-19T06:21:30Z  darwin/amd64, ...
```

TiUPのデータ保存先を変更する

TiUPはTiDBの各種コンポーネントのさまざまなバージョンをダウンロードしたり、後に説明するTiDB Playgroundを起動したりしますが、デフォルトでは実行ユーザーの/home/<ユーザー>/.tiupディレクトリが使用されます。/homeが十分なディスク容量を確保できないといった場合は、別の場所に変更することも検討してください。TiUPが使用するディレクトリはTIUP_HOMEの環境変数で変更できます。

TiUPのインストール時にTIUP_HOMEを指定しておきます。ここでは例として/srv/data/tiupを作業ディレクトリとして設定します。

TiUPのインストール先を変更

```
$ curl --proto 'https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/install.sh | \
  TIUP_HOME="/srv/data/tiup" sh
```

その後のTiUPの操作の際にはTIUP_HOMEを設定することを忘れないようにしてください。指定しない場合はデフォルトの/home/<ユーザー>/.tiupディレクトリが使用されます。

TiUPが使用するディレクトリを変更して操作

```
$ TIUP_HOME=/srv/data/tiup /srv/data/tiup/bin/tiup playground
```

もしくはシェルの設定ファイルでTIUP_HOMEを設定しておくとい良いでしょう。Bashの場合はホームディレクトリ内の.bash_profileファイルに設定します。

設定ファイルにTIUP_HOMEを追加

```
$ echo 'export TIUP_HOME=/srv/data/tiup' >> ~/.bash_profile
```

設定後にシェルを再読み込みするか、ログインし直すことでTIUP_HOMEの設定が反映されます。

設定ファイルの読み直し

```
$ source ~/.bash_profile
```

TiUPが使用する環境変数を確認する

TiUPが使用する環境変数はTIUP_HOMEの他にもいくつかあります。tiup env コマンドはTiUPが使用する環境変数がどのように設定されているかを確認できます。

TiUPが使用する環境変数の表示

```
$ tiup env
TIUP_HOME=""
TIUP_SSHPASS_PROMPT=""
TIUP_SSH_PATH=""
TIUP_SCP_PATH=""
TIUP_KEEP_SOURCE_TARGET=""
TIUP_MIRROR_SYNC_SCRIPT=""
TIUP_LOG_PATH=""
TIUP_CLUSTER_DEBUG=""
```

各環境変数についての説明を表3.1に示します。

表3.1 TiUPが使用する環境変数の一覧

環境変数	説明
TIUP_HOME	TiUPが使用するディレクトリ
TIUP_SSHPASS_PROMPT	sshpasコマンドに認識させるプロンプト文字列
TIUP_SSH_PATH	sshコマンドのパス
TIUP_SCP_PATH	scpコマンドのパス
TIUP_KEEP_SOURCE_TARGET	ダウンロードしたソースファイルを保持するかどうか
TIUP_MIRROR_SYNC_SCRIPT	ミラーサーバーへ同期する際のスクリプト
TIUP_LOG_PATH	デバッグログの保存先
TIUP_CLUSTER_DEBUG	クラスター操作時のデバッグログの有効化

3-2

TiDB Playgroundの使い方

TiUPにはTiDB Playgroundを起動する機能があります。TiDB Playgroundは簡易的な構成のTiDBを起動する機能です。TiDBをインストールまたはバージョンアップする前にTiDB Playgroundで新機能の動作を確認したり、ちょっとした作業をしたりする際に役に立つ機能です。

TiDB Playgroundを起動する

TiDB Playgroundを起動する場合はtiup playground コマンドを実行します。初回実行時は必要なコンポーネントを実行環境にダウンロードしてくるため、起動まで時間がかかる場合があります。起動

に成功すると「TiDB Playground Cluster is started, enjoy!」のメッセージが表示されます。

```
TiDB Playgroundの実行
$ tiup playground
...
(省略)
...
🔥 TiDB Playground Cluster is started, enjoy!

Connect TiDB:  mysql --comments --host 127.0.0.1 --port 4000 -u root
TiDB Dashboard: http://127.0.0.1:2379/dashboard
Grafana:        http://127.0.0.1:3000
```

「Connect TiDB:」に表示されているコマンドを使用して、TiDB PlaygroundにMySQLプロトコルで接続できます。

```
mysqlコマンドを使用した接続
$ mysql --comments --host 127.0.0.1 --port 4000 -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1010827270
Server version: 8.0.11-TiDB-v8.5.0 TiDB Server (Apache License 2.0) ...

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

その後のTiDBの操作はMySQLと同様に行えます。SELECT VERSION() ステートメントを実行してバージョンを確認してみましょう。

```
バージョンの確認
mysql> SELECT VERSION();
+-----+
| VERSION() |
+-----+
| 8.0.11-TiDB-v8.5.0 |
+-----+
1 row in set (0.00 sec)
```

SELECT VERSION() ステートメントによって表示されるバージョン文字列は、先頭の8.0.11はMySQL 8.0の最初のGAバージョンで、これがTiDBがMySQL 8.0互換であることを示しており、その後のTiDB-v8.5.0はTiDBとしてのバージョンを示しています。

データベースの作成やテーブルの作成もMySQLの操作と同様に行うことができます。

MySQLと同じように操作

```
mysql> CREATE DATABASE testdb;
Query OK, 0 rows affected (0.07 sec)

mysql> USE testdb;
Database changed

mysql> CREATE TABLE tidb_sample (i int);
Query OK, 0 rows affected (0.07 sec)

mysql> DESCRIBE tidb_sample;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key  | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| i     | int  | YES  |      | NULL    |       |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.01 sec)
```

mysql コマンドを使用する際は別途インストール作業が必要になる場合がありますが、mysql コマンドを使用できない環境の場合でも tiup client コマンドを使用して TiDB に接続できます。tiup client コマンドを実行すると「Choose an endpoint to connect」と表示されますので、**[Enter]** キーで選択します。エンドポイントと同時に表示されている行頭の文字列(次の例では UU8i5cc)は TiDB Playground で起動している TiDB のタグ(識別子)になります。TiDB Playground を実行する度に新しいタグが割り当てられますが、固定も可能です。タグの使い方は後述します。

tiup clientを使用した接続

```
$ tiup client
┌─ Choose a endpoint to connect ─┐
│ UU8i5cc mysql://root@127.0.0.1:4000 │
└────────────────────────────────┘

(実際は画面が切り替わり選択画面になります)

Starting component client: ...
Connected with driver mysql (8.0.11-TiDB-v8.5.0)
Type "help" for help.

my:root@127.0.0.1:4000=>
```

接続後はmysql コマンドと同様に操作できます。

mysqlコマンドと同じように操作可能

```
my:root@127.0.0.1:4000=> SHOW DATABASES;
Database
```

```

-----
INFORMATION_SCHEMA
METRICS_SCHEMA
PERFORMANCE_SCHEMA
mysql
sys
test
(6 rows)

```

起動するTiDBバージョンを指定する

tiup playground コマンドを実行すると、デフォルトではDMRバージョンとLTSバージョンを含むリリースの中から最新のバージョンのTiDBが起動されます。執筆時点では8.5.0が最新バージョンとして使用されています。この8.5.0はLTSバージョンになりますが、1つ前のLTSバージョンである8.1.2を使用したい場合もあるかもしれません。TiDB Playgroundでは特定のバージョンを指定してTiDBを起動できます。8.1.2で起動する例を示します(指定するバージョン文字列の最初のvは省略可)。

v8.1.2を指定して起動

```
$ tiup playground v8.1.2
```

起動が終わったら接続してバージョンを確認します。

v8.1.2で起動していることを確認

```

$ mysql --comments --host 127.0.0.1 --port 4000 -u root -e "SELECT VERSION();"
+-----+
| VERSION() |
+-----+
| 8.0.11-TiDB-v8.1.2 |
+-----+

```

tiup playground コマンド実行時に指定できるバージョン表記は、他に^8のような指定の仕方やnightlyを指定できます。^8であればメジャーバージョン8の最新が使用されます。nightlyを指定した場合は毎日ビルドされている最新の開発バージョンが使用されます。

バージョン8系の最新を使用する

```
$ tiup playground ^8
```

^7を指定した場合はメジャーバージョン7の最新が使用されます。

バージョン7系の最新を使用する

```
$ tiup playground ^7
```

執筆時点では7.6.0が使用されました。

バージョン7系の最新バージョンを確認

```
$ mysql --comments --host 127.0.0.1 --port 4000 -u root -e "SELECT VERSION()"
+-----+
| VERSION() |
+-----+
| 8.0.11-TiDB-v7.6.0 |
+-----+
```

タグを指定してTiDB Playgroundを起動する

TiDB Playgroundは複数のTiDBを同時に起動できます。その際に役に立つのがタグ(識別子)機能です。タグはtiup playgroundコマンド実行時に--tagオプションを使用することで設定できます。--tagオプションを使用しなかった場合は自動生成されたタグが割り当たります。

sample1というタグで1つ目のTiDB Playgroundを起動します。

タグにsample1を使用してTiDB Playgroundを起動する

```
$ tiup playground --tag sample1
🔥 TiDB Playground Cluster is started, enjoy!

Connect TiDB:  mysql --comments --host 127.0.0.1 --port 4000 -u root
TiDB Dashboard: http://127.0.0.1:2379/dashboard
Grafana:        http://127.0.0.1:3000
```

続いて2つ目をsample2として起動します。tiup playgroundコマンドは別端末から実行してください。

タグにsample2を使用してTiDB Playgroundを起動する

```
$ tiup playground --tag sample2
🔥 TiDB Playground Cluster is started, enjoy!

Connect TiDB:  mysql --comments --host 127.0.0.1 --port 45323 -u root
TiDB Dashboard: http://127.0.0.1:38647/dashboard
Grafana:        http://127.0.0.1:40997
```

sample2を指定して起動したTiDBはsample1とは違うポート番号が使用されて起動されていることが確認できます。

それぞれの状態を確認するためにtiup playground displayコマンドを実行します。tiup playground displayコマンドはTiDB Playgroundによって起動されているプロセスのプロセスIDと起動時間を表示するコマンドです。tiup playground displayコマンド実行時に--tagオプションを指定して対象を指定できます。

sample1の状態を確認します。

sample1の状態

```
$ tiup playground display --tag sample1
Pid  Role    Uptime
---  ---
1620 pd      9m0.93918208s
1621 tikv    9m0.908315032s
1623 tidb   9m0.877071065s
1845 tiflash 8m51.710440612s
```

続いてsample2の状態を確認します。

sample2の状態

```
$ tiup playground display --tag sample2
Pid  Role    Uptime
---  ---
4308 pd      4m0.50737717s
4314 tikv    4m0.441407518s
4335 tidb   4m0.378260284s
4356 tiflash 4m0.310578612s
```

sample2はプロセスIDがsample1とは違うことが確認できます。

このようにタグを使用することで、同一環境に複数のTiDB Playgroundを起動できます。スケールアウトなどの各種オペレーションもタグを指定して対象を選択可能です。スケールアウトについての詳細は後述します。

sample2のTiDBサーバーをスケールアウト

```
$ tiup playground scale-out --db 1 --tag sample2
To connect new added TiDB: mysql --comments --host 127.0.0.1 --port 33717 ...

$ tiup playground display --tag sample2
Pid  Role    Uptime
---  ---
4308 pd      4m40.370020329s
4314 tikv    4m40.318741003s
4335 tidb   4m40.237922481s
6451 tidb   1m11.39228297s
4356 tiflash 4m40.170240809s
```

TiDB Playground 起動時に --tag オプションを使用しない場合は、自動生成された値がタグに付与されます。付与されているタグを確認したい場合はtiup statusコマンドを実行して確認します。Nameの部分がタグになります。

タグの確認

```
$ tiup status
Name      Component  PID    Status  Created Time  ...
```

```
UUA2ZHc playground 6518 RUNNING 2024-11-15T02:57:08Z ...
```

TiDB Playgroundを複数起動させる場合はCPUやメモリなどのリソースが十分に足りていることを確認してください。リソースが不足している場合は正常に起動しない場合があります。

▶ タグを利用したデータの永続化

複数のTiDB Playgroundを扱う際に便利なタグ機能ですが、それ以外にタグを使用した便利な使い方を1つ紹介しておきます。

タグを指定せずにTiDB Playgroundを起動した場合、データは終了時に削除されるため作業内容が失われます。つまり、タグを指定しないと毎回まっさらなデータベースで起動し、前回の作業内容は反映されていないことになります。そこで役に立つのがタグです。タグを指定してTiDB Playgroundを起動した場合には、終了時にデータは消えません。そのため、いつも使うデータを毎回起動時に読み込ませたいといったユースケースの場合は、タグを使用しておくことをおすすめします。保存されるデータはデータベースの内容だけでなく、各種コンポーネントのログや設定変更内容も保存されています。各種コンポーネントの設定変更は本節内の「特定の設定を読み込ませてTiDB Playgroundを起動する」を参照してください。

データが永続化されることを確認するために、公開されているMySQLのデータ^{注3}を使用した例を示します。タグはemployeesを指定します。まずTiDB Playground起動後にデータを読み込ませます。

```
employeeのtagを使用して起動後にデータを読み込む
$ tiup playground --tag employees

$ mysql --comments --host 127.0.0.1 --port 4000 -u root < employees.sql
INFO
CREATING DATABASE STRUCTURE
INFO
storage engine: InnoDB
INFO
LOADING departments
INFO
LOADING employees
INFO
LOADING dept_emp
INFO
LOADING dept_manager
INFO
LOADING titles
INFO
LOADING salaries
```

注3 <https://dev.mysql.com/doc/employee/en/employees-installation.html>

```
data_load_time_diff
NULL
```

投入されたデータの確認をします。

データの確認

```
$ mysql --comments --host 127.0.0.1 --port 4000 -u root -e "SELECT COUNT(*) FROM employees.employees"
+-----+
| COUNT(*) |
+-----+
|    300024 |
+-----+
```

TiDB Playgroundを起動している端末で **[Ctrl] + [C]** を押して停止します。tiup status コマンドで停止されていることを確認します。Status 部分が TERM になります。

停止の確認

```
$ tiup status
Name      Component  PID   Status  Created Time      ...
-----
employees playground 7852  TERM    2024-11-15T03:55:15Z ...
```

再度 employees のタグを使用して起動し、先ほど投入したデータが復元されているか確認します。

再度 employees のタグで起動する

```
$ tiup playground --tag employees
```

先ほど作成した employees データベースが読み込まれていることを確認します。

employees データベースが復元されていることを確認

```
$ mysql --comments --host 127.0.0.1 --port 4000 -u root \
-e "SHOW DATABASES"
+-----+
| Database |
+-----+
| INFORMATION_SCHEMA |
| METRICS_SCHEMA |
| PERFORMANCE_SCHEMA |
| employees |
| mysql |
| sys |
| test |
+-----+

$ mysql --comments --host 127.0.0.1 --port 4000 -u root \
-e "SELECT COUNT(*) FROM employees.employees"
+-----+
```

```
| COUNT(*) |  
+-----+  
| 300024 |  
+-----+
```

このように同じタグを使用し続けることでデータの永続化が可能になります。さまざまなデータセットを読み込ませたTiDB Playgroundをタグで管理しておくことで、いつでも好きなデータセットにアクセスできるようになります。ただし、明示的に削除しない限りデータは残り続けることになりますので、ディレクトリ容量を圧迫する要因になります。必要なくなったデータは定期的に削除することをおすすめします。データを削除する方法は本節内の「TiDB Playgroundのデータをクリーンアップする」を参照してください。

▶ タグを使用しても前回起動時に指定した構成情報は引き継がれない

同じタグを使用しても、前回起動した時の構成情報(PDサーバーの台数やTiFlashサーバーの台数など)は引き継がれません。たとえばPDサーバーを--pd 3オプションで3台起動させ、その後同じタグで起動する場合、--pd 3オプションを省略するとPDサーバーはデフォルトの1台で起動しようとしします。そのため、同じタグを使用して起動する場合は--pdオプションや--kvオプションなどはなるべく前回起動時と合わせるようにしてください。起動後にスケールアウトさせた場合は、次回起動時にはスケールアウトした台数を含めてオプションで指定してください。コンポーネントの台数が変わるとTiDB Playgroundの起動に失敗する場合があります。

自動生成されるタグはどのように生成されているのか

column

自動生成されるタグに使用されるアルゴリズムにはBase62エンコードが使用されています。Base62エンコードはデータを英数字62種(アルファベットの大文字と小文字26種ずつ、数字10種の合計62種)に変換する手法です。よく知られているのはBase64エンコードですが、Base62エンコードはBase64エンコードで使用される記号+ / と=を使用しない手法です。変換の元になるデータには現在時刻が使用されています。詳細はTiUPのソースコード内のBase62Tag()関数を参照^{注a}してください。

注a <https://github.com/pingcap/tiup/blob/v1.16.1/pkg/utlis/utlis.go#L54-L65>

■ 特定の設定を読み込ませてTiDB Playgroundを起動する

TiDB Playgroundを起動する際に特定の設定を変更してカスタマイズしたい場合は、変更したい設定を記述したコンフィグファイルを読み込ませてTiDB Playgroundを起動させることができます。例として次のTOMLファイルを読み込ませ、バージョン文字列の変更とログをJSON形式に変更させます。

```
custom_db.toml
```

```
server-version = "This is a TiDB Book"
```

```
[log]
```

```
format = "json"
```

```
custom_db.tomlを読み込ませて起動する
```

```
$ tiup playground --db.config custom_db.toml
```

起動したらバージョンを確認します。

```
バージョンを確認
```

```
$ mysql --comments --host 127.0.0.1 --port 4000 -u root -e "SELECT VERSION()"
```

```
+-----+
| VERSION() |
+-----+
| This is a TiDB Book |
+-----+
```

TiDB サーバーのログは /home/<ユーザー>/.tiup/data/<タグ>/tidb-<N>/tidb.log にあります。

```
ログがJSON形式になっていることを確認
```

```
$ tail /home/tidb/.tiup/data/UUBsnQb/tidb-0/tidb.log
```

```
...
```

```
(省略)
```

```
...
```

```
{ "level": "INFO", "time": "2024/11/15 10:42:45.544 +00:00", "caller": "gc_worker.go:1263", "message":
"sent safe point to PD", "category": "gc worker", "uuid": "64cbd817a7c0004", "safe point":
453946026313121792 }
{ "level": "WARN", "time": "2024/11/15 10:43:05.025 +00:00", "caller": "ddl.go:885", "message": "unknown
server version, might be changed directly in config", "category": "ddl", "sampled": "", "version":
"This is TiDB Book" }
```

バージョン文字列を変更した影響で WARN が表示されていますが、JSON 形式になっていることが確認できます。

主なコンポーネントの設定内容は表 3.2 のドキュメントを参照してください。また、一部の設定は TiDB Playground 起動時に上書きされますので注意してください。

表 3.2 主なコンポーネントの設定項目

コンポーネント	ドキュメント
TiDB	https://docs.pingcap.com/tidb/stable/tidb-configuration-file
TiKV	https://docs.pingcap.com/tidb/stable/tikv-configuration-file
PD	https://docs.pingcap.com/tidb/stable/pd-configuration-file
TiFlash	https://docs.pingcap.com/tidb/stable/tiflash-configuration
TiProxy	https://docs.pingcap.com/tidb/stable/tiproxy-configuration

TiDB Playgroundで起動するコンポーネントを調整する

TiDB Playgroundではデフォルトでさまざまなコンポーネントが起動しますが、コンポーネントごとに起動するプロセス数の変更や、モニタリング環境が必要ない場合はモニタリングコンポーネントを起動しないといった選択も可能です。

デフォルトではPDサーバー、TiKVサーバー、TiDBサーバー、TiFlashサーバーが1プロセスずつに加えて、モニタリング環境としてPrometheusとGrafanaとNgMonitoringの3つのプロセスが起動します。TiDB Playground起動後の各プロセスの状態は`tiup playground display`コマンドで確認できます。複数のTiDB Playgroundが起動している場合は`--tag`オプションでタグを指定することで対象を選択できます。`tiup playground display`コマンドではPrometheusやGrafanaのモニタリング環境のプロセスは表示されませんが、`ps`コマンドで確認できます。

playgroundのプロセスの確認

```
$ tiup playground display
Pid  Role    Uptime
---  ---
5070  pd      4m13.617888713s
5076  tikv    4m13.576933998s
5078  tidb    4m13.538100967s
5299  tiflash 4m5.835645618s
```

モニタリングプロセスの確認

```
$ ps -ef | grep ".tiup/components"
...
(省略)
...
tidb 5273 ... /home/tidb/.tiup/components/prometheus/v8.5.0/prometheus/prometheus
tidb 5274 ... /home/tidb/.tiup/components/prometheus/v8.5.0/ng-monitoring-server
tidb 5292 ... /home/tidb/.tiup/components/grafana/v8.5.0/bin/grafana-server
...
(省略)
...
```

モニタリングコンポーネントの起動の抑制

モニタリングコンポーネントが必要ない場合はTiDB Playground起動時に`--without-monitor`オプションを指定して起動を抑制できます。

モニタリング環境の起動を抑制してTiDB Playgroundを起動

```
$ tiup playground --without-monitor
🔥 TiDB Playground Cluster is started, enjoy!

Connect TiDB:  mysql --comments --host 127.0.0.1 --port 4000 -u root
TiDB Dashboard: http://127.0.0.1:2379/dashboard
```

起動後の表示出力から「Grafana:」の行が消えました。ps コマンドでも起動していないことが確認できます。TiDB Playgroundでは複数のコンポーネントが起動するので、実行サーバーのCPUやメモリのリソースを消費が多くなる場合があります。リソースの使用量を減らしたい場合には必要無いコンポーネントの起動は抑制しておいた方が良いでしょう。

▶ 各種コンポーネントのスケールアウトとスケールイン

TiDBサーバーやTiKVサーバーなどの起動数をデフォルトの1から増やしたい場合はtiup playground scale-out コマンドを実行します。今回はTiDBサーバーの台数を変更してみましょう。スケールアウトさせたいTiDBサーバーの台数を--db オプションで指定します。1台から2台に増やす場合は1台増やせば良いので--db 1を指定します。

TiDBサーバーを1台から2台に増やす

```
$ tiup playground scale-out --db 1
To connect new added TiDB: mysql --comments --host 127.0.0.1 --port 43543 ...
```

TiDBサーバーが追加で1台起動され、新しく43543番ポートで立ち上がりました。mysqlコマンドで新しいポート番号を指定し、接続後にserver_idを確認します。server_idを比較することで、最初に起動したTiDBサーバーと違うサーバーにつながっていることが確認できます。

新しく起動したTiDBサーバーに接続

```
$ mysql --host 127.0.0.1 --port 43543 -u root -e 'SHOW STATUS like "server_id"'
+-----+
| Variable_name | Value |
+-----+
| server_id     | 26d6f195-e24d-467a-a61a-96a029bf84b2 |
+-----+
```

最初に起動したTiDBサーバーに接続

```
$ mysql --host 127.0.0.1 --port 4000 -u root -e 'SHOW STATUS like "server_id"'
+-----+
| Variable_name | Value |
+-----+
| server_id     | 625c5a12-b64e-4f93-93c7-cb0d649d93df |
+-----+
```

tiup playground display コマンドでTiDBサーバーが増えていることを確認できます。起動時間(Uptime)が短い方が新しく起動したTiDBサーバーになります。

TiDBサーバーが2台に増えている様子

```
$ tiup playground display
Pid   Role   Uptime
---   ---   -
9926  pd     11m59.377663011s
9933  tikv   11m59.333093355s
```

```
9954  tidb  11m59.272022909s
11194 tidb  8.102740348s
10131 tiflash 11m52.22486585s
```

TiKVサーバーをスケールアウトさせる場合は--kvオプション、PDサーバーをスケールアウトさせる場合は--pdオプション、TiFlashサーバーをスケールアウトさせる場合は--tiflashオプションを使用します。tiup playground scale-out コマンドはその他のコンポーネントにも対応していますので、詳細は--help オプションを指定して確認してください。

スケールアウトした各種コンポーネントはスケールインさせることも可能です。スケールインする場合はtiup playground scale-in コマンドを実行します。tiup playground scale-in コマンドはスケールインさせたいプロセスのID(pid)を指定します。そのため、tiup playground display コマンドでプロセスIDを確認してから実行すると良いでしょう。スケールアウトの時とは少し実行の仕方が違いますので注意してください。

先ほどスケールアウトしたTiDBサーバーをスケールインしてみましょう。

```
pidを確認してスケールインする
$ tiup playground display
Pid  Role  Uptime
---  ----  -
9926  pd     11m59.377663011s
9933  tikv   11m59.333093355s
9954  tidb   11m59.272022909s
11194 tidb   8.102740348s
10131 tiflash 11m52.22486585s

$ tiup playground scale-in --pid 11194
scale in tidb success
```

TiDBサーバーが1台に戻っていることを確認します。

```
tidbが1つだけ表示されていることを確認
$ tiup playground display
Pid  Role  Uptime
---  ----  -
9926  pd     12m47.922585439s
9933  tikv   12m47.878015599s
9954  tidb   12m47.816944804s
10131 tiflash 12m40.769788763s
```

▶ 各種コンポーネントの起動数の指定

TiDB Playgroundを起動する際に、起動するサーバーの数を各種コンポーネントごとに指定できます。たとえばPDサーバーを最初から3台起動したり、TiFlashのサーバー数を0にして起動しないといったことが可能です。

PDサーバーを3台指定して起動する

```
$ tiup playground --pd 3

$ tiup playground display
Pid    Role    Uptime
---    -
11770  pd      2m17.797874409s
11777  pd      2m17.764563033s
11783  pd      2m17.694439198s
11790  tikv    2m17.576349381s
11796  tidb    2m17.491231882s
11978  tiflash 2m6.270386691s
```

PDサーバーを3台指定かつTiFlashサーバーを起動しない

```
$ tiup playground --pd 3 --tiflash 0

$ tiup playground display
Pid    Role    Uptime
---    -
13046  pd      1m19.25963493s
13054  pd      1m19.224259697s
13059  pd      1m19.156944021s
13070  tikv    1m19.061227934s
13092  tidb    1m18.977394593s
```

このようにTiDB Playgroundは起動時に各種コンポーネントの起動台数を指定できます。

TiDBの最小限のコンポーネントのみ起動する場合は下記のように指定すると良いでしょう。デフォルトで1台しか起動しないコンポーネントも明示的にオプションで指定しています。

最低限必要なコンポーネントで起動

```
$ tiup playground --pd 1 --kv 1 --db 1 --tiflash 0 --without-monitor

$ tiup playground display
Pid    Role    Uptime
---    -
13599  pd      1m35.871090421s
13606  tikv    1m35.823866722s
13627  tidb    1m35.763945731s
```

▶ TiProxyを使用した接続

TiDB PlaygroundはTiProxyを有効化することで、TiProxy経由でTiDBに接続できます。TiProxyを有効化する場合は--tiproxyオプションを使用して起動するTiProxyサーバーの数を指定します。デフォルトでは0になっており、TiProxyは起動しません。

TiProxyのバージョンを指定する場合は--tiproxy.versionオプションで行います。使用可能なバージョンはtiup list tiproxyコマンドで確認できます。

TiProxyの動作を確認してみましょう。TiProxyの動作を効果的にするため、TiDBサーバーに対してgraceful-wait-before-shutdownの設定をしておきます。また、ログの場所を分かりやすくするためにタグの設定もしておきます。

```
graceful_db.toml
graceful-wait-before-shutdown = 15
```

```
TiProxyサーバーを1台起動する
$ tiup playground --tiproxy 1 --db.config graceful_db.toml --tag tiproxy_demo
🔥 TiDB Playground Cluster is started, enjoy!

Connect TiDB:  mysql --comments --host 127.0.0.1 --port 4000 -u root
Connect TiProxy: mysql --comments --host 127.0.0.1 --port 6000 -u root
TiDB Dashboard: http://127.0.0.1:2379/dashboard
Grafana:        http://127.0.0.1:3000
```

TiProxyを有効にした場合は、TiDB Playground 起動後にTiProxy経由でのアクセス方法(Connect TiProxy行)が追加で表示されます。

さっそくTiProxy経由で接続してみましょう。接続する時にmysqlコマンドで--skip-reconnectオプションを有効にしておきます。--skip-reconnectオプションは接続が切れた際に自動的に接続を試みる挙動を抑制するオプションです。TiProxyの機能を確認しやすくするため、このオプションを有効にします。

```
TiProxyサーバー経由で接続
$ mysql --comments --host 127.0.0.1 --port 6000 -u root --skip-reconnect
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 8.0.11-TiDB-v8.5.0 TiDB Server (Apache License 2.0) ...

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SELECT VERSION();
+-----+
| VERSION() |
+-----+
| 8.0.11-TiDB-v8.5.0 |
+-----+
1 row in set (0.00 sec)
```

TiProxyの機能を確認しましょう。現在のTiProxyのバックエンドであるTiDBサーバーは1台です。まずserver_idを確認しておきます。

```
server_idを確認しておく
mysql> SHOW STATUS like "server_id";
+-----+-----+
| Variable_name | Value                               |
+-----+-----+
| server_id     | c1922687-2eda-4671-9ef6-ac1ba33beb3e |
+-----+-----+
1 row in set (0.04 sec)
```

別端末で同じように接続し、server_idを確認しておきます。TiDBは現在1台しかないため、2端末上の表示は当然同じserver_idになっています。続いてTiDBサーバーを1台スケールアウトします。

```
TiDBサーバーを1台スケールアウト
$ tiup playground scale-out --db 1 --tag tiproxy_demo
```

TiDBサーバーが1台増えたところで、TiProxyのログを確認します。

```
update backendのイベントを確認
$ grep "update backend" /home/tidb/.tiup/data/tiproxy_demo/tiproxy-0/tiproxy.log
[2024/11/16 06:57:35.851 +00:00] [INFO] [main.nsmgr.observer] [observer/backend_observer.go:200]
[update backend] [namespace=default] [backend_addr=127.0.0.1:4000] [prev="down, err: connect
status port failed: Get \"http://127.0.0.1:10080/status\": dial tcp 127.0.0.1:10080:
connect: connection refused"] [cur=healthy]
[2024/11/16 07:04:17.947 +00:00] [INFO] [main.nsmgr.observer] [observer/backend_observer.go:200]
[update backend] [namespace=default] [backend_addr=127.0.0.1:45611] [prev="down, err:
connect status port failed: Get \"http://127.0.0.1:43327/status\": dial tcp 127.0.0.1:43327:
connect: connection refused"] [cur=healthy]
```

最初の行はbackend_addr=127.0.0.1:4000と表示されており、1台目のTiDBサーバーのイベントであることが分かります。ログの内容はバックエンド(TiDBサーバー)の状態がprev=からcur=の状態に変わったことを表しており、cur=healthyと表示されていることから127.0.0.1:4000がバックエンドとして認識されたことが分かります。2行目ではbackend_addr=127.0.0.1:45611がcur=healthyとなっており、2台目のTiDBサーバーがバックエンドとして認識されたことを示しています。

バックエンドの状況はログ以外にもTiProxyのメトリクスでも確認できます。tiproxy_backend_b_statusを参照するとバックエンドが2台認識されていることが確認できます。

```
TiProxyのメトリクスを確認
$ curl -s http://127.0.0.1:3080/metrics/ | grep "tiproxy_backend_b_status"
# HELP tiproxy_backend_b_status Gauge of backend status.
# TYPE tiproxy_backend_b_status gauge
tiproxy_backend_b_status{backend="127.0.0.1:4000"} 1
tiproxy_backend_b_status{backend="127.0.0.1:45611"} 1
```

TiDB サーバーが無事に増えましたので、先ほどの端末で `server_id` を確認します。

再接続無しに `server_id` が変わることを確認

```
mysql> SHOW STATUS like "server_id";
```

Variable_name	Value
server_id	b779c351-f2fd-4324-8e20-6ca636df5144

```
1 row in set (0.02 sec)
```

再接続無しに片方の端末で `server_id` が変更されたことが確認できます。続いて TiDB サーバーをスケールインした際の動作を確認します。

スケールイン対象のTiDBはどちらでも問題ない

```
$ tiup playground display --tag tiproxy_demo
```

Pid	Role	Uptime
46020	pd	40m33.438769239s
46026	tikv	40m33.325709607s
46028	tidb	40m33.191174302s
46343	tidb	33m39.839238647s
46036	tiproxy	40m33.134137603s

```
$ tiup playground scale-in --tag tiproxy_demo --pid 46028
```

scale in tidb success

`graceful-wait-before-shutdown` で設定した秒数が経過し、TiDB サーバーのスケールインが終了すると TiDB Playground を実行している端末で `tidb quit` とメッセージが出力されます。その後 TiDB に接続している端末の両方の `server_id` を確認します。両方の端末で接続が切れること無く同じ `server_id` が表示されていることが確認できます。このように TiProxy は透過的にバックエンドの TiDB サーバーを切り替えることができ、TiDB の利便性を大きく向上させるコンポーネントです。

TiDB Playgroundのデータをクリーンアップする

TiDB Playground のデータは `/home/<ユーザー>/<tiup>/data/<tag>` ディレクトリに保存されます。タグを指定して起動しない場合は自動生成されたタグが割り振られ、TiDB Playground 終了時にデータはクリーンアップされます。タグを指定して起動した場合はデータは保存されるため、さまざまなタグを使用して起動するとデータディレクトリが次第に肥大化していきます。そのため、使用しなくなったデータは定期的に削除することをおすすめします。

データの削除は `tiup clean` コマンドで行います。削除したいデータを保持しているタグを `--tag` オプションで指定してください。

unused_playgroundのデータを削除

```
$ tiup playground --tag unused_playground

$ tiup clean unused_playground
Clean instance of `playground`, directory: /home/tidb/.tiup/data/unused_playground
```

現在どのタグのデータが存在しているのかはtiup statusコマンドで確認すると良いでしょう。

tiup statusでタグを確認

```
$ tiup status
```

Name	Component	PID	Status	...	Directory	...
sample1	playground	67913	TERM	...	/home/tidb/.tiup/data/sample1	...
sample2	playground	68097	TERM	...	/home/tidb/.tiup/data/sample2	...
sample3	playground	68182	TERM	...	/home/tidb/.tiup/data/sample3	...

保持しているデータを一括で削除したい場合はタグの指定を省略し、代わりに--allオプションを指定します。削除対象がすべてのタグとなり、/home/<ユーザー>/.tiup/data/ディレクトリ内が一掃されます。

すべてのタグのデータを削除

```
$ tiup clean --all
Clean instance of `playground`, directory: /home/tidb/.tiup/data/sample1
Clean instance of `playground`, directory: /home/tidb/.tiup/data/sample2
Clean instance of `playground`, directory: /home/tidb/.tiup/data/sample3
```

TiKVのPlaygroundを起動する

実はTiDBだけではなく、TiKVのPlaygroundを構築したい場合にもTiUPは便利です。TiDB Playground 起動時に--mode=tikv-slimオプションを指定することで、TiKVサーバーおよびPDサーバーのみ起動させることができます。TiKVの動作確認をしたい場合は利用を検討してください。

TiKVのPlaygroundを起動する

```
$ tiup playground --mode=tikv-slim

$ tiup playground display
```

Pid	Role	Uptime
26954	pd	32.27883827s
26961	tikv	32.224519691s

tiup playground --db 0コマンドでも同じような環境を作り出すことはできますが、その場合tiup playground displayコマンドでの確認がエラーになってしまう現象が執筆時に確認できました。TiKVの機能のみを使用したい場合は--mode=tikv-slimオプションを使用してください。

3-3

TiUPを使用した仮想サーバーまたは物理サーバーへのTiDBの構築

本節では複数のサーバーを対象にTiDBを構築する手順を紹介します。

サーバー環境を準備する

本節で使用するサーバー環境を表3.3に示します。

表3.3 サーバー環境

ホスト名	IPアドレス	用途
ope	10.0.0.1	TiUP実行
pd01	10.0.0.11	PDクラスター
pd02	10.0.0.12	PDクラスター
pd03	10.0.0.13	PDクラスター
tidb01	10.0.0.21	TiDBクラスター
tidb02	10.0.0.22	TiDBクラスター
tidb03	10.0.0.23	TiDBクラスター
tikv01	10.0.0.31	ストレージクラスター
tikv02	10.0.0.32	ストレージクラスター
tikv03	10.0.0.33	ストレージクラスター
monitor	10.0.0.41	ダッシュボード、モニタリング

作業用のユーザーとしてtidbを使用します。opeサーバーは他のサーバーに対してtidbユーザーでsshログインできるようにしておきます。これはTiUPはさまざまな作業をssh経由で実行するためです。また、tidbユーザーは各サーバーでsudo権限を与えておきます。opeサーバーには前述のインストール手順でTiUPをインストールしておきます。

サーバー環境を元にトポロジーファイルを作成する

TiUPを使用してTiDBを構築する前に、構成ファイルであるトポロジーファイルを作成します。このファイルはどのサーバーにどのコンポーネントの役割を与えるかを定義するファイルです。TiUPはこのトポロジーファイルを元に必要なコンポーネントを対象サーバーにインストールします。

TiUPはトポロジーファイルを作成する時に便利なテンプレート表示機能が実装されています。tiup cluster templateコマンドを使用して出力できますので、出力結果を元に環境に合わせてお好みのエディターで編集すると良いでしょう。次の例はviコマンドを使用した例になります。

トポロジーファイルのテンプレートを保存してviで編集する例

```
$ tiup cluster template > topology.yaml  
  
$ vi topology.yaml
```

テンプレートを表示する際に、より詳細な情報を出力させる`--full`オプションと、シンプルな形式で出力させる`--local`オプションがあります。お好みの方法でテンプレートを作成してください。また、複数のデータセンターにまたがってTiDBを構成する時に役に立つ`--multi-dc`オプションもあります。

より詳細なトポロジーファイルのテンプレート表示

```
$ tiup cluster template --full
```

シンプルなトポロジーファイルのテンプレート表示

```
$ tiup cluster template --local
```

複数のデータセンターにまたがる構成のトポロジーファイルのテンプレート表示

```
$ tiup cluster template --multi-dc
```

生成されたテンプレートを元にトポロジーファイルを編集する必要がありますが、`--local`オプションを指定した時は各種サーバーのIPアドレスや各種設定を上書きしてテンプレートを生成できます。こちらもうまく利用してトポロジーファイルの作成に役立ててください。

さまざまな情報を上書きしてテンプレートを出力

```
$ tiup cluster template --local --pd-servers 10.0.0.1 --tidb-servers 10.0.0.2 --tikv-servers 10.0.0.3  
...  
(省略)  
...  
pd_servers:  
  - host: 10.0.0.1  
  
tidb_servers:  
  - host: 10.0.0.2  
  
tikv_servers:  
  - host: 10.0.0.3  
...  
(省略)  
...
```

本節のサーバー構成のトポロジーファイルを生成してみましょう。TiDBサーバーは現時点では2台だけ使用します。

実際に使用する環境のトポロジーファイルを生成する

```
$ tiup cluster template --local \  
  --pd-servers 10.0.0.11,10.0.0.12,10.0.0.13 \  
  --tidb-servers 10.0.0.14,10.0.0.15 \  
  --tikv-servers 10.0.0.16,10.0.0.17,10.0.0.18
```

```
--tidb-servers 10.0.0.21,10.0.0.22 \  
--tikv-servers 10.0.0.31,10.0.0.32,10.0.0.33 \  
--grafana-servers 10.0.0.41 \  
--monitoring-servers 10.0.0.41 > topology.yaml
```

生成されたトポロジーファイルの内容は次のようになります。

```
topology.yaml  
global:  
  # # The OS user who runs the tidb cluster.  
  user: "tidb"  
  # # systemd_mode is used to select whether to use sudo permissions.  
  systemd_mode: "system"  
  # # SSH port of servers in the managed cluster.  
  ssh_port: 22  
  # # Storage directory for cluster deployment files, startup scripts, and configuration files.  
  deploy_dir: "/tidb-deploy"  
  # # TiDB Cluster data storage directory  
  data_dir: "/tidb-data"  
  # # Supported values: "amd64", "arm64" (default: "amd64")  
  arch: "amd64"  
  
pd_servers:  
  - host: 10.0.0.11  
  - host: 10.0.0.12  
  - host: 10.0.0.13  
  
tidb_servers:  
  - host: 10.0.0.21  
  - host: 10.0.0.22  
  
tikv_servers:  
  - host: 10.0.0.31  
  - host: 10.0.0.32  
  - host: 10.0.0.33  
  
monitoring_servers:  
  - host: 10.0.0.41  
  
grafana_servers:  
  - host: 10.0.0.41
```

本手順ではこのトポロジーファイルを使用してTiDBを構築していきます。トポロジーファイルでは、TiDB構築やTiUP実行に関するさまざまな設定を記述しますが、詳細な説明は--fullオプションを使用して作成したテンプレートのコメントを参照してください。

TiUP実行時のsshログイン方法を指定する

TiUPはトポロジーファイル内に記述された各サーバーに対してsshを使用してログインし、さまざまなコマンドを実行します。そのため、TiUPを実行する環境から作業対象のサーバーへsshログインが可能である必要があります。

TiUPはsshを使用してログインする際にデフォルトでは公開鍵認証を使用します。ホームディレクトリ内の`.ssh/id_rsa`ファイルが秘密鍵として使用されますが、別の秘密鍵を使用する場合は`--identity_file`オプションを指定して別の秘密鍵を指定してください。

公開鍵認証ではなく、パスワード認証を使用する場合は`--password`オプションを使用してください。

sshログインに使用するユーザーは`--user`オプションで指定可能です。

本項で説明したオプションはトポロジーファイルの構成チェック時やTiDBを構築する際に使用できます。次項で説明するトポロジーファイルのチェック(`tiup cluster check`コマンド)を例に、オプションの指定例を示します。

トポロジーファイルの構成チェックを実行する際に使用する秘密鍵を指定する

```
$ tiup cluster check topology.yaml --identity_file /home/tidb/.ssh/id_rsa_tidb
```

トポロジーファイルの構成チェックを実行する際にパスワード認証を使用する

```
$ tiup cluster check topology.yaml --password
```

トポロジーファイルを元に構成のチェックをする

トポロジーファイルの準備ができたらいよいよ構築です。ですが、構築する前にトポロジーファイルの内容を`tiup cluster check`コマンドでチェックしておくとい良いでしょう。

トポロジーファイルの構成チェック

```
$ tiup cluster check topology.yaml
```

`tiup cluster check`コマンドを実行するとさまざまなチェックが行われ、各種コンポーネントを動かす際の要件とそのチェック結果が表示されます。

tiup cluster checkの実行結果例

Node	Check	Result	Message
10.0.0.11	cpu-cores	Pass	number of CPU cores / threads: 4
10.0.0.11	memory	Pass	memory size is 8192MB
10.0.0.11	disk	Warn	mount point / does not have 'noatime' option set
10.0.0.11	sysctl	Fail	net.core.somaxconn = 4096, should be greater than 32768
10.0.0.11	sysctl	Fail	net.ipv4.tcp_syncookies = 1, should be 0

```
10.0.0.11 sysctl      Fail    vm.swappiness = 60, should be 0
10.0.0.11 thp         Fail    THP is enabled, please disable it for best performance
10.0.0.11 service     Fail    service irqbalance not found, should be installed and started
10.0.0.11 service     Fail    service firewalld is running but should be stopped
10.0.0.11 timezone    Pass    time zone is the same as the first PD machine: UTC
10.0.0.11 os-version   Pass    OS is Rocky Linux 9.4 (Blue Onyx)
10.0.0.11 cpu-governor Warn    Unable to determine current CPU frequency governor policy
10.0.0.11 limits      Fail    soft limit of 'nofile' for user 'tidb' is not set or too low
10.0.0.11 limits      Fail    hard limit of 'nofile' for user 'tidb' is not set or too low
10.0.0.11 limits      Fail    soft limit of 'stack' for user 'tidb' is not set or too low
10.0.0.11 selinux     Fail    SELinux is not disabled
10.0.0.11 command     Fail    numactl not usable, bash: line 1: numactl: command not found
...
(省略)
...
```

ResultがFailになっている項目は特に注意してください。各サーバーに必要な設定をする必要があります。tiup cluster check コマンドは各サーバーに必要な設定を自動で反映してくれる--apply オプションがありますので、--apply オプションと共に実行して修正を試みることも可能です。ただし、--apply を実行しても修正されない項目もありますので、その際には手動で修正が必要になります。特に本番環境ではドキュメントのTiDB Environment and System Configuration Check^{注4}を参考にして環境構築することをおすすめします。

できるだけTiUPで修正を試みる

```
$ tiup cluster check topology.yaml --apply
```

最終的にFailの項目が無くなるまで何度も確認しながら修正してください。

トポロジーファイルを元にTiDBを構築する

tiup cluster check コマンドの結果が問題なくなればよいTiDB構築です。構築はtiup cluster deploy コマンドで行います。構築時にはクラスター名とTiDBバージョンを指定します。このクラスター名はTiUPによって構築するTiDBの識別子となります。クラスター名に使用できる文字は英数字と記号「-」「_」「.」です。先頭文字は英字である必要があります。本節では執筆時点での最新のLTSバージョンである8.5.0を使用します。クラスター名はtidb-sampleとします。

クラスター名とバージョンを指定して構築する

```
$ tiup cluster deploy tidb-sample v8.5.0 topology.yaml
```

注4 <https://docs.pingcap.com/tidb/stable/check-before-deployment>

実行すると確認画面が表示されます。

```
構築前の最終確認
Please confirm your topology:
Cluster type:  tidb
Cluster name:  tidb-sample
Cluster version: v8.5.0
Role          Host          Ports          OS/Arch          Directories
----          -
pd            10.0.0.11     2379/2380      linux/x86_64     /tidb-deploy/pd-2379, ...
pd            10.0.0.12     2379/2380      linux/x86_64     /tidb-deploy/pd-2379, ...
pd            10.0.0.13     2379/2380      linux/x86_64     /tidb-deploy/pd-2379, ...
tikv          10.0.0.31     20160/20180    linux/x86_64     /tidb-deploy/tikv-20160, ...
tikv          10.0.0.32     20160/20180    linux/x86_64     /tidb-deploy/tikv-20160, ...
tikv          10.0.0.33     20160/20180    linux/x86_64     /tidb-deploy/tikv-20160, ...
tidb          10.0.0.21     4000/10080     linux/x86_64     /tidb-deploy/tidb-4000
tidb          10.0.0.22     4000/10080     linux/x86_64     /tidb-deploy/tidb-4000
prometheus    10.0.0.41     9090/12020     linux/x86_64     /tidb-deploy/prometheus-9090, ...
grafana       10.0.0.41     3000           linux/x86_64     /tidb-deploy/grafana-3000
Attention:
  1. If the topology is not what you expected, check your yaml file.
  2. Please confirm there is no port/directory conflicts in same host.
Do you want to continue? [y/N]: (default=N)
```

問題なければyを入力します。なお、確認は--yes オプションを使用することで省略可能です。構築が終了すると次のメッセージが表示されます。

```
構築後のメッセージ
Cluster `tidb-sample` deployed successfully, you can start it with command: `tiup cluster start
tidb-sample --init`
```

現時点ではまだTiDBは起動していません。TiDBの状態はtiup cluster display コマンドで確認できますので確認してみましょう。

```
TiDBの状態を表示
$ tiup cluster display tidb-sample
Cluster type:      tidb
Cluster name:      tidb-sample
Cluster version:   v8.5.0
Deploy user:       tidb
SSH type:          builtin
Grafana URL:       http://10.0.0.41:3000
ID                Role      Host          Ports          OS/Arch          Status ...
--              ---
10.0.0.41:3000    grafana   10.0.0.41     3000           linux/x86_64     Down ...
10.0.0.11:2379    pd        10.0.0.11     2379/2380      linux/x86_64     Down ...
10.0.0.12:2379    pd        10.0.0.12     2379/2380      linux/x86_64     Down ...
10.0.0.13:2379    pd        10.0.0.13     2379/2380      linux/x86_64     Down ...
```

```
10.0.0.41:9090    prometheus  10.0.0.41    9090/12020   linux/x86_64  Down  ...
10.0.0.21:4000    tidb        10.0.0.21    4000/10080   linux/x86_64  Down  ...
10.0.0.22:4000    tidb        10.0.0.22    4000/10080   linux/x86_64  Down  ...
10.0.0.31:20160   tikv        10.0.0.31    20160/20180  linux/x86_64  N/A    ...
10.0.0.32:20160   tikv        10.0.0.32    20160/20180  linux/x86_64  N/A    ...
10.0.0.33:20160   tikv        10.0.0.33    20160/20180  linux/x86_64  N/A    ...
Total nodes: 10
```

Statusの項目がDownまたはN/Aとなっていることが確認できます。

それではいよいよTiDBを起動してみましょう。起動はtiup cluster start コマンドで行います。TiDBの起動方法は、rootユーザーのパスワードを自動生成されたランダムな文字列に設定して起動する方法と、設定しないで起動する方法の2種類あります。

パスワードを設定して起動する場合は--init オプションを使用します。パスワードは一度しか表示されませんので、忘れずに保存してください。

--initオプションを使用しない場合はrootユーザーのパスワードは設定されません。

データベースの初期パスワードを生成して起動する

```
$ tiup cluster start tidb-sample --init
...
(省略)
...
Started cluster `tidb-sample` successfully
The root password of TiDB database has been changed.
The new password is: 'i-863Zw^y_9PJ4@rk2'.
Copy and record it to somewhere safe, it is only displayed once, and will not be stored.
The generated password can NOT be get and shown again.
```

「Started cluster <クラスター名> successfully」が表示されれば成功です。再びtiup cluster display コマンドで状態を確認します。

TiDBの状態を表示

```
$ tiup cluster display tidb-sample
Cluster type:      tidb
Cluster name:      tidb-sample
Cluster version:   v8.5.0
Deploy user:       tidb
SSH type:          builtin
Grafana URL:       http://10.0.0.41:3000
ID                 Role      Host      Ports      OS/Arch      Status ...
--                 ---      ---      ---      ---      ---
10.0.0.41:3000     grafana   10.0.0.41  3000       linux/x86_64 Up    ...
10.0.0.11:2379     pd        10.0.0.11  2379/2380  linux/x86_64 Up    ...
10.0.0.12:2379     pd        10.0.0.12  2379/2380  linux/x86_64 Up|L   ...
10.0.0.13:2379     pd        10.0.0.13  2379/2380  linux/x86_64 Up|UI  ...
10.0.0.41:9090     prometheus 10.0.0.41  9090/12020 linux/x86_64 Up    ...
```

```

10.0.0.21:4000    tidb      10.0.0.21  4000/10080  linux/x86_64  Up    ...
10.0.0.22:4000    tidb      10.0.0.22  4000/10080  linux/x86_64  Up    ...
10.0.0.31:20160   tikv      10.0.0.31  20160/20180 linux/x86_64  Up    ...
10.0.0.32:20160   tikv      10.0.0.32  20160/20180 linux/x86_64  Up    ...
10.0.0.33:20160   tikv      10.0.0.33  20160/20180 linux/x86_64  Up    ...
Total nodes: 10

```

StatusがUpになっていれば起動成功です。PDサーバーのStatusのLはリーダーの役割、UIはダッシュボードの役割を示しています。

表示されたrootユーザーのパスワードでデータベースへの接続を確認します。

初期パスワードでのデータベースへのログイン

```

$ mysql -u root --host 10.0.0.11 --port 4000 -p
Enter password: (初期パスワードを入力)
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1600126982
Server version: 8.0.11-TiDB-v8.5.0 TiDB Server (Apache License 2.0) ...

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>

```

ログインが確認できたら初期パスワードは適宜変更してください。変更する場合は次のようにSQLを実行します。

rootユーザーのパスワード変更

```
mysql> ALTER USER 'root'@'%' IDENTIFIED BY 'newpassword';
```

▶ データベースのrootパスワードを紛失した場合

もしもデータベースのrootユーザーのパスワードが分からなくなってしまった場合は次の手順で復旧させてください。

まずTiDBサーバーにログインし、設定ファイルのtidb.tomlをエディターで編集します。TiUPで構築時に特に設定を変更していなければ、通常は/tidb-deploy/tidb-4000/tidb.tomlに存在します。

エディターでtidb.tomlを編集:viの場合

```
$ vi /tidb-deploy/tidb-4000/tidb.toml
```

次の内容を書き込みます。

```
[security]
skip-grant-table = true
```

次にtidb-serverプロセスを停止します。

```
tidb-serverプロセスの停止
$ sudo systemctl stop tidb-4000.service
```

次にtidb-serverプロセスを手動で立ち上げます。起動スクリプトがありますのでそれを使用して起動します。通常は/tidb-deploy/tidb-4000/scripts/run_tidb.shに配置されています。run_tidb.shを実行するとフォアグラウンドでtidb-serverプロセスが起動されます。

```
tidb-serverプロセスを一時的に手動で起動
$ sudo /tidb-deploy/tidb-4000/scripts/run_tidb.sh
```

tidb-serverが起動したらmysqlコマンドで接続します。

```
一時的に起動したtidb-serverプロセスへ接続
$ mysql -u root --host 10.0.0.21 --port 4000
```

接続したらALTERステートメントを実行して新しいパスワードを設定します。FLUSH PRIVILEGESは必要に応じて実行してください。

```
rootユーザーのパスワードを新しく設定
mysql> ALTER USER 'root'@'%' IDENTIFIED BY 'newpassword';
Query OK, 0 rows affected (0.04 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)
```

パスワードの変更が終わったらrun_tidb.shの実行を`Ctrl` + `C`で停止します。

```
手動実行したtidb-serverプロセスの停止
$ sudo /tidb-deploy/tidb-4000/scripts/run_tidb.sh
^C
```

その後編集したtidb.tomlを元に戻します。エディターで開き、skip-grant-tableの設定を削除してください。

```
エディターでtidb.tomlの設定を元に戻す:viの場合
$ vi /tidb-deploy/tidb-4000/tidb.toml
...
(省略)
...
[security]
```

```
skip-grant-table = true ←この行を削除
```

最後にtidb-serverプロセスを起動します。

```
tidb-serverプロセスの起動
```

```
$ sudo systemctl start tidb-4000.service
```

これでrootユーザーのパスワード復旧作業は終了です。

▶ TiDB構築時における特定の設定の反映

TiDBを構築する際に、各種コンポーネントの設定を変更できます。適用させたい設定がある場合は、トポロジーファイル内のserver_configsで定義します。

例としてTiDBコンポーネントの使用するCPUコア数を2に制限し、TiKV側でのストレージI/Oの最大量を300MiBに制限する設定でTiDBを構築する例を示します。

```
topology_custom_config.yaml
```

```
server_configs:
  tikv:
    storage.io-rate-limit.max-bytes-per-sec: 300MiB
  tidb:
    performance.max-procs: 2

pd_servers:
- host: 10.0.0.11
- host: 10.0.0.12
- host: 10.0.0.13
...
(省略)
...
```

設定できる項目は多岐にわたります。server_configsをはじめ、トポロジーファイルの完全な書式はドキュメントのTopology Configuration File for TiDB Deployment Using TiUP^{注5}を参考にしてください。

■ 現在のトポロジー状態をファイルに保存する

TiDB構築に使用したトポロジーファイルを保存しておくことは、TiDBの状態を把握する上で重要です。しかし、たとえば構築後にTiDB内のコンポーネントをスケールアウトやスケールインをした場合、構築時のトポロジーファイルとは差分が出てしまう問題があります。この問題を解決するために、構築終了後にtiup cluster show-configコマンドを実行して、現在の構成や設定をダンプしたものを保存することをおすすめします。

注5 <https://docs.pingcap.com/tidb/stable/tiup-cluster-topology-reference>

現在のTiDBの状態を出力

```
$ tiup cluster show-config tidb-sample > tidb-sample-$(date +%Y%m%d-%H%M%S).yaml
```

tiup cluster show-configコマンドで出力されるトポロジーファイルは構築時に使用したトポロジーファイルで省略した設定も表示され、より詳細に状態を把握できるというメリットもあります。トポロジーファイルをGitのようなバージョン管理システムで管理することでTiDBの状態遷移をたどれるようになりますので、定期的にtiup cluster show-configコマンドを実行して保存しておくことを検討してください。

TiUPの同時実行数を変更する

TiUP実行時に対象サーバーの数が多く場合、TiUPの実行完了に時間がかかってしまう場合があります。その場合はTiUPが行う操作の同時数を増やすことで操作効率が良くなります。同時実行数を変更する場合はTiUP実行時に--concurrencyオプションを使用してください。デフォルトは5が設定されています。

3-4

各種コンポーネントのスケールアウトとスケールイン

構築したTiDBをスケールアウトやスケールインさせる場合、TiUPから行うことができます。スケールアウトする際にはスケールアウト対象のみを記述したトポロジーファイルを用意し、TiUP実行の際に読み込ませます。スケールインする際には対象のサーバーを指定してスケールインします。スケールアウトとスケールインで少々TiUPの使い方が違いますので注意してください。

各種コンポーネントをスケールアウトする

先ほど作成したtidb-sampleはTiDBサーバーが2台の構成でした。ここではTiDBサーバーを1台スケールアウトさせてみます。まず対象のサーバー(10.0.0.23)をスケールアウト用のトポロジーファイルに記述します。

scale_out.yaml

```
tidb_servers:
- host: 10.0.0.23
```

繰り返しになりますが、スケールアウト対象のみトポロジーファイルに記述することに注意してください。トポロジーファイルが用意できたらtiup cluster checkコマンドを実行します。すでに構築しているTiDBに対するチェックになりますので、構築時とは違い--clusterオプションを指定します。

---clusterを使用して確認する

```
$ tiup cluster check tidb-sample scale_out.yaml --cluster
```

構築時に行ったのと同様の事前チェックが新しいサーバー(10.0.0.23)に対して行われます。同様にResultがFailになっているチェック項目を無くすように修正してください。--applyオプションも使用できます。

新規サーバーに必要な設定を自動で適用する

```
$ tiup cluster check tidb-sample scale_out.yaml --cluster --apply
```

準備ができればtiup cluster scale-outコマンドを実行します。

準備ができればスケールアウトする

```
$ tiup cluster scale-out tidb-sample scale_out.yaml
```

構築時と同じように確認を求められます。問題なければyを入力します。こちらも確認は--yesオプションを使用することで省略可能です。

スケールアウト前の確認

```
+ Detect CPU Arch Name
- Detecting node 10.0.0.23 Arch info ... Done

+ Detect CPU OS Name
- Detecting node 10.0.0.23 OS info ... Done
Please confirm your topology:
Cluster type:   tidb
Cluster name:   tidb-sample
Cluster version: v8.5.0
Role  Host      Ports      OS/Arch      Directories
----  ---
tidb  10.0.0.23    4000/10080  linux/x86_64 /tidb-deploy/tidb-4000
Attention:
  1. If the topology is not what you expected, check your yaml file.
  2. Please confirm there is no port/directory conflicts in same host.
Do you want to continue? [y/N]: (default=N)
```

「Scaled cluster <クラスター名> out successfully」が表示されたら成功です。

スケールアウト成功時のメッセージ

```
...
(省略)
...
Scaled cluster `tidb-sample` out successfully
```

TiDBの状態をtiup cluster displayコマンドで確認します。

スケールアウト後のTiDBの状態を表示

```
$ tiup cluster display tidb-sample
Cluster type:      tidb
Cluster name:      tidb-sample
Cluster version:   v8.5.0
Deploy user:       tidb
SSH type:          builtin
Grafana URL:       http://10.0.0.41:3000
```

ID	Role	Host	Ports	OS/Arch	Status ...
10.0.0.41:3000	grafana	10.0.0.41	3000	linux/x86_64	Up ...
10.0.0.11:2379	pd	10.0.0.11	2379/2380	linux/x86_64	Up ...
10.0.0.12:2379	pd	10.0.0.12	2379/2380	linux/x86_64	Up L ...
10.0.0.13:2379	pd	10.0.0.13	2379/2380	linux/x86_64	Up UI ...
10.0.0.41:9090	prometheus	10.0.0.41	9090/12020	linux/x86_64	Up ...
10.0.0.21:4000	tidb	10.0.0.21	4000/10080	linux/x86_64	Up ...
10.0.0.22:4000	tidb	10.0.0.22	4000/10080	linux/x86_64	Up ...
10.0.0.23:4000	tidb	10.0.0.23	4000/10080	linux/x86_64	Up ...
10.0.0.31:20160	tikv	10.0.0.31	20160/20180	linux/x86_64	Up ...
10.0.0.32:20160	tikv	10.0.0.32	20160/20180	linux/x86_64	Up ...
10.0.0.33:20160	tikv	10.0.0.33	20160/20180	linux/x86_64	Up ...

Total nodes: 11

3台目のTiDBサーバー(10.0.0.23)のStatusがUpになり、TiDBのノードとして認識されていることが確認できます。

ここではTiDBサーバーのスケールアウトを紹介しましたが、もちろん他のコンポーネントも同様にスケールアウトできます。スケールアウト用のトポロジーファイルの例をいくつか記述しておきますので、参考にしてください。TiFlashやTiProxyのコンポーネントを新規に構築する場合も、こちらのスケールアウト手順で構築してください。

TiKVサーバーのscale_out.yamlの例

```
tikv_servers:
- host: 10.0.0.61
```

TiFlashサーバーのscale_out.yamlの例

```
tiflash_servers:
- host: 10.0.0.71
```

TiProxyサーバーのscale_out.yamlの例

```
tiproxy_servers:
- host: 10.0.0.81
```

各種コンポーネントをスケールインする

コンポーネントをスケールインする場合はスケールアウトの時とは違い、トポロジーファイルを用意する必要はありません。その代わりにノード(IPアドレスとポート番号の組み合わせ)を指定してスケールインします。ノードの情報はtiup cluster displayコマンドで確認できます。

先ほどスケールアウトしたTiDBサーバー(10.0.0.23:4000)を--nodeオプションで指定してスケールインしてみましょう。

IPアドレスとポート番号を指定してスケールイン

```
$ tiup cluster scale-in tidb-sample --node 10.0.0.23:4000
```

スケールイン前に確認を求められます。問題なければyを入力します。なお、確認は--yesオプションを使用することで省略可能です。

スケールイン前の確認

```
This operation will delete the 10.0.0.23:4000 nodes in `tidb-sample` and all their data.
Do you want to continue? [y/N]:(default=N)
```

「Scaled cluster <クラスター名> in successfully」が表示されたら成功です。

スケールイン成功時のメッセージ

```
Scaled cluster `tidb-sample` in successfully
```

再度tiup cluster displayコマンドで確認します。

正常なクラスターの状態を表示

```
$ tiup cluster display tidb-sample
```

```
...
(省略)
```

```
...
ID          Role      Host      Ports      OS/Arch      Status ...
--          ---      ---      ---      ---      ---
10.0.0.41:3000 grafana   10.0.0.41  3000      linux/x86_64 Up      ...
10.0.0.11:2379 pd        10.0.0.11  2379/2380 linux/x86_64 Up      ...
10.0.0.12:2379 pd        10.0.0.12  2379/2380 linux/x86_64 Up|L    ...
10.0.0.13:2379 pd        10.0.0.13  2379/2380 linux/x86_64 Up|UI   ...
10.0.0.41:9090 prometheus 10.0.0.41  9090/12020 linux/x86_64 Up      ...
10.0.0.21:4000 tidb      10.0.0.21  4000/10080 linux/x86_64 Up      ...
10.0.0.22:4000 tidb      10.0.0.22  4000/10080 linux/x86_64 Up      ...
10.0.0.31:20160 tikv      10.0.0.31  20160/20180 linux/x86_64 Up      ...
10.0.0.32:20160 tikv      10.0.0.32  20160/20180 linux/x86_64 Up      ...
10.0.0.33:20160 tikv      10.0.0.33  20160/20180 linux/x86_64 Up      ...
Total nodes: 10
```

一覧から10.0.0.23:4000が消えていることを確認します。

3-5

TiDB Dashboardの構築

TiDBは運用に役立つ組み込みのダッシュボード機能が備わっています(後に説明するGrafanaのダッシュボードと区別するために、ここではTiDB Dashboardと表記します)。TiDB Dashboardでは、TiDB クラスターのQPS(1秒あたりのクエリ数)、クエリ実行時間、最も多くのリソースを消費しているSQL ステートメントの種類などを確認できます。TiDB DashboardはオープンソースとしてGitHubに公開^{注6}されています。

TiDB Dashboardに接続する

TiDB DashboardはPDサーバーの2379番ポートにアクセスすることで表示できます。TiDB Playground 起動時はTiDB DashboardのURLは`http://127.0.0.1:2379/dashboard`になります。TiUPでTiDBを構築している場合は`http://<PDサーバー>:2379/dashboard`になります。このURLは`tiup cluster display`コマンドで確認でき、「Dashboard URL:」として表示されます。

TiDB DashboardのURLを確認する

```
$ tiup cluster display tidb-sample
Cluster type:      tidb
Cluster name:      tidb-sample
Cluster version:   v8.5.0
Deploy user:       tidb
SSH type:          builtin
Dashboard URL:     http://10.0.0.13:2379/dashboard
Grafana URL:       http://10.0.0.41:3000
...
(省略)
...
```

また、実行時に`--dashboard`オプションを付けた場合はTiDB DashboardのURLのみ表示されます。

TiDB DashboardのURLのみを表示

```
$ tiup cluster display tidb-sample --dashboard
Dashboard URL: http://10.0.0.13:2379/dashboard/
```

PDサーバーが複数台で構成されている環境では、その中の1台のみがTiDB Dashboardを担当します。TiDB Dashboardを担当しているPDサーバーに障害が発生しダウンした場合や、PDサーバーの台数が変化したとしても自動でTiDB Dashboard担当は切り替わりません。理由としてはTiDB Dashboard担当

注6 <https://github.com/pingcap-incubator/tidb-dashboard>

のPDサーバーのローカルディスクにKey Visualizeの履歴や検索履歴などのデータが保存されるからです。

TiDB Dashboardを担当していないPDサーバーの2379番ポートにアクセスした場合は、担当しているPDサーバーへリダイレクトされます。先ほど作成したtidb-sampleの例ではpd03(10.0.0.13)がTiDB Dashboardを担当しています。pd01(10.0.0.11)に対してアクセスした場合は、pd03にリダイレクトされています。

正しいURLへリダイレクトされる

```
$ curl -I http://10.0.0.11:2379/dashboard/
HTTP/1.1 307 Temporary Redirect
Access-Control-Allow-Headers: accept, content-type, authorization
Access-Control-Allow-Methods: POST, GET, OPTIONS, PUT, DELETE
Access-Control-Allow-Origin: *
Content-Type: text/html; charset=utf-8
Location: http://10.0.0.13:2379/dashboard/
Date: Mon, 18 Nov 2024 16:51:34 GMT
```

TiDB Dashboardを無効化する

TiDB Dashboard機能はデフォルトで有効になっていますが、必要に応じて無効化できます。無効化が必要な場合はPDサーバーの設定を変更します。

稼働しているTiDBおよびTiDB Playground

すでに稼働しているTiDBおよびTiDB PlaygroundにてPDサーバーの設定を変更する場合は、tiupコマンドからpd-ctlコマンドを呼び出して操作してください。pd-ctlコマンドはPDサーバーに対して設定を変更したり、さまざまな操作ができるクライアントです。pd-ctlコマンドはtiup cli:<クラスタのバージョン> pdから実行できます。実行時には操作対象のTiDBのバージョンを指定する必要があります。

tidb-sampleに対する操作の例を記述します。TiDB Playgroundも同様の手順でTiDB Dashboardを無効化できます。その場合は--pdオプションでhttp://127.0.0.1:2379を指定してください。

tidb-sampleでTiDB Dashboardを無効化する

```
$ tiup ctl:v8.5.0 pd --pd http://10.0.0.13:2379 config set dashboard-address none
Starting component ctl: ...
Success!
```

「Success!」と表示されれば成功です。TiDB Dashboardにアクセスしてみます。

TiDB Dashboard無効化の確認 その1

```
$ curl http://10.0.0.13:2379/dashboard/
Dashboard is not started.
```

「Dashboard is not started.」と表示され、無効化されていることが確認できます。tiup cluster display コマンドでも同様に確認できます。

TiDB Dashboard無効化の確認 その2

```
$ tiup cluster display tidb-sample --dashboard
Error: TiDB Dashboard is disabled
```

▶ TiDB構築時およびTiDB Playground起動時

TiDB 構築時に最初から TiDB Dashboard を無効化しておきたい場合は、PD サーバーの設定項目 dashboard-address を none にしたファイルを読み込ませて起動させます。TiUP による構築の場合はトポロジーファイルで設定します。

topology_pd_disable_dashboard.yaml

```
...
(省略)
...
server_configs:
  pd:
    dashboard-address: none
...
(省略)
...
```

TiDB構築時にTiDB Dashboardを無効化

```
$ tiup cluster deploy v8.5.0 no-dashboard-cluster topology_pd_disable_dashboard.yaml
```

TiDB Playground の起動時は --pd.config オプションを使用して設定ファイルを読み込ませてください。

pd_disable_dashboard.toml

```
[pd-server]
  dashboard-address = "none"
```

TiDB Playground起動時にTiDB Dashboardを無効化

```
$ tiup playground --pd.config pd_disable_dashboard.toml
```

無効化した TiDB Dashboard を再び有効化する場合は、次項の「TiDB Dashboard を担当している PD サーバーを切り替える」の手順で PD サーバーを設定してください。

■ TiDB Dashboardを担当しているPDサーバーを切り替える

TiDB Dashboard の無効化手順とほぼ同様の操作をすることで、TiDB Dashboard を担当する PD サー

バーを切り替えることができます。無効化する際はPDサーバーの設定項目 dashboard-address に対して none を設定しましたが、他のPDサーバーのURLを指定することで担当を切り替えることができます。tidb-sampleのpd03(10.0.0.13)のTiDB Dashboard担当をpd01(10.0.0.11)へ変更する手順を示します。

pd03からpd01へ担当を変更する

```
$ tiup ctl:v8.5.0 pd --pd http://10.0.0.13:2379 config set dashboard-address http://10.0.0.11:2379
Starting component ctl: ...
Success!
```

設定が反映されたか確認します。

pd01に切り替わっているか確認

```
$ tiup cluster display tidb-sample --dashboard
Dashboard URL: http://10.0.0.11:2379/dashboard/
```

切り替えた際はローカルディスクに保存されたデータ(Key Visualizerの履歴や検索履歴など)は引き継がれませんので注意してください。また、PDサーバーが複数の場合はその中に1台だけリーダーがあります。リーダーのPDサーバーにはTiDB Dashboardの役割を振らない方がいいでしょう。どのPDサーバーがリーダーの役割を持っているかはtiup cluster displayコマンドで確認できます。

リーダーを確認

```
$ tiup cluster display tidb-sample --role pd
...
(省略)
...


| ID             | Role | Host      | Ports     | OS/Arch      | Status | ... |
|----------------|------|-----------|-----------|--------------|--------|-----|
| 10.0.0.11:2379 | pd   | 10.0.0.11 | 2379/2380 | linux/x86_64 | Up UI  | ... |
| 10.0.0.12:2379 | pd   | 10.0.0.12 | 2379/2380 | linux/x86_64 | Up L   | ... |
| 10.0.0.13:2379 | pd   | 10.0.0.13 | 2379/2380 | linux/x86_64 | Up     | ... |


Total nodes: 3
```

--roleオプションを使用してPDサーバーだけ表示しています。StatusのところにLが表示されているサーバーがリーダーとなります。

3-6 モニタリング環境の構築

TiDBではメトリクスの収集、保存する手段として、内部ではPrometheus^{注7}が使用されており、メト

注7 <https://prometheus.io/>

リクスの可視化はGrafana^{注8}ダッシュボードより可能です。ですが、PrometheusやGrafana環境のセットアップの手間を考える必要はありません。なぜならばTiUPによってPrometheusやGrafana環境の構築は自動化されているからです。また、必要に応じて定義したPrometheusのアラート通知を管理するAlertmanager^{注9}も構築できます。

各種メトリクスに必要なコンポーネントを構築するためには、TiUPのトポロジーファイル内で定義します。Prometheusは`monitoring_servers`としてセットアップしたい対象のサーバー情報を定義します。Grafanaは`grafana_servers`で定義します。Alertmanagerが必要な場合は`alertmanager_servers`で定義します。

トポロジーファイル内での各種モニタリングの構築

```
monitoring_servers:
  - host: 10.0.0.41

grafana_servers:
  - host: 10.0.0.41

alertmanager_servers:
  - host: 10.0.0.41
```

複数のサーバー情報を定義できますが、`monitoring_servers`に関しては複数台設定しても最初のサーバーしか利用されませんので注意してください。

Prometheus、GrafanaおよびAlertmanagerのバージョンを指定する場合はトポロジーファイル内の`component_versions`で定義します。

トポロジーファイル内でのcomponent_versionsによるバージョンの指定

```
component_versions:
  prometheus: "v8.5.0"
  grafana: "v8.5.0"
  alertmanager: "v0.26.0"
```

PrometheusとGrafanaはTiDBと同じバージョン体系ですが、Alertmanagerは個別の体系になっています。使用可能なバージョンの確認はTiUPで確認します。

利用可能なPrometheusのバージョン

```
$ tiup list prometheus
Available versions for prometheus:
Version          Installed  Release    ...
-----          -
nightly -> v9.0.0-alpha-nightly  2024-12-26T14:15:31Z ...
...
```

注8 <https://grafana.com/grafana/>

注9 <https://prometheus.io/docs/alerting/latest/alertmanager/>


```
...
v8.5.0                YES      2024-12-19T06:11:28Z ...
v9.0.0-alpha-nightly 2024-12-26T14:15:31Z ...
利用可能なAlertmanagerのバージョン
$ tiup list alertmanager
Available versions for alertmanager:
Version  Installed  Release                                Platforms
-----  -
v0.17.0                2020-05-28T18:49:09+08:00  linux/arm64,darwin/amd64,linux/amd64
v0.26.0                2024-05-24T14:27:41+08:00  linux/arm64,linux/amd64
```

PrometheusおよびGrafanaのバージョンを確認する

PrometheusおよびGrafanaのバージョンはTiDBのバージョン体系で提供されています。そのため、内部で使用されているそれぞれのソフトウェアのバージョンはTiUPなどで確認できません。参考までに内部で使用されているPrometheusとGrafanaのバージョンを表3.4に示します。

表3.4 PrometheusおよびGrafanaのバージョン対応表

TiDBバージョン	Prometheusバージョン	Grafanaバージョン
8.5.z	v2.49.1	v7.5.17
7.5.z	v2.27.1	v7.5.17

PrometheusおよびGrafanaは、TiDBのモニタリングのために動作を確認したバージョンが^{どうこん}同梱されています。そのためこれらのバージョンは任意に更新されます。

TiDB Playgroundでモニタリング環境の起動を抑制する

TiDB Playgroundでは起動時にデフォルトでPrometheusとGrafanaが起動され、モニタリング環境が有効になります。モニタリング環境が起動しないようにする場合は`--without-monitor`オプションを使用します。

```
モニタリング環境の起動を抑制してTiDB Playgroundを起動
$ tiup playground --without-monitor
```

なお、TiDB PlaygroundではAlertmanagerには対応していません。

3-7

TiDBのライフサイクル操作

TiUPを使用してTiDB全体および一部のコンポーネントごとにライフサイクル操作(起動、停止、再起動、削除)を行うことができます。起動の仕方はすでに構築する手順の中で紹介しました。本節では停止、再起動、削除について説明していきます。TiDBの停止時には各サーバーで起動するプロセスが停止されますが、データやログは残ります。TiDBを削除する場合はデフォルトではデータやログが失われますので、実行の際には注意してください。

TiDBを停止する

TiDB全体を停止する場合は`tiup cluster stop`コマンドで行います。確認が求められますので問題なければ`y`を入力します。`--yes`オプションを使用することで確認をスキップできます。

tidb-sampleの停止

```
$ tiup cluster stop tidb-sample
Will stop the cluster tidb-sample with nodes: , roles: .
Do you want to continue? [y/N]:(default=N)
```

TiUPによる停止作業が正常に終了すると「Stopped cluster '<クラスター名>' successfully」が表示されます。

tidb-sampleの停止完了

```
Stopped cluster `tidb-sample` successfully
```

`tiup cluster display`コマンドで確認すると各プロセスのStatusがDownまたはN/Aになっていることが確認できます。

停止しているTiDBの状態を表示

```
$ tiup cluster display tidb-sample
Cluster type:      tidb
Cluster name:      tidb-sample
Cluster version:   v8.5.0
Deploy user:       tidb
SSH type:          builtin
Grafana URL:       http://10.0.0.41:3000
ID                 Role      Host      Ports      OS/Arch      Status ...
--                 -
10.0.0.41:3000     grafana  10.0.0.41  3000       linux/x86_64 Down ...
10.0.0.11:2379     pd       10.0.0.11  2379/2380  linux/x86_64 Down ...
10.0.0.12:2379     pd       10.0.0.12  2379/2380  linux/x86_64 Down ...
```

```

10.0.0.13:2379    pd      10.0.0.13    2379/2380    linux/x86_64 Down ...
10.0.0.41:9090    prometheus 10.0.0.41    9090/12020   linux/x86_64 Down ...
10.0.0.21:4000    tidb      10.0.0.21    4000/10080   linux/x86_64 Down ...
10.0.0.22:4000    tidb      10.0.0.22    4000/10080   linux/x86_64 Down ...
10.0.0.31:20160   tikv      10.0.0.31    20160/20180  linux/x86_64 N/A ...
10.0.0.32:20160   tikv      10.0.0.32    20160/20180  linux/x86_64 N/A ...
10.0.0.33:20160   tikv      10.0.0.33    20160/20180  linux/x86_64 N/A ...
Total nodes: 10

```

再び起動したい場合には `tiup cluster start` コマンドを実行します。

```

tidb-sampleを再び起動
$ tiup cluster start tidb-sample
...
(省略)
...
Started cluster `tidb-sample` successfully

```

停止しているTiDBに対して、TiUPから停止してもエラーにならず正常終了します。起動も同様に、起動しているTiDBに対してTiUPから起動してもエラーになりません。

コンポーネントの停止の順番は次のようになります。

- ・モニタリングコンポーネント > TiFlash コンポーネント > TiDB クラスター > ストレージクラスター > PD クラスター

TiDBを再起動する

停止と起動を一度に行う再起動もTiUPから実行できます。再起動する時は `tiup cluster restart` コマンドを実行します。

```

tidb-sampleを再起動
$ tiup cluster restart tidb-sample
...
(省略)
...
Restarted cluster `tidb-sample` successfully

```

TiDBを削除する

TiDBを削除する場合は `tiup cluster destroy` コマンドを実行します。繰り返しになりますが、TiDBを削除した場合はオプションで指定しない限りはデータやログが失われますので必要に応じて削除前にバックアップを取得してください。バックアップの仕方は第6章の「バックアップとリストア」で説明

します。

TiDBの削除

```
$ tiup cluster destroy tidb-sample
```

実行すると注意が表示されます。

TiDB削除前の確認

This operation will destroy tidb v8.5.0 cluster tidb-sample and its data.

Are you sure to continue?

(Type "Yes, I know my cluster and data will be deleted." to continue)

問題無ければ「Yes, I know my cluster and data will be deleted.」を入力してください。正常に TiDB が削除されると「Destroyed cluster 'クラスター名' successfully」が表示されます。

TiDB削除後のメッセージ

(省略)

Destroyed cluster `tidb-sample` successfully

▶ 削除時のデータやログの保持

データやログをサーバー上に保持したまま TiDB を削除する場合は `--retain-node-data` オプションまたは `--retain-role-data` オプションを使用します。

データやログを保持してTiDBを削除:特定のノードを指定

```
$ tiup cluster destroy tidb-sample --retain-node-data 10.0.0.21:4000
```

データやログを保持してTiDBを削除:特定のロールを指定

```
$ tiup cluster destroy tidb-sample --retain-role-data tikv
```

OS起動時のプロセス自動起動の有効／無効を切り替える

TiDBを構成する各サーバーになんらかの障害が起こり、OSが停止してしまうケースもあります。OS再起動を伴うメンテナンスをしなければならない場合もあるかもしれません。TiDBのコンポーネントは、デフォルトではOSが起動された際に自動的にプロセスが起動します。自動で起動させたくない場合は `tidb cluster disable` コマンドで設定を変更します。自動起動を無効にした後に再度有効化する

ためにはtidb cluster enableコマンドを実行します。

TiUPによる構築を行うと、各プロセスは各サーバー内のsystemd^{注10}配下で管理されます。たとえばTiDBサーバーではtidb-4000.serviceというサービス名で登録されており、<ロール名>-<Port 番号>.serviceで管理されています。ロール名の確認の仕方は次項の「コンポーネントやノードを指定して操作する」で説明します。

自動起動の設定はsystemctl is-enabledコマンドで確認でき、デフォルトで有効になっています。

自動起動設定が有効か確認

```
$ systemctl is-enabled tidb-4000.service
enabled
```

TiUPから無効化を実行すると自動起動の設定がdisabledに変わります。

OS起動時にプロセス自動起動を抑制

```
$ tiup cluster disable tidb-sample
...
(省略)
...
Disabled cluster `tidb-sample` successfully
```

自動起動設定が無効か確認

```
$ systemctl is-enabled tidb-4000.service
disabled
```

予期せぬOS再起動が発生した際に、調査が終わるまではプロセスを自動で起動させたくないといった場合は自動起動を無効化しておくことをおすすめします。

再度自動起動を有効にするにはtiup cluster enableを実行します。

OS起動時にプロセス自動起動を有効化

```
$ tiup cluster enable tidb-sample
...
(省略)
...
Enabled cluster `tidb-sample` successfully
```

コンポーネントやノードを指定して操作する

TiDB全体ではなく一部のコンポーネントを指定してさまざまな操作ができます。一部のコンポーネントを指定する場合は--roleオプションを使用します。1台だけ指定して操作する場合は--nodeオプションを使用します。--roleオプションで指定するロールはtiup cluster displayコマンドの実行結果で表示されるRoleの値を使用してください。たとえばTiDBサーバーのロールはtidbになります。

注10 <https://systemd.io/>

ノードを指定して停止

```
$ tiup cluster stop tidb-sample --node 10.0.0.6:4000
```

Prometheusを再起動する場合は次のように実行します。

Prometheusを再起動

```
$ tiup cluster restart tidb-sample --role prometheus
```

--roleオプションと--nodeオプションはカンマ区切りで複数の値を指定できます。

複数ノードを指定して停止

```
$ tiup cluster stop tidb-sample --node 10.0.0.6:4000,10.0.0.7:4000
```

PrometheusとGrafanaを再起動

```
$ tiup cluster restart tidb-sample --role prometheus,grafana
```

両オプションは同時に指定しても問題ありません。

両方のオプションを指定して停止

```
$ tiup cluster stop tidb-sample --node 10.0.0.6:4000 --role prometheus,grafana
```

操作の履歴および操作内容の詳細を確認する

tiup clusterコマンドによる操作の履歴は保存されています。履歴の一覧はtiup cluster auditコマンドで表示します。

操作の履歴一覧

```
$ tiup cluster audit
ID          Time          Command
--          -
gxRJZ3R84dW 2024-11-14T20:53:22Z /home/tidb/.tiup/components/cluster ...
gxRNQxB1nfy 2024-11-14T21:50:59Z /home/tidb/.tiup/components/cluster ...
...
(省略)
...
```

各操作の詳細を確認する場合は表示されたIDを指定します。

操作履歴の詳細

```
$ tiup cluster audit gxRJZ3R84dW

-----
- OPERATION TIME: 2024-11-14T20:53:22 -
-----

%2Fhome%2Ftidb%2F.tiup%2Fcomponents%2Fcluster%2Fv1.16.1%2Ftiup-cluster list
2024-11-14T20:53:22.797Z INFO Execute command ...
```

```
2024-11-14T20:53:22.797Z  DEBUG  Environment variables      ...
2024-11-14T20:53:22.802Z  DEBUG  Initialize repository finished ...
2024-11-14T20:53:22.802Z  INFO   Execute command finished      ...
```

▶ 操作履歴の内容を再現する

操作履歴を再現したい場合は `tiup cluster replay` コマンドを実行します。

操作の再現

```
$ tiup cluster replay gy8Jvrz65yw
Will replay the command `tiup cluster upgrade tidb-sample v8.5.0`
Do you want to continue? [y/N]: (default=N)
```

問題無ければ `y` を入力して実行します。確認は `--yes` オプションを使用することで省略可能です。

▶ 操作履歴を削除する

操作履歴を削除する場合は `tiup cluster audit cleanup` コマンドを実行します。デフォルトでは60日より前の操作記録が削除されます。保存日数は `--retain-days` オプションで変更可能です。

60日より前の操作記録の削除

```
$ tiup cluster audit cleanup
Audit logs before 2024-09-24T08:03:07 will be deleted!
Files to be deleted are:
/home/tidb/.tiup/storage/cluster/audit/gxRJZ3R84dW
...
(省略)
...
/home/tidb/.tiup/storage/cluster/audit/gy8xcQq6r49
Total count: 204
Total size: 24.67 MB
Do you want to continue? [y/N]:(default=N)
```

15日間より古い操作記録の削除

```
$ tiup cluster audit cleanup --retain-days 15
```

問題無ければ `y` を入力して消去します。確認は `--yes` オプションを使用することで省略可能です。

3-8

TiDBのバージョンアップ

TiDBは開発が活発なソフトウェアです。新しいバージョンやパッチバージョンの更新も頻繁に行われています(バージョンの履歴はリリースタイムライン^{注11}から確認できます)。構築したTiDBはバージョンアップせずにそのまま放置してしまうと、使用しているバージョンがサポート期間を過ぎてしまうリスクが発生します。リリースサポートポリシー^{注12}を確認して、期間が過ぎる前にバージョンアップするようにしてください。

TiUPを使用してTiDBをオンラインでバージョンアップできます。ダウングレードまたはロールバックは執筆時点ではサポートされていません。

ダウングレードはできない

```
$ tiup cluster upgrade tidb-sample v8.4.0
Error: please specify a higher or equle version than v8.5.0
```

なお、`--ignore-version-check`オプションを指定することでバージョンアップ前に行われるバージョンのチェックをスキップして強制的にダウングレードできますが、動作は保証されていないので注意してください。

本節では8.4.0から8.5.0へのバージョンアップを行います。

バージョンアップ前に確認すべきこと

バージョンアップを行う前に、各バージョンの互換性の変更をよく確認してください。互換性の変更はリリースノート内の各バージョン別のページ内にあります。8.5.0の互換性の変更は次のURLになります。

- <https://docs.pingcap.com/tidb/stable/release-8.5.0#compatibility-changes>

その他バージョンアップする際の注意点や制約についてはドキュメントのUpgrade TiDB Using TiUP^{注13}を参照してください。

オンラインでバージョンアップする

TiUPを使用したバージョンアップは、デフォルトではオンラインで行われます。バージョンアップ

注11 <https://docs.pingcap.com/tidb/stable/release-timeline>

注12 <https://www.pingcap.com/tidb-release-support-policy/>

注13 <https://docs.pingcap.com/tidb/stable/upgrade-tidb-using-tiup>

は各サーバーごとに行われますので、TiDBは変わらずサービスを提供し続けることが可能です。ただし、TiDB クラスターをバージョンアップする際にTiDBサーバーに接続しているクライアントは切断されますので注意してください。クライアントで再接続する実装をしていれば問題ありませんが、そうではない場合はTiProxyの使用を検討してください。

バージョンアップする際は新しいバージョンを指定してtiup cluster upgradeコマンドを実行します。現在使用しているバージョンはtiup cluster displayコマンドやtiup cluster listコマンドで確認すると良いでしょう。

現在のバージョンを確認

```
$ tiup cluster display tidb-sample-8-4-0
Cluster type:      tidb
Cluster name:      tidb-sample-8-4-0
Cluster version:   v8.4.0
...
(省略)
...
```

バージョンアップ前の確認

```
$ tiup cluster upgrade tidb-sample-8-4-0 v8.5.0
Before the upgrade, it is recommended to read the upgrade guide at
https://docs.pingcap.com/tidb/stable/upgrade-tidb-using-tiup
and finish the preparation steps.
This operation will upgrade tidb v8.4.0 cluster tidb-sample-8-4-0 to v8.5.0:
will upgrade and restart component "      pd" to "v8.5.0",
will upgrade and restart component "      tikv" to "v8.5.0",
will upgrade and restart component "      tidb" to "v8.5.0",
will upgrade and restart component "      prometheus" to "v8.5.0",
will upgrade and restart component "      grafana" to "v8.5.0",
will upgrade component "node-exporter" to "",
will upgrade component "blackbox-exporter" to "".
Do you want to continue? [y/N]:(default=N)
```

バージョンアップする前に確認を求められますので、問題なければyを入力します。確認は--yes オプションを使用することで省略可能です。

正常にバージョンアップが終了すると「Upgraded cluster '<クラスター名>' successfully」が表示されます。

バージョンアップ終了

```
...
(省略)
...
Upgraded cluster `tidb-sample-8-4-0` successfully
```

新しくなったバージョンを確認します。

現在のバージョンを確認

```
$ tiup cluster display tidb-sample-8-4-0
Cluster type:      tidb
Cluster name:      tidb-sample-8-4-0
Cluster version:   v8.5.0
...
(省略)
...
```

バージョンアップ時には、ストレージクラスターのTiKVのデータアクセスに影響がないようにリージョンのリーダーを移しながら行われます。リージョンの数が多い場合はリーダーの移動が制限時間内に収まらない可能性があります。その場合は制限時間を伸ばしてTiUPを実行してください。デフォルトでは制限時間は600秒です。--transfer-timeout オプションで時間を変更できます。この設定はPDサーバーのリーダー切り替えの時間制限でも使用されます。

PDとTiKVのリーダー変更の時間制限を伸ばす

```
$ tiup cluster upgrade tidb-sample-8-4-0 v8.5.0 --transfer-timeout 1800
```

オフラインでバージョンアップする

TiUPはオンラインバージョンアップだけではなく、停止されたクラスターに対するバージョンアップにも対応しています。サービス停止のメンテナンス時間がしっかり確保できる場合にはオフラインでのバージョンアップを検討してください。

オフラインバージョンアップをするためにクラスターを停止します。

クラスターの停止

```
$ tiup cluster stop tidb-sample-8-4-0
...
(省略)
...
Stopped cluster `tidb-sample-8-4-0` successfully
```

クラスターが停止したら--offline オプションを指定してバージョンアップを実行します。

オフラインでバージョンアップ

```
$ tiup cluster upgrade tidb-sample-8-4-0 v8.5.0 --offline
```

正常にバージョンアップが終了すると「Upgraded cluster '<クラスター名>' successfully」が表示されます。

バージョンアップ終了

```
...
(省略)
```

```
...
```

```
Upgraded cluster `tidb-sample-8-4-0` successfully
```

バージョンアップが終わってもクラスターは停止状態のままになります。

なお、停止されたクラスターに対して`--offline`オプションを指定せずにバージョンアップを行った場合、バージョンアップは失敗しますので注意してください。

3-9

クラスターメタ情報のバックアップ

クラスターのメタ情報は`/home/<ユーザー>/.tiup/storage/cluster/clusters/<クラスター名>`ディレクトリに格納されています。このメタ情報はTiUPによるさまざまな操作をする際に参照、または更新されます。クラスターのメタ情報をバックアップする場合は次のように実行します。

tidb-sampleクラスターのメタ情報のバックアップ

```
$ tiup cluster meta backup tidb-sample
successfully backup meta of cluster tidb-sample on tiup...07-39-11Z.tar.gz
```

生成されるバックアップファイルはデフォルトでは`tiup-cluster_<クラスター名>_metabackup_<日付>.tar.gz`となります。`--file`オプションを指定することで任意のファイル名で保存できます。

ファイル名を指定してメタ情報をバックアップ

```
$ tiup cluster meta backup tidb-sample \
--file tidb-sample-backup-$(date +%Y%m%d-%H%M%S).tar.gz
successfully backup ... tidb-sample-backup-20241114-074527.tar.gz
```

生成されたバックアップファイルは、TiUPを実行するサーバーとは別のサーバーやバックアップ用のオブジェクトストレージのバケットに定期的に保存しておくのがよいでしょう。万が一普段TiUPを実行しているサーバーが何らかの理由で失われてしまった場合、既存のクラスターに対する操作がTiUPから実行できなくなってしまうです。

メタ情報をバックアップから復元する際には`tiup cluster meta restore`コマンドを実行します。バックアップを確認するために、既存のクラスターのメタ情報を削除します(テスト目的以外では実行しないようにしてください)。

バックアップの確認のためメタ情報を削除

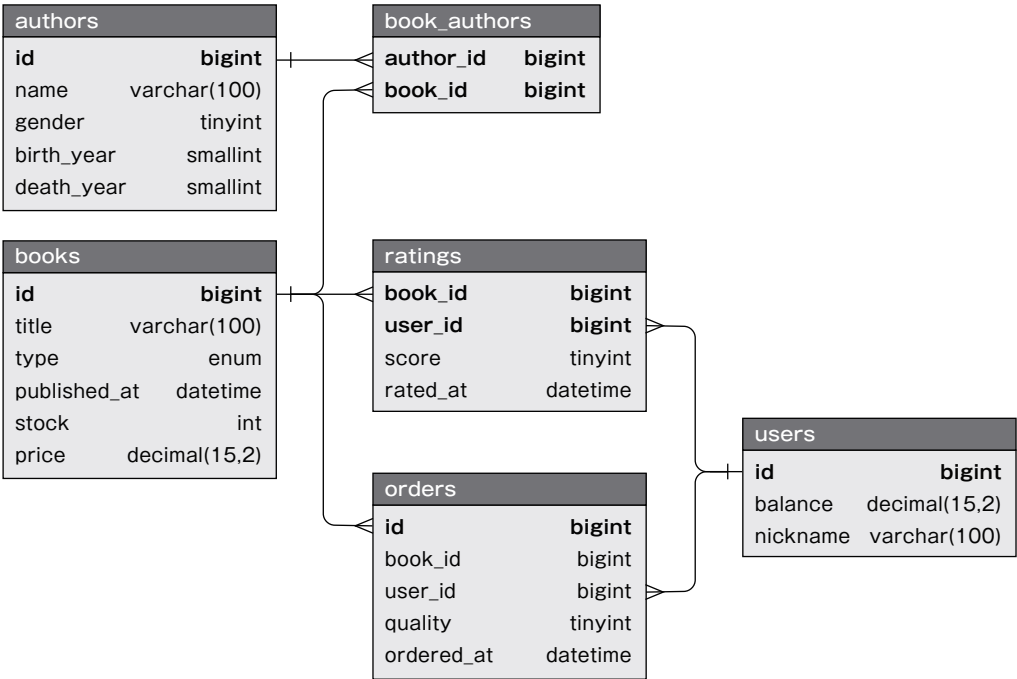
```
$ rm -rf /home/tidb/.tiup/storage/cluster/clusters/tidb-sample
```

`tiup cluster meta restore`コマンドを実行して復元します。

メタ情報を復元

```
$ tiup cluster meta restore tidb-sample \
```


図3.1 bookshopのER図



データセットの投入はtiup demo bookshop prepareコマンドで行います。スキーマの作成とデータの投入が行われます。

```
仮想書店アプリケーションのデータセット準備
$ tiup demo bookshop prepare
```

デフォルトではアドレス127.0.0.1の4000番ポートにrootユーザーで接続します。接続先を変更する場合は表3.5のオプションを使用してください。使用するデータベースも変更できます。

表3.5 接続先変更のオプション

オプション	デフォルト値
--host	127.0.0.1
--port	4000
--user	root
--password	無し
--database	bookshop

たとえば10.0.0.21に対して接続する場合の実行例は次のようになります。

接続先を指定してprepare

```
$ tiup demo bookshop prepare --host 10.0.0.21
```

実行するとデモ用のデータベースとテーブルが作成され、データが投入されます。

prepareの実行結果

```
INFO[0000] Creating the tables if not existed...      dataset=bookshop
INFO[0000] Creating table books.                      dataset=bookshop
INFO[0001] Creating table users.                     dataset=bookshop
INFO[0001] Creating table authors.                   dataset=bookshop
INFO[0002] Creating table book_authors.              dataset=bookshop
INFO[0002] Creating table orders.                    dataset=bookshop
INFO[0003] Creating table ratings.                   dataset=bookshop
INFO[0003] Finished creating tables!                  dataset=bookshop
INFO[0003] Clearing the old data...                   dataset=bookshop
INFO[0007] Loading users data...                     dataset=bookshop
INFO[0008] Loading books data...                     dataset=bookshop
INFO[0009] Loading authors data...                   dataset=bookshop
INFO[0010] Loading book authors data...              dataset=bookshop
INFO[0011] Loading book orders data...               dataset=bookshop
INFO[0033] Loading book ratings data...              dataset=bookshop
INFO[0056] Finished!                                dataset=bookshop
```

投入されたデータを確認します。次のSQLを実行することで投入されたデータ行数およびサイズを確認できます。

投入データの確認

```
mysql> SELECT
  CONCAT(table_schema,'.',table_name) AS 'Table Name',
  table_rows AS 'Number of Rows',
  CONCAT(ROUND(data_length/(1024*1024*1024),4),'G') AS 'Data Size',
  CONCAT(ROUND(index_length/(1024*1024*1024),4),'G') AS 'Index Size',
  CONCAT(ROUND((data_length+index_length)/(1024*1024*1024),4),'G') AS 'Total'
FROM
  INFORMATION_SCHEMA.TABLES
WHERE table_schema LIKE 'bookshop';
```

Table Name	Number of Rows	Data Size	Index Size	Total
bookshop.book_authors	20000	0.0003G	0.0003G	0.0006G
bookshop.orders	311232	0.0116G	0.0023G	0.0139G
bookshop.books	20000	0.0011G	0.0000G	0.0011G
bookshop.users	9988	0.0002G	0.0002G	0.0004G
bookshop.ratings	300000	0.0089G	0.0089G	0.0179G
bookshop.authors	20000	0.0009G	0.0000G	0.0009G

```
6 rows in set (0.02 sec)
```

INFORMATION_SCHEMAからデータを取得しますので、データ投入直後に実行すると正常な件数で出力されない場合があります。その場合はANALYZE TABLEを使用して統計情報を更新してください。

booksテーブルの統計情報の更新

```
mysql> ANALYZE TABLE bookshop.books;
Query OK, 0 rows affected, 1 warning (0.89 sec)
```

各テーブルに投入するデータ行数はオプションで指定できます。必要に応じてオプションを指定してデータ行数を調節してください。各オプションとデフォルト値を表3.6に示します。

表3.6 各テーブルのデータ行数の指定

オプション	テーブル名	デフォルト値
--users	users	10000
--authors	authors	20000
--books	books	20000
--orders	orders	300000
--ratings	ratings	300000

--drop-tables オプションを使用することで、各テーブルをDROPしてからデータの投入ができます。

一度DROPしてから再作成

```
$ tiup demo bookshop prepare --drop-tables
INFO[0000] Dropping the old tables... dataset=bookshop
INFO[0000] Dropping table orders. dataset=bookshop
INFO[0000] Dropping table ratings. dataset=bookshop
INFO[0001] Dropping table book_authors. dataset=bookshop
INFO[0001] Dropping table authors. dataset=bookshop
INFO[0002] Dropping table users. dataset=bookshop
INFO[0002] Dropping table books. dataset=bookshop
INFO[0003] Creating the tables if not existed... dataset=bookshop
INFO[0003] Creating table books. dataset=bookshop
INFO[0003] Creating table users. dataset=bookshop
INFO[0004] Creating table authors. dataset=bookshop
INFO[0004] Creating table book_authors. dataset=bookshop
INFO[0005] Creating table orders. dataset=bookshop
INFO[0005] Creating table ratings. dataset=bookshop
INFO[0006] Finished creating tables! dataset=bookshop
INFO[0006] Clearing the old data... dataset=bookshop
INFO[0009] Loading users data... dataset=bookshop
INFO[0010] Loading books data... dataset=bookshop
INFO[0011] Loading authors data... dataset=bookshop
INFO[0013] Loading book authors data... dataset=bookshop
INFO[0013] Loading book orders data... dataset=bookshop
INFO[0034] Loading book ratings data... dataset=bookshop
INFO[0055] Finished! dataset=bookshop
```

3-II

まとめ

本章では TiUP をインストールする方法、そして TiDB Playground および TiDB に対するさまざまな操作方法を説明しました。TiUP は TiDB の管理に便利なツールであり、TiDB の運用を効率化できます。ここでは記載しきれない機能も多くありますので、ドキュメントの TiUP Documentation Map^{注14}を参照してください。次章では Kubernetes を使用した TiDB の構築方法を説明します。

注14 <https://docs.pingcap.com/tidb/stable/tiup-documentation-guide>

第 4 章

Kubernetes環境への TiDBの構築

TiDB Operatorを使った構築と操作

本章ではKubernetes^{注1}環境へのTiDBの構築方法について説明します。

Kubernetesはコンテナ化されたアプリケーションのデプロイメント、スケーリング、管理を行うためのプラットフォームです。Kubernetesはクラウドネイティブなアプリケーションのデプロイメントに最適とされており、TiDBもクラウドネイティブ指向で開発されていることもあり、Kubernetesとの相性は非常に良くKubernetes上のアプリケーションとして問題なく構築、運用できます。

Kubernetesは各種リソースをマニフェストファイル(YAML形式またはJSON形式のファイル)で定義し、`kubectl`コマンドを使用して適用します。この方式は宣言型の管理方式であり、あるべき状態を定義することでKubernetesがその状態に遷移するように制御します。

また、Kubernetesにはオペレーターという概念があります。オペレーターはKubernetes上でアプリケーションを管理するためのフレームワークと言ってもいいでしょう。オペレーターはデプロイメント、スケーリング、アップグレード、バックアップ、フェイルオーバー、構成変更など、アプリケーションのライフサイクルを管理します。TiDBにおいては、Kubernetes上でTiDBのライフサイクルを管理するためのオペレーターとしてTiDB Operator^{注2}が提供されており、TiDB Operatorを使用することでKubernetes上でTiDBを簡単に構築、運用できます。

本書ではKubernetesの詳細については触れませんが、Kubernetesの知識が無い方でも分かりやすいように説明することを心がけています。本章におけるKubernetesはv1.31.4を使用しています。

4-1

Kubernetes環境の準備

Kubernetes環境を構築する方法はいくつかありますが、本章では`kind`^{注3}および`minikube`^{注4}を使用する方法について説明します。どちらのソフトウェアもKubernetesクラスターを手軽に構築するという意味では違いはありませんが、`kind`はシンプルな使い方ができ、`minikube`は加えて多機能なアドオン機能を使用できます。また、`kind`がコンテナベースでKubernetesクラスターを構築するのに対し、`minikube`はコンテナ環境だけではなく、仮想サーバーを使用してKubernetesクラスターを構築できます。

動作検証目的であれば`kind`や`minikube`を使用しても問題はありますが、本番環境での運用を考える場合は各種クラウドプロバイダーのマネージドKubernetesサービスを利用するか、ドキュメントのGet Started with TiDB on Kubernetes内の「See also」セクション^{注5}に記載されている環境でKubernetes

注1 <https://kubernetes.io/>

注2 <https://github.com/pingcap/tidb-operator>

注3 <https://kind.sigs.k8s.io/>

注4 <https://minikube.sigs.k8s.io/>

注5 <https://docs.pingcap.com/tidb-in-kubernetes/stable/get-started#see-also>

を構築してください。もしくはTiDBのマネージドサービスであるTiDB Cloudの利用を検討してください。TiDB Cloudについては第5章で説明します。

kubectlをインストールする

kubectlコマンドはKubernetesクラスターと通信するためのクライアントツールです。kubectlコマンドを使用することでKubernetesクラスターに対して各種操作が可能になります。本章におけるさまざまなKubernetesクラスターに対する操作はkubectlコマンドを使用して行います。そのため、まず最初にkubectlコマンドをインストールする必要があります。

Linux環境の場合は次のようにインストールします。curlコマンドを使用してバイナリーを入手し、installコマンドを使用して任意の場所(/usr/local/binディレクトリなど)に配置します。バイナリーの入手先のURLにアクセスするとリダイレクトされますので、curlコマンドで-Lオプションを使用してリダイレクト先を追従し、-Oオプションを使用してファイル名をそのまま保存します(任意のファイル名に変更して保存する場合は小文字の-oオプションを使用します)。installコマンドは対象ファイルのオーナー(-oオプション)、グループ(-gオプション)やパーミッションのモード(-mオプション)の指定が可能です。

Linux環境へのインストール

```
$ curl -LO "https://dl.k8s.io/release/v1.31.4/bin/linux/amd64/kubectl"
$ sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

$ kubectl version --client
Client Version: v1.31.4
Kustomize Version: v5.4.2
```

kubectlコマンドのバージョンはKubernetesクラスターのバージョンと合わせることをおすすめします。

その他の環境へのインストール方法はKubernetesのドキュメントのツールのインストール^{注6}を参照してください。

kindを使用してKubernetes環境を準備する

kindはDocker^{注7}またはPodman^{注8}を使用してローカル環境でKubernetesクラスターを構築するためのツールです。kindを使用することでローカル環境にKubernetesクラスターを簡単に構築できます。kindを使用する場合は別途DockerまたはPodmanがインストールされていることが前提となります。

注6 <https://kubernetes.io/ja/docs/tasks/tools/#kubectl>

注7 <https://www.docker.com/>

注8 <https://podman.io/>

Dockerのインストール方法はDockerのドキュメントのGet Docker^{注9}を参照してください。Podmanのインストール方法はPodmanのドキュメントのPodman Installation Instructions^{注10}を参照してください。

執筆時点でのkindの最新バージョンはv0.26.0になります。Linux環境へのkindのインストール方法は次の通りです。

kindのインストール

```
$ curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.26.0/kind-linux-amd64
$ sudo install -o root -g root -m 0755 kind /usr/local/bin/kind

$ kind --version
kind version 0.26.0
```

その他の環境へのインストール方法はkindのドキュメントのQuick Start^{注11}を参照してください。

kind create cluster コマンドを使用することでKubernetes クラスターを構築できます。その際に構成ファイルを使用することで、Kubernetes のバージョンやノード数を指定できます。次の例ではKubernetes のバージョンをv1.31.4として、コントロールプレーンを1台、ワーカーノードを3台構成するクラスターを構築しています。--config オプションを使用して構成ファイルを指定し、--name オプションを使用してクラスター名を指定しています。

node3-1.31.4.yaml

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
  image: kindest/node:v1.31.4@sha256:2cb39f7295fe7eafee0842b1052a599a4fb0f8bcf3f83d96c7f4864c357c6c30
- role: worker
  image: kindest/node:v1.31.4@sha256:2cb39f7295fe7eafee0842b1052a599a4fb0f8bcf3f83d96c7f4864c357c6c30
- role: worker
  image: kindest/node:v1.31.4@sha256:2cb39f7295fe7eafee0842b1052a599a4fb0f8bcf3f83d96c7f4864c357c6c30
- role: worker
  image: kindest/node:v1.31.4@sha256:2cb39f7295fe7eafee0842b1052a599a4fb0f8bcf3f83d96c7f4864c357c6c30
```

kindを使用したKubernetesクラスターの構築

```
$ kind create cluster --config node3-1.31.4.yaml --name k8s-tidb
Creating cluster "k8s-tidb" ...
✓ Ensuring node image (kindest/node:v1.31.4) 📦
✓ Preparing nodes 📦 📦 📦
✓ Writing configuration 📄
✓ Starting control-plane 🚦
✓ Installing CNI 🛠️
```

注9 <https://docs.docker.com/get-started/get-docker/>

注10 <https://podman.io/docs/installation>

注11 <https://kind.sigs.k8s.io/docs/user/quick-start/#installation>

```

✓ Installing StorageClass 🖨️
✓ Joining worker nodes 🤖
Set kubectl context to "kind-k8s-tidb"
You can now use your cluster with:

kubectl cluster-info --context kind-k8s-tidb

Have a nice day! 🙌

```

構築が完了すると kubectl コマンドを使用して Kubernetes クラスターにアクセスするためのコンテキスト (kind-k8s-tidb) が設定されます。このコンテキストを使用して kind で構築した Kubernetes クラスターにアクセスできます。

Kubernetes クラスターの情報確認

```

$ kubectl cluster-info --context kind-k8s-tidb
Kubernetes control plane is running at https://127.0.0.1:49715
CoreDNS is running at ...

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

$ kubectl get node --context kind-k8s-tidb

```

NAME	STATUS	ROLES	AGE	VERSION
k8s-tidb-control-plane	Ready	control-plane	8m14s	v1.31.4
k8s-tidb-worker	Ready	<none>	8m3s	v1.31.4
k8s-tidb-worker2	Ready	<none>	8m3s	v1.31.4
k8s-tidb-worker3	Ready	<none>	8m3s	v1.31.4

--context オプションを毎回使用するのが面倒な場合、kubectl config use-context を使用してデフォルトのコンテキストを設定できます。

デフォルトのコンテキストの設定

```

$ kubectl config use-context kind-k8s-tidb
Switched to context "kind-k8s-tidb".

$ kubectl get node

```

NAME	STATUS	ROLES	AGE	VERSION
k8s-tidb-control-plane	Ready	control-plane	9m17s	v1.31.4
k8s-tidb-worker	Ready	<none>	9m6s	v1.31.4
k8s-tidb-worker2	Ready	<none>	9m6s	v1.31.4
k8s-tidb-worker3	Ready	<none>	9m6s	v1.31.4

作成した Kubernetes クラスターを削除する場合は kind delete cluster コマンドを使用します。--name オプションでクラスター名を指定してください。

Kubernetes クラスターの削除

```

$ kind delete cluster --name k8s-tidb

```

minikubeを使用してKubernetes環境を準備する

minikubeはローカル環境でKubernetesクラスターを構築するためのツールです。kindと同様にローカル環境でKubernetesクラスターを簡単に構築できます。minikubeは仮想化ソフトウェア(VirtualBox、KVM、Dockerなど)をドライバーとして使用してKubernetesクラスターを構築します。また、アドオン機能が提供されており、モニタリング、ログ収集、Ingressコントローラーなどの機能を簡単に有効化できます。アドオンの種類はminikubeのドキュメントのAddons^{注12}から確認できます。

minikubeを使用するためには、使用するドライバーがインストールされていることが前提となります。ドライバーの一覧とインストール方法はminikubeのドキュメントのDrivers^{注13}を参照してください。

執筆時点でのminikubeの最新バージョンはv1.34.0になります。Linux環境へのminikubeのインストール方法は次の通りです。

minikubeのインストール

```
$ curl -o minikube \
-L https://storage.googleapis.com/minikube/releases/v1.34.0/minikube-linux-amd64
$ sudo install -o root -g root -m 0755 minikube /usr/local/bin/minikube

$ minikube version
minikube version: v1.34.0
commit: 210b148df93a80eb872ecbeb7e35281b3c582c61
```

その他の環境へのminikubeのインストール方法はminikubeのドキュメントのminikube start^{注14}を参照してください。

minikube startコマンドでKubernetesクラスターを構築できます。その際に--kubernetes-versionオプションを使用してKubernetesのバージョンを指定できます。次の例ではKubernetesのバージョンをv1.31.4として、コントロールプレーンを1台、ワーカーノードを3台構成するKubernetesクラスターを構築します。ノードの数は合計4台になりますので、--nodesオプションで4を指定します。--driverオプションでdockerを指定していますが、自身の環境に合わせて変更してください。

minikubeを使用したKubernetesクラスターの構築

```
$ minikube start --kubernetes-version v1.31.4 --nodes 4 --driver docker
😄 minikube v1.34.0 on Rocky 9.4 (amd64)
❗ Specified Kubernetes version 1.31.4 is newer than the newest supported version: ...
❗ Specified Kubernetes version 1.31.4 not found in Kubernetes version list
🔍 Searching the internet for Kubernetes version...
✅ Kubernetes version 1.31.4 found in GitHub version list
```

注12 <https://minikube.sigs.k8s.io/docs/handbook/addons/>

注13 <https://minikube.sigs.k8s.io/docs/drivers/>

注14 <https://minikube.sigs.k8s.io/docs/start/>

```

👉 Using the docker driver based on user configuration
👉 Using Docker driver with root privileges
👉 Starting "minikube" primary control-plane node in "minikube" cluster
...
(省略)
...
👉 Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default

```

minikubeはkubectlを別途インストールしなくてもminikube kubectl コマンドを使用してKubernetes クラスターにアクセスできます。

minikubeからkubectlを使用

```

$ minikube kubectl -- get node

```

NAME	STATUS	ROLES	AGE	VERSION
minikube	Ready	control-plane	102s	v1.31.4
minikube-m02	Ready	<none>	72s	v1.31.4
minikube-m03	Ready	<none>	45s	v1.31.4
minikube-m04	Ready	<none>	20s	v1.31.4

作成したKubernetesクラスターを停止する場合はminikube stop コマンドを実行します。

Kubernetesクラスターの停止

```

$ minikube stop
👉 Stopping node "minikube-m04" ...
🔴 Powering off "minikube-m04" via SSH ...
👉 Stopping node "minikube-m03" ...
🔴 Powering off "minikube-m03" via SSH ...
👉 Stopping node "minikube-m02" ...
🔴 Powering off "minikube-m02" via SSH ...
👉 Stopping node "minikube" ...
🔴 Powering off "minikube" via SSH ...
🔴 4 nodes stopped.

```

削除する場合はminikube delete コマンドを実行してください。起動している場合でも削除できます。

Kubernetesクラスターの削除

```

$ minikube delete
🔴 Deleting "minikube" in docker ...
🔴 Deleting container "minikube" ...
🔴 Deleting container "minikube-m02" ...
🔴 Deleting container "minikube-m03" ...
🔴 Deleting container "minikube-m04" ...
🔴 Removing /home/tidb/.minikube/machines/minikube ...
🔴 Removing /home/tidb/.minikube/machines/minikube-m02 ...
🔴 Removing /home/tidb/.minikube/machines/minikube-m03 ...
🔴 Removing /home/tidb/.minikube/machines/minikube-m04 ...
👤 Removed all traces of the "minikube" cluster.

```

4-2

TiDB Operator のインストール

Kubernetes クラスターの環境ができれば、次に TiDB Operator をインストールします。インストールする TiDB Operator のバージョンは執筆時点での最新バージョンである 1.6.1 を使用します。TiDB Operator のバージョンとサポートしている TiDB のバージョンは表 4.1 の通りになりますので、自身の環境に合わせて TiDB Operator のバージョンを選択してください。

表 4.1 TiDB Operator のバージョンとサポートしている TiDB のバージョン

TiDB バージョン	TiDB Operator バージョン
TiDB $\geq$ 8.0	1.6 (推奨)、1.5
$7.1 \leq$ TiDB $<$ 8.0	1.5 (推奨)、1.4
$6.5 \leq$ TiDB $<$ 7.1	1.5、1.4 (推奨)、1.3
$5.4 \leq$ TiDB $<$ 6.5	1.4、1.3 (推奨)
$5.1 \leq$ TiDB $<$ 5.4	1.4、1.3 (推奨)、1.2
$3.0 \leq$ TiDB $<$ 5.1	1.4、1.3 (推奨)、1.2、1.1
$2.1 \leq$ TiDB $<$ v3.0	1.0 (サポート終了)

カスタムリソース定義をインストールする

TiDB Operator をインストールする前に、TiDB Operator が使用するカスタムリソース定義 (CRD) をインストールする必要があります。カスタムリソース定義は Kubernetes の拡張機能であり、Kubernetes に新しくカスタムリソース (CR) を追加するために使用されます。カスタムリソースについての詳細は Kubernetes のドキュメントのカスタムリソース^{注15}を参照してください。

カスタムリソース定義は次のコマンドでインストールできます。カスタムリソース定義のマニフェストファイルは GitHub のリポジトリから取得します。使用する TiDB Operator のバージョン用のマニフェストファイルを指定してください。

なお、カスタムリソース定義のインストール時に `kubectl apply` コマンドは使用しないでください。
`metadata.annotations: Too long` のエラーが表示されて正常にインストールできません。

CRD のインストール

```
$ kubectl create \
  -f https://raw.githubusercontent.com/pingcap/tidb-operator/v1.6.1/manifests/crd.yaml
customresourcedefinition.apiextensions.k8s.io/backups.pingcap.com created
customresourcedefinition.apiextensions.k8s.io/backupschedules.pingcap.com created
customresourcedefinition.apiextensions.k8s.io/dmclusters.pingcap.com created
```

注 15 <https://kubernetes.io/ja/docs/concepts/extend-kubernetes/api-extension/custom-resources/>


```
customresourcedefinition.apiextensions.k8s.io/restores.pingcap.com created
customresourcedefinition.apiextensions.k8s.io/tidbclusterautoscalers.pingcap.com created
customresourcedefinition.apiextensions.k8s.io/tidbclusters.pingcap.com created
customresourcedefinition.apiextensions.k8s.io/tidbdashboards.pingcap.com created
customresourcedefinition.apiextensions.k8s.io/tidbinitializers.pingcap.com created
customresourcedefinition.apiextensions.k8s.io/tidbmonitors.pingcap.com created
customresourcedefinition.apiextensions.k8s.io/tidbngmonitorings.pingcap.com created
```

TiDB Operator で使用するカスタムリソースは表 4.2 の通りです。

表 4.2 TiDB Operator が使用するカスタムリソース

カスタムリソース	短縮形	用途
Backup	bk	バックアップの定義
BackupSchedule	bks	バックアップスケジュールの定義
DMCluster	dc	DM クラスターの定義
Restore	rt	リストアの定義
TidbClusterAutoScaler	ta	TiDB クラスターオートスケーラーの定義 (コラム参照)
TidbCluster	tc	TiDB の定義
TidbDashboard	td	TiDB Dashboard の定義
TidbInitializer	ti	TiDB の初期化の定義
TidbMonitor	tm	モニタリングコンポーネントの定義
TidbNGMonitoring	tngm	NG モニタリングコンポーネントの定義

カスタムリソースに `TidbCluster` という単語が出てきますが、ここでは TiDB の各種コンポーネントで構成されている TiDB 全体を指します。TiDB のアーキテクチャ図に記載されている TiDB クラスター (コンピュート層のコンポーネント) とは異なりますので、ご注意ください。

カスタムリソースは `kubectl get` コマンドや `kubectl describe` コマンドで確認できます。各カスタムリソースの具体的な設定の仕方は後述します。たとえば `TidbCluster` の確認は次のように行います。

カスタムリソースの確認

```
$ kubectl get TidbCluster
```

短縮形を使用しても同様に確認できます。

短縮形を使用したカスタムリソースの確認

```
$ kubectl get tc
```

Kubernetes 内で定義されているカスタムリソース定義の一覧は次のコマンドで確認できます。

カスタムリソース定義の一覧表示

```
$ kubectl get crd
```

各定義の詳細は `kubectl describe` コマンドで確認できます。

カスタムリソース定義の詳細表示

```
$ kubectl describe crd tidbclusters.pingcap.com
```

TiDB クラスターオートスケーラーについて

column

TiDB クラスターオートスケーラー (TidbClusterAutoScaler) は過去に実装されましたが、執筆時点ではドキュメントから削除されています。よって使用はなるべく控えることをおすすめします。

Helm を使用して TiDB Operator をインストールする

TiDB Operator をインストールする方法はいくつかありますが、本項では Helm^{注16} を使用してインストールする方法について説明します。Helm は Kubernetes のパッケージマネージャーであり、Kubernetes のリソースをパッケージ化して簡単にインストールできます。Kubernetes リソースのテンプレートをパッケージ化したものは Helm チャートと呼ばれ、Helm リポジトリからダウンロードして使用します。Helm のインストール方法は Helm のドキュメントの Installing Helm^{注17} を参照してください。

Helm のインストールが完了したら Helm リポジトリを追加します。TiDB Operator の Helm チャートは PingCAP によって提供されていますので、まず始めに PingCAP 提供の Helm リポジトリを追加します。追加した後はリポジトリの情報を最新に更新するために `helm repo update` コマンドを実行します。

Helm リポジトリの追加と更新

```
$ helm repo add pingcap https://charts.pingcap.org/
"pingcap" has been added to your repositories

$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "pingcap" chart repository
Update Complete. 🍻Happy Helming! 🍻
```

インストール可能な Helm チャートの一覧は `helm search repo` コマンドで確認できます。先ほど追加した PingCAP の Helm リポジトリ内の Helm チャートの一覧を表示します。

PingCAP のリポジトリ内の Helm チャートの一覧表示

```
$ helm search repo pingcap
```

NAME	CHART VERSION	APP VERSION	DESCRIPTION
pingcap/br-federation	v1.6.1	v1.6.1	br-federation Helm chart for Kubernetes
pingcap/diag	v1.3.1	v1.3.1	clinic diag Helm chart for Kubernetes
pingcap/tidb-backup	v1.5.1		DEPRECATED ...
pingcap/tidb-cluster	v1.5.1		DEPRECATED ...
pingcap/tidb-drainer	v1.6.1		A Helm chart for TiDB Binlog drainer.
pingcap/tidb-lightning	v1.6.1		A Helm chart for TiDB Lightning
pingcap/tidb-operator	v1.6.1	v1.6.1	tidb-operator ...

注16 <https://helm.sh/>注17 <https://helm.sh/docs/intro/install/>

```
pingcap/tikv-importer v1.5.1 A Helm chart for TiKV Importer ...
pingcap/tikv-operator v0.1.0 v0.1.0 A Helm chart for Kubernetes ...
```

TiDB OperatorのHelmチャートはpingcap/tidb-operatorとして提供されています。

一覧では最新のバージョンのみ表示されますが、--versionsオプションを使用するとすべてのバージョンを確認できます。pingcap/tidb-operatorのすべてのバージョンを表示する例を次に示します。

すべてのバージョンを表示

```
$ helm search repo pingcap/tidb-operator --versions
NAME          CHART VERSION  APP VERSION  DESCRIPTION
pingcap/tidb-operator v1.6.1        v1.6.1       tidb-operator ...
pingcap/tidb-operator v1.6.0        v1.6.0       tidb-operator ...
pingcap/tidb-operator v1.5.4        v1.5.4       tidb-operator ...
pingcap/tidb-operator v1.5.3        v1.5.3       tidb-operator ...
...
(省略)
...
```

今回使用するTiDB Operatorのバージョンは1.6.1ですので、APP VERSIONがv1.6.1であるCHART VERSIONを指定してインストールします。CHART VERSIONはHelmチャート自体のバージョンで、APP VERSIONはHelmチャートによってインストールされるアプリケーションのバージョン(今回の例ではTiDB Operatorのバージョン)を指しています。TiDB OperatorのチャートはCHART VERSIONとAPP VERSIONが同じですが、他のHelmチャートでは異なる場合がありますので注意してください。

インストールはhelm installコマンドを使用します。--versionオプションを使用してバージョンを指定します。helm installコマンドはKubernetesクラスターにチャートをインストールしますが、kubectlコマンドと同様に--kube-contextオプションを使用してコンテキストを切り替えられますので、必要に応じて指定してください。

TiDB Operatorが使用するNamespace(名前空間)を指定する場合は--namespaceオプションを使用します。本節の手順ではTiDB Operatorをtidb-adminのNamespaceにインストールします。--namespaceオプションを使用しなかった場合はデフォルトのdefaultというNamespaceが使用されます。

TiDB Operatorのインストール

```
$ kubectl create namespace tidb-admin
namespace/tidb-admin created

$ helm install tidb-operator pingcap/tidb-operator --version v1.6.1 --namespace tidb-admin
NAME: tidb-operator
LAST DEPLOYED: Thu Nov 28 23:55:53 2024
NAMESPACE: tidb-admin
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Make sure tidb-operator components are running:
```

```
kubectl get pods --namespace tidb-admin -l app.kubernetes.io/instance=tidb-operator
```

なお、指定した namespace が存在しない場合は自動的に作成する `--create-namespace` オプションも使用できます。`--create-namespace` オプションを使用する場合は namespace を事前に作成する必要はありません。

TiDB Operator のインストール (Namespace の指定と自動作成)

```
$ helm install tidb-operator pingcap/tidb-operator --version v1.6.1 \
  --namespace tidb-admin --create-namespace
```

表示されたメッセージに従って、TiDB Operator の Pod が正常に起動されているか確認します。

TiDB Operator の Pod の確認

```
$ kubectl get pod --namespace tidb-admin --selector app.kubernetes.io/instance=tidb-operator
```

NAME	READY	STATUS	RESTARTS	AGE
tidb-controller-manager-bc986bd9-cvbc4	1/1	Running	0	43s

STATUS が Running になっていれば正常に起動されています。

正常に起動しない場合は `kubectl describe` コマンドや `kubectl logs` コマンドを使用して Pod の詳細を確認してください。

Pod の詳細確認

```
$ kubectl describe pods --namespace tidb-admin tidb-controller-manager-bc986bd9-cvbc4
$ kubectl logs --namespace tidb-admin tidb-controller-manager-bc986bd9-cvbc4
```

▶ TiDB Operator のインストール時に設定可能なオプション

Helm チャートをインストールする際にはさまざまなオプションを指定できます。たとえば TiDB Operator の Pod の数や CPU、メモリなどのリソースの割当量などを指定できます。次の例は TiDB Operator の Pod の数を 2 に指定し、冗長性を持たせる設定です。`--set` オプションを使用して設定します。

冗長性を持たせた TiDB Operator のインストール

```
$ helm install tidb-operator pingcap/tidb-operator --version v1.6.1 \
  --namespace tidb-admin --set controllerManager.replicas=2
```

```
$ kubectl get pod --namespace tidb-admin --selector app.kubernetes.io/instance=tidb-operator
```

NAME	READY	STATUS	RESTARTS	AGE
tidb-controller-manager-bc986bd9-4sqpn	1/1	Running	0	89s
tidb-controller-manager-bc986bd9-clbm2	1/1	Running	0	19m

TiDB Operator は複数の Pod が起動した場合は、そのうちの 1 つの Pod が TiDB Operator のリーダーとなり処理を行います。リーダーがダウンした場合は他の Pod がリーダーに昇格して処理を続けます。どの Pod がリーダーを担当しているかを確認するには Pod のログを確認してください。successfully acquired lease <namespace>/tidb-controller-manager というメッセージがログに表示されている Pod

がリーダーとして動作しています。

TiDB Operatorのログの確認

```
$ kubectl logs --namespace tidb-admin tidb-controller-manager-bc986bd9-4sqpn
...
I0114 ... successfully acquired lease tidb-admin/tidb-controller-manager
...
```

その他のすべての設定可能な項目はhelm show values コマンドで確認できますので、必要に応じて設定してください。

設定可能な項目の確認

```
$ helm show values pingcap/tidb-operator --version v1.6.1
# Default values for tidb-operator

# clusterScoped is whether tidb-operator should manage ...
# Also see rbac.create, controllerManager.serviceAccount, ...
clusterScoped: true

# Also see clusterScoped and controllerManager.serviceAccount
rbac:
  create: true

# timezone is the default system timzone
timezone: UTC
...
(省略)
...
```

4-3

TiDB Operatorを使用したTiDBの構築

TiDB Operatorが問題なく起動していることを確認できたら、次にTidbClusterリソースを定義してTiDBをKubernetesに構築します。TidbClusterリソースを作成すると、その内容に従ってTiDB OperatorがTiDBを構築します。

マニフェストファイルを作成して適用する

TidbClusterリソースのマニフェストファイルを作成します(リスト4.1)。ファイルは手動で作成する必要がありますが、TiDB OperatorのGitHubのリポジトリに公開されているマニフェスト^{注18}の例を

注18 <https://github.com/pingcap/tidb-operator/tree/master/examples>

参考にしても問題ありません。

リスト4.1 TidbClusterマニフェストの例

```
tidb-sample-cluster.yaml
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: tidb-sample
spec:
  version: v8.5.0 ①
  timezone: UTC
  pd:
    baseImage: pingcap/pd
    replicas: 1 ②
    requests:
      storage: "1Gi" ③
    config: {} ⑤
  tikv:
    baseImage: pingcap/tikv
    replicas: 1
    requests:
      storage: "1Gi"
    config: {}
  tidb:
    baseImage: pingcap/tidb
    replicas: 1
    service: ④
    type: ClusterIP
    config: {}
```

TidbCluster リソースの定義にはさまざまな設定項目がありますが、最小限の設定でTiDBを構築する方法について説明します。

.spec.versionでTiDBのバージョンを指定します(リスト4.1の①)。`.spec.pd`、`.spec.tikv`、`.spec.tidb`ではそれぞれPDクラスター、ストレージクラスター、TiDBクラスターの設定を指定します。`.spec.*.replicas`でPodの数を指定(リスト4.1の②)し、`.spec.*.requests.storage`でストレージの割当量を指定します(リスト4.1の③)。PDクラスターおよびストレージクラスターはストレージの割当量を指定する必要がありますが、TiDBクラスターは指定しなくても起動します。TiDBクラスターは`.spec.tidb.service`を設定しておきます(リスト4.1の④)。設定しておくことでTiDBクラスターのServiceリソースが作成されます。これは後の動作確認で使用するためです。`.spec.*.config`では各種コンポーネントの設定を指定できますが、現時点では空にしておきます(リスト4.1の⑤)。

作成したマニフェストファイルを使用して `kubectl apply` コマンドでTidbClusterリソースを作成します。

TidbClusterの作成

```
$ kubectl apply -f tidb-sample-cluster.yaml
tidbcluster.pingcap.com/tidb-sample created
```

しばらくすると各コンポーネントが順次起動します。kubectl get コマンドでPodの状態を確認してください。使用したマニフェストファイルではreplicasを1に指定しているため、各コンポーネントが1つつ起動します。

Podの状態確認

```
$ kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
tidb-sample-discovery-768fcb86f5-lmsw7	1/1	Running	0	6m28s
tidb-sample-pd-0	1/1	Running	0	6m28s
tidb-sample-tidb-0	2/2	Running	0	5m43s
tidb-sample-tikv-0	1/1	Running	0	6m13s

tidb-sample-discovery-768fcb86f5-lmsw7以外のPodは、Pod名の末尾に0が付いていることが確認できます。0はPodのインデックスを表しており、Podの数が2以上の場合は1、2、3と増えていきます。これはPodがTiDB Operatorによって作成されたStatefulSetリソースにより管理されているためです。

StatefulSetの確認

```
$ kubectl get statefulset
```

NAME	READY	AGE
tidb-sample-pd	1/1	15m
tidb-sample-tidb	1/1	14m
tidb-sample-tikv	1/1	15m

StatefulSetについてはKubernetesのドキュメントのStatefulSet^{注19}を参照してください。

構築したTiDBへ接続する

TiDBの構築が完了したら、TiDBに接続して操作できます。さまざまな方法で接続できますので、本項ではいくつか方法を紹介します。なお、Kubernetesクラスター内でNetworkPolicy^{注20}リソースが設定されている場合、次に紹介する手順ではアクセスできない場合があります。その場合はNetworkPolicyリソースの設定を確認してください。

Kubernetesクラスター内からの接続

先ほど作成したtidb-sampleはマニフェストの定義で.spec.tidb.serviceを設定しました。この設定

注19 <https://kubernetes.io/ja/docs/concepts/workloads/controllers/statefulset/>

注20 <https://kubernetes.io/ja/docs/concepts/services-networking/network-policies/>

により、Kubernetes クラスター内に<metadata.name>-tidb という Service リソースが作成されています。

Serviceの確認

```
$ kubectl get service
```

NAME	TYPE	CLUSTER-IP	...	PORT(S)	AGE
kubernetes	ClusterIP	10.96.0.1	...	443/TCP	40h
tidb-sample-discovery	ClusterIP	10.96.70.41	...	10261/TCP,10262/TCP	45m
tidb-sample-pd	ClusterIP	10.96.182.67	...	2379/TCP	45m
tidb-sample-pd-peer	ClusterIP	None	...	2380/TCP,2379/TCP	45m
tidb-sample-tidb	ClusterIP	10.96.171.212	...	4000/TCP,10080/TCP	6m38s
tidb-sample-tidb-peer	ClusterIP	None	...	10080/TCP	44m
tidb-sample-tikv-peer	ClusterIP	None	...	20160/TCP	45m

tidb-sample-tidb という Service リソースが ClusterIP として 10.96.171.212 で作成されていることが確認できます。Kubernetes クラスター内部からこの tidb-sample-tidb の Service リソースに対して接続することで TiDB にアクセスできます。

Kubernetes クラスター内に mysql コマンドが実行できる Pod を作成し、TiDB に接続します。mysql:8.0 のコンテナイメージを kubectl run コマンドで起動します。コマンド実行のため、まずは Pod 内でシェルの起動します。

接続用のPodの作成

```
$ kubectl run mysql-client --image=mysql:8.0 -it --rm --restart=Never -- /bin/bash
bash-5.1#
```

mysql コマンドを使用して TiDB に接続します。接続先のホスト名に先ほど確認した tidb-sample-tidb を指定します。

TiDBへの接続

```
bash-5.1# mysql --user root --host tidb-sample-tidb --port 4000
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2371879542
Server version: 8.0.11-TiDB-v8.5.0 TiDB Server (Apache License 2.0) ...

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SELECT VERSION();
+-----+
| VERSION() |
+-----+
| 8.0.11-TiDB-v8.5.0 |
```



```
+-----+
1 row in set (0.00 sec)
```

なおKubernetesのServiceリソース名はFQDN(Fully Qualified Domain Name)が使用できます。リソース名の後に、<Namespace>.svc.cluster.localを付けることでKubernetes内から名前解決できます。ですのでtidb-sample-tidb.default.svc.cluster.localというFQDNを指定して接続する方法もあります。

TiDBへFQDNで接続

```
bash-5.1# mysql --user root --host tidb-sample-tidb.default.svc.cluster.local --port 4000
```

接続するPodと構築したTiDBが違うNamespaceにあるといった場合には、こちらのFQDNを使用して接続してください。同じNamespaceにある場合はServiceリソース名だけで接続できます。

KubernetesのServiceリソースは負荷分散機能を持っているため、接続先のTiDBコンポーネントのPodが複数ある場合は自動で振り分けられます。また、Podの台数が増減した場合は自動で振り分け先を変更します。次のように接続した後にserver_idを確認することで、自動で接続先が振り分けられていることが確認できます。

複数のTiDBコンポーネントがある場合の接続

```
bash-5.1# mysql --user root --host tidb-sample-tidb.default.svc.cluster.local --port 4000 \
-e 'SHOW STATUS like "server_id"'
+-----+
| Variable_name | Value |
+-----+
| server_id     | 72ad07f2-eb4f-4777-a9cc-45cf0dcf3799 |
+-----+

bash-5.1# mysql --user root --host tidb-sample-tidb.default.svc.cluster.local --port 4000 \
-e 'SHOW STATUS like "server_id"'
+-----+
| Variable_name | Value |
+-----+
| server_id     | b6ec61cd-135a-47b4-b9d2-f84cb3459c13 |
+-----+
```

Kubernetesクラスター外からの接続

Kubernetesクラスター外からTiDBに接続する場合は、作成されるServiceリソースの種別をLoadBalancerに変更するか、NodePortに変更して接続する必要があります。LoadBalancerを使用する場合はクラウドプロバイダーなどによって外部IPアドレスが割り当てられます。NodePortを使用する場合はKubernetesクラスターのノードのIPアドレスとポート番号を使用してアクセスします。

LoadBalancerを使用する場合はマニフェストファイルの内容を次のように変更します(リスト4.2の

①)。

リスト4.2 LoadBalancerを使用するマニフェストの例

```
tidb-sample-cluster.yaml
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: tidb-sample
spec:
  ...
  (省略)
  ...
  tidb:
    baseImage: pingcap/tidb
    replicas: 1
    config: {}
    service:
      type: LoadBalancer ❶
```

変更後のマニフェストファイルを使用して `kubectl apply` コマンドで `TidbCluster` リソースを更新します。

```
TidbClusterの更新
$ kubectl apply -f tidb-sample-cluster.yaml
tidbcluster.pingcap.com/tidb-sample configured
```

Service リソースの TYPE が `LoadBalancer` に変更されたことを確認します。外部の IP アドレスが `EXTERNAL-IP` として割り当てられていることが確認できます。

```
Serviceの確認
NAME                TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)        ...
tidb-sample-tidb    LoadBalancer 10.96.171.212 172.18.0.6    4000:30306/TCP, ...
```

その後は同じように外部IPアドレスを使用してTiDBに接続することができます。

```
外部IPアドレスを使用して接続
$ mysql --user root --host 172.18.0.6 --port 4000
```

各種クラウドプロバイダーのKubernetesサービスを使用している場合は、`EXTERNAL-IP`を取得することが難しいのですが、ローカル環境(kindやminikube)でKubernetesを使用している場合は`EXTERNAL-IP`が<pending>になることがあります。その場合はkindではcloud-provider-kind^{注21}を、minikubeの場合はminikube tunnel^{注22}を使用して`EXTERNAL-IP`を取得してください。もしくは次に説明するNodePortを使用して接続してください。

注21 <https://github.com/kubernetes-sigs/cloud-provider-kind>

注22 <https://minikube.sigs.k8s.io/docs/commands/tunnel/>

NodePortを使用した接続の仕方を説明します。先ほど使用したマニフェストファイル(tidb-sample-cluster.yaml)を編集して.spec.tidb.service.typeをNodePortに変更します(リスト4.3の❶)。

リスト4.3 NodePortを使用するマニフェストの例

```
tidb-sample-cluster.yaml
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: tidb-sample
spec:
  ...
  (省略)
  ...
  tidb:
    baseImage: pingcap/tidb
    replicas: 1
    config: {}
    service:
      type: NodePort  —❶
```

変更後のマニフェストファイルを使用してkubectl applyコマンドでTidbClusterリソースを更新します。

```
TidbClusterの更新
$ kubectl apply -f tidb-sample-cluster.yaml
tidbcluster.pingcap.com/tidb-sample configured
```

ServiceリソースのTYPEがNodePortに変更されたことを確認します。

```
Serviceの確認
$ kubectl get service tidb-sample-tidb
NAME                TYPE        CLUSTER-IP    EXTERNAL-IP  PORT(S)          ...
tidb-sample-tidb    NodePort    10.96.171.212 <none>       4000:30540/TCP, ...
```

Serviceリソースの4000番ポートに対応するNodePortとして30540番ポートが割り当たりました。続いてKubernetesクラスターのノードのIPアドレスを取得します。KubernetesのノードのIPアドレスはkubectl get nodeコマンド実行時に-o wideオプションを使用し、INTERNAL-IPの列を表示させることで確認できます。

```
ノードのIPアドレスの確認
$ kubectl get node -o wide
NAME                STATUS    ROLES    AGE   VERSION   INTERNAL-IP ...
k8s-tidb-control-plane Ready    control-plane  40h   v1.31.4   172.18.0.2   ...
k8s-tidb-worker      Ready    <none>       40h   v1.31.4   172.18.0.3   ...
k8s-tidb-worker2     Ready    <none>       40h   v1.31.4   172.18.0.4   ...
k8s-tidb-worker3     Ready    <none>       40h   v1.31.4   172.18.0.5   ...
```

Kubernetes クラスターのノードのIPアドレスのどれか1つと、割り当てられたNodePortを使用してTiDBに接続できます。

Kubernetesクラスター外からの接続

```
$ mysql --user root --host 172.18.0.3 --port 30540
```

なお、NodePortはデフォルトではランダムにポート番号が割り当てられますが、`.spec.tidb.service.mysqlNodePort`を指定することでNodePortを固定できます(リスト4.4の❶)。Kubernetesで利用できるNodePortの範囲内(30000から32767まで)で指定してください。

リスト4.4 NodePortを固定して使用するマニフェストの例

tidb-sample-cluster.yaml

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: tidb-sample
spec:
  ...
  (省略)
  ...
  tidb:
    baseImage: pingcap/tidb
    replicas: 1
    config: {}
    service:
      type: NodePort
      mysqlNodePort: 30306 —❶
```

▶ ポートフォワーディングを使用した接続

kubectl コマンドにはポートフォワーディング機能があります。この機能を使用するとKubernetes クラスター内のService リソースが使用しているポートをkubectl コマンド実行サーバーにフォワーディングできます。kubectl port-forward コマンドを使用してTiDBのService リソースのポートをフォワーディングする場合は次のように実行します。

ポートフォワーディングを実行

```
$ kubectl port-forward svc/tidb-sample-tidb 4000:4000
```

kubectl port-forward コマンドはフォアグラウンドで実行されたままになります。kubectl port-forward コマンドを実行した端末とは別の端末を使用して接続してください。手元の環境でkubectl port-forward コマンドを実行した場合、127.0.0.1の4000番ポートにアクセスするとTiDBに接続できます。

ポートフォワーディングでTiDBに接続

```
$ mysql --user root --host 127.0.0.1 --port 4000
```

各種コンポーネントをスケールアウト、スケールインする

本項ではTiDB Operatorにおける各コンポーネントのスケールアウトとスケールインの方法について説明します。

先ほど作成したTidbClusterリソースは各コンポーネントが1つずつ起動する定義になっています。スケールアウトの方法は非常に簡単で、TidbClusterのマニフェストファイル内の`.spec.*.replicas`を変更して`kubectl apply`コマンドで更新するだけです。本項では`kubectl apply`の手順のみ説明しますが、`kubectl edit`コマンドや`kubectl patch`コマンドを使用して直接編集して反映させることもできます。

TiUPを使用して各コンポーネントをスケールアウトする場合は、スケールアウト対象のみを記述したトポロジーファイルを使用しました。TiDB Operatorの場合はTidbClusterリソースのマニフェストファイル内の一部分を変更したとしても、マニフェストファイル全体を使用して`kubectl apply`コマンドで更新する必要がありますので注意してください。

まず始めにPDクラスターを3台にスケールアウトします。リスト4.5の❶の箇所を編集して3に変更します。PDクラスターの台数はスプリットブレインを防ぐために奇数台で構成してください。

リスト4.5 PDクラスターのスケールアウト

```
tidb-sample-cluster.yaml
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: tidb-sample
spec:
  ...
  (省略)
  ...
  pd:
    baseImage: pingcap/pd
    replicas: 3 —❶
    requests:
      storage: "1Gi"
    config: {}
  ...
  (省略)
  ...
```

変更後のマニフェストファイルを使用して`kubectl apply`コマンドでTidbClusterリソースを更新します。

```
PDクラスターのスケールアウト適用
$ kubectl apply -f tidb-sample-cluster.yaml
tidbcluster.pingcap.com/tidb-sample configured
```

PDクラスターのPodが3つに増えていることを確認します。

PD クラスターの Pod の確認

```
$ kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
tidb-sample-discovery-768fcb86f5-grs9t	1/1	Running	0	3m34s
tidb-sample-pd-0	1/1	Running	0	3m34s
tidb-sample-pd-1	1/1	Running	0	73s
tidb-sample-pd-2	1/1	Running	0	91s
tidb-sample-tidb-0	2/2	Running	0	2m34s
tidb-sample-tikv-0	1/1	Running	0	3m16s

続いて TiDB クラスターとストレージクラスター (TiKV) の両方を3台にスケールアウトします。同じようにマニフェストファイルを更新して `kubectl apply` コマンドで `TidbCluster` リソースを更新します (リスト 4.6 の①と②)。

リスト 4.6 TiDB クラスターと TiKV のスケールアウト

Tidb-sample-cluster.yaml

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: tidb-sample
spec:
  ...
  (省略)
  ...
  tikv:
    baseImage: pingcap/tikv
    replicas: 3 — ①
    config: {}
    requests:
      storage: "1Gi"
  tidb:
    baseImage: pingcap/tidb
    replicas: 3 — ②
    config: {}
```

TiDB クラスターと TiKV のスケールアウト確認

```
$ kubectl apply -f tidb-sample-cluster.yaml
tidbcluster.pingcap.com/tidb-sample configured
```

```
$ kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
tidb-sample-discovery-768fcb86f5-grs9t	1/1	Running	0	16m
tidb-sample-pd-0	1/1	Running	0	16m
tidb-sample-pd-1	1/1	Running	0	14m
tidb-sample-pd-2	1/1	Running	0	14m
tidb-sample-tidb-0	2/2	Running	0	15m

```

tidb-sample-tidb-1      2/2    Running    0          3m29s
tidb-sample-tidb-2      2/2    Running    0          3m29s
tidb-sample-tikv-0       1/1    Running    0          16m
tidb-sample-tikv-1       1/1    Running    0          3m29s
tidb-sample-tikv-2       1/1    Running    0          3m29s

```

これですべてのコンポーネントが3台にスケールアウトされました。

スケールインを行う時も同様に `.spec.*,replicas` を変更して `kubectl apply` コマンドで更新します。3台にスケールアウトさせた TiDB クラスターの Pod を1台に戻してみましょう(リスト 4.7の❶)。

リスト4.7 TiDBクラスターのスケールイン

```

tidb-sample-cluster.yaml
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: tidb-sample
spec:
  ...
  (省略)
  ...
  tidb:
    baseImage: pingcap/tidb
    replicas: 1  —❶
    config: {}

```

TiDBクラスターのスケールイン確認

```

$ kubectl apply -f tidb-sample-cluster.yaml
tidbcluster.pingcap.com/tidb-sample configured

$ kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE
tidb-sample-discovery-768fcb86f5-grs9t  1/1     Running   0          19m
tidb-sample-pd-0                       1/1     Running   0          19m
tidb-sample-pd-1                       1/1     Running   0          17m
tidb-sample-pd-2                       1/1     Running   0          17m
tidb-sample-tidb-0                     2/2     Running   0          18m
tidb-sample-tikv-0                     1/1     Running   0          19m
tidb-sample-tikv-1                     1/1     Running   0          6m39s
tidb-sample-tikv-2                     1/1     Running   0          6m39s

```

起動していた Pod の `tidb-sample-tidb-1` と `tidb-sample-tidb-2` が削除され、`tidb-sample-tidb-0` のみが残っていることが確認できます。Pod の台数が減る場合は Pod の末尾のインデックスが大きいものから削除されます。これは StatefulSet リソースによる管理のためです。そのため、通常はインデックスの数字は欠番が存在しないようになっていますが、運用上の理由などで欠番が発生する状況にしなければならない場合は Advanced StatefulSet の使用を検討してください。Advanced StatefulSet について

は第7章で説明します。

各種コンポーネントをスケールアップ、スケールダウンする

KubernetesではPodに対してCPUやメモリのリソースの割当量を指定できます。TiDBの各コンポーネントのリソースの割り当ても同様にTidbClusterリソースのマニフェストファイルで行います。

TiDB クラスターのPodに対してリソースの割当量を変更してみましょう。リソースの割当量はrequestsとlimitsの2つを指定できます。Kubernetesのリソースの割当量についてはKubernetesのドキュメントのコンテナのリソース管理^{注23}を参照してください。

まずTiDBクラスターのPodに対してCPUを1コア、メモリを256MiB割り当てます。.spec.tidb.requests.cpuおよび.spec.tidb.requests.memoryを指定します(リスト4.8の❶と❷)。requestsと同様にlimitsも指定できますが、ここではrequestsのみ指定しています。

リスト4.8 リソースの割当量の変更

```
tidb-sample-cluster.yaml
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: tidb-sample
spec:
  ...
  (省略)
  ...
  tidb:
    baseImage: pingcap/tidb
    replicas: 3
    config: {}
    requests:
      cpu: 1000m —❶
      memory: 256Mi —❷
```

kubectl applyコマンドを使用して設定を反映させます。TiDBクラスターのPodの状態をkubectl describe podコマンドで確認します。

```
describeによる確認
$ kubectl describe pod tidb-sample-tidb-0 | grep -A 2 Requests
Requests:
  cpu:      1
  memory:   256Mi
```

注23 <https://kubernetes.io/ja/docs/concepts/configuration/manage-resources-containers/>

あらかじめ定義した値を変更して、スケールアップおよびスケールダウンを行う場合もマニフェストファイルのリソース量を変更して `kubectl apply` で反映させます。たとえばTiDB クラスターのPodが使用できるメモリ量を倍の512MiBにしたい場合は次のように変更し、反映させます(リスト4.9の❶)。

リスト4.9 メモリの割当量を変更

```
tidb-sample-cluster.yaml
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: tidb-sample
spec:
  ...
  (省略)
  ...
  tidb:
    baseImage: pingcap/tidb
    replicas: 3
    config: {}
    requests:
      cpu: 1000m
      memory: 512Mi —❶
```

変更したマニフェストファイルを反映させると、Pod名のインデックスが大きいTiDBのPodから順次再作成されます。しばらくすると3台とも再作成されますので、再作成が終わったら再びPodのリソース割り当てを確認しましょう。

再度describeによる確認

```
$ kubectl describe pod tidb-sample-tidb-0 | grep -A 2 Requests
Requests:
  cpu:      1
  memory:   512Mi
```

メモリのリソース割り当てが512MiBに変更されていることが確認できます。

なお、少々長いコマンドになりますが次のように全台まとめて表示させることも可能です。

全台のリソース割り当ての確認

```
$ kubectl get pod --selector app.kubernetes.io/component=tidb \
  -o custom-columns='NAME:.metadata.name,CPU_REQ:spec.containers[1].resources.requests.cpu, MEM_REQ:spec.containers[1].resources.requests.memory'
NAME                CPU_REQ  MEM_REQ
tidb-sample-tidb-0   1        512Mi
tidb-sample-tidb-1   1        512Mi
tidb-sample-tidb-2   1        512Mi
```

このようにKubernetesの仕組みを使用してスケールアップやスケールダウンも簡単に行うことがで

きます。執筆時点では、KubernetesはPodのリソース量の変更を無停止で行うことはできません。Podの再作成がどうしても必要になりますので、スケールアップやスケールダウンを行う場合は注意してください。

TiProxyを構築する

TiProxyはTiDBクラスターの負荷分散、コネクションハンドリングを担当するコンポーネントです。TiDB OperatorでTiProxyを構築する場合は、次のようにTiDBClusterリソースのマニフェストファイルに`.spec.tiproxy`を追加します(リスト4.10の❶)。注意すべき点として、TiProxyはTiDBとバージョン体系が異なるため、`.spec.tiproxy.version`を指定する必要があります(リスト4.10の❷)。提供されているTiProxyのバージョンはTiProxyのGitHubリポジトリ^{注24}から確認できます。

リスト4.10 TiProxy用のマニフェストファイルの例

```
tidb-sample-cluster.yaml
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: tidb-sample
spec:
  ...
  (省略)
  ...
  tiproxy: —❶
    version: v1.3.0 —❷
    replicas: 2
    config: {}
```

また、TiProxyを利用する場合はTiDBクラスターの設定で`graceful-wait-before-shutdown`を入れておくことを推奨します。設定を入れる場合は`.spec.tidb.config`に設定を追加します。

```
graceful-wait-before-shutdownの設定
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: tidb-sample
spec:
  ...
  (省略)
  ...
  tidb:
```

注24 <https://github.com/pingcap/tiproxy/releases>

```
config: |
  graceful-wait-before-shutdown = 30
```

変更後のマニフェストファイルを使用して `kubectl apply` コマンドで `TidbCluster` リソースを更新し、`TiProxy` の Pod が起動していることを確認します。`TiProxy` も `StatefulSet` リソースによって管理されます。

TiProxyのPodの確認

```
$ kubectl apply -f tidb-sample-cluster.yaml
tidbcluster.pingcap.com/tidb-sample configured

$ kubectl get pod --selector app.kubernetes.io/component=tiproxy
NAME                READY  STATUS   RESTARTS  AGE
tidb-sample-tiproxy-0  1/1    Running   0          68m
tidb-sample-tiproxy-1  1/1    Running   0          69m
```

TiDB OperatorはTiProxy構築時にNodePortのServiceリソースを自動で作成するので、NodePortのServiceリソース経由でKubernetesクラスター外からTiProxyに接続できます。接続時には6000番ポートに対応するNodePortを使用してください。なお、Kubernetes内部で使用する`-peer`のサフィックスが付いたServiceリソースも同時に作成されます。

TiProxyに接続

```
$ kubectl get service --selector app.kubernetes.io/component=tiproxy
NAME                TYPE        ...  PORT(S)                ...
tidb-sample-tiproxy  NodePort    ...  3080:32537/TCP,6000:32449/TCP ...
tidb-sample-tiproxy-peer  ClusterIP   ...  3080/TCP,3081/TCP      ...

$ mysql --user root --host 172.18.0.3 --port 32449
```

NodePortではなく、LoadBalancerでServiceリソースを作成したい場合は手動で作成する必要があります。次の例はTiProxy用のServiceリソースをLoadBalancerで作成するマニフェストファイル例です。`.spec.selector`の`app.kubernetes.io/instance`は`TidbCluster`の`.metadata.name`と同じにしてください(リスト4.11の❶)。

リスト4.11 TiProxy用のServiceリソースの例

tiproxy-loadbalancer.yaml

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    name: tidb-sample-tiproxy-lb
spec:
  type: LoadBalancer
  selector:
    app.kubernetes.io/component: tiproxy
```

```
app.kubernetes.io/instance: tidb-sample — ❶
ports:
- name: tiproxy-api
  port: 3080
  protocol: TCP
  targetPort: 3080
- name: tiproxy-sql
  port: 6000
  protocol: TCP
  targetPort: 6000
```

マニフェストファイルを適用して、TYPEがLoadBalancerであるServiceリソースが作成されていることを確認します。

TiProxyのServiceリソースの作成

```
$ kubectl apply -f tiproxy-loadbalancer.yaml
service/tidb-sample-tiproxy-lb created

$ kubectl get service tidb-sample-tiproxy-lb
NAME                                TYPE             CLUSTER-IP      EXTERNAL-IP ...
tidb-sample-tiproxy-lb             LoadBalancer     10.96.43.209    172.18.0.3 ...
```

EXTERNAL-IPに割り当てられたIPアドレスを使用してTiProxyに接続できます。

TiFlashを構築する

TiFlashはOLAP用の列指向ストレージデータを保存するコンポーネントです。TiDB OperatorでTiFlashを構築する場合は、次のようにTidbClusterリソースのマニフェストファイルに`.spec.tiflash`を追加します(リスト4.12の❶)。`.spec.tiflash.storageClaims`は必須項目です(リスト4.12の❷)。`.spec.tiflash.storageClaims.resources.requests.storage`にTiFlashが使用するストレージ容量を指定します(リスト4.12の❸)。

リスト4.12 TiFlashのマニフェストファイルの例

```
tidb-sample-cluster.yaml
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: tidb-sample
spec:
  ...
  (省略)
  ...
  tiflash: — ❶
    replicas: 1
```

```

config: {}
storageClaims: — ❷
  - resources:
    requests:
      storage: 10Gi — ❸
    storageClassName: standard — ❹

```

.spec.tiflash.storageClaims.storageClassNameは省略できます(リスト4.12の❹)。その場合はKubernetesクラスター内のデフォルトのStorageClassが使用されます。デフォルトのStorageClassを確認する場合は、`kubectl get storageclass` コマンドを使用して (default) と表示されるStorageClassを確認してください。

デフォルトのStorageClassの確認

```

$ kubectl get storageclass
NAME                PROVISIONER             RECLAIMPOLICY   ...
local-storage       kubernetes.io/no-provisioner   Delete         ...
standard (default)  rancher.io/local-path      Delete         ...

```

TiFlashのPodが起動していることを確認します。

TiFlashのPodの確認

```

$ kubectl get pod --selector app.kubernetes.io/component=tiflash
NAME                READY   STATUS    RESTARTS   AGE
tidb-sample-tiflash-0  4/4     Running   0           9m8s

```

TiDB Dashboardを構築する

TiDB Dashboardを使用する場合はTidbDashboardリソースのマニフェストファイルを作成して`kubectl apply`コマンドで適用します。TidbDashboardリソースのマニフェストファイルは次のようになります。

リスト4.13 TiDB Dashboardのマニフェストファイルの例

tidb-sample-dashboard.yaml

```

apiVersion: pingcap.com/v1alpha1
kind: TidbDashboard
metadata:
  name: tidb-sample-dashboard
spec:
  baseImage: pingcap/tidb-dashboard
  version: v8.5.0 — ❶
  clusters: — ❷
    - name: tidb-sample
  requests:
    storage: 1Gi — ❸

```

.spec.versionでTiDB Dashboardのバージョンを指定します(リスト4.13の❶)。構築したTiDBと同じバージョンを指定してください。TiDBClusterリソースの.spec.versionと同じバージョンを指定しておくといでしょう。spec.clustersで対象のTiDB(TidbClusterリソースの.metadata.name)を指定します(リスト4.13の❷)。配列で定義できますが、執筆時点では1つの対象のみ対応しています。spec.requestsではTiDB DashboardのPodのリソース量を設定します。spec.requests.storageでストレージの割当量を指定します(リスト4.13の❸)。ストレージは必須項目となります。その他はCPUやメモリのリソース量も指定できます。

作成したマニフェストファイルを使用してkubectl applyコマンドでTidbDashboardリソースを作成します。

TidbDashboardの作成

```
$ kubectl apply -f tidb-sample-dashboard.yaml
tidbdashboard.pingcap.com/tidb-sample-dashboard configured
```

TiDB OperatorによりStatefulSetリソースの<metadata.name>-tidb-dashboardおよびServiceリソースの<metadata.name>-tidb-dashboard-exposedが作成されます。

StatefulSetとServiceの確認

```
$ kubectl get statefulset --selector app.kubernetes.io/component=tidb-dashboard
NAME                                READY  AGE
tidb-sample-dashboard-tidb-dashboard 1/1    22m

$ kubectl get service --selector app.kubernetes.io/component=tidb-dashboard
NAME                                TYPE        CLUSTER-IP  ...
tidb-sample-dashboard-tidb-dashboard-exposed  ClusterIP   10.96.32.20  ...
```

作成されたServiceリソースの12333番ポートに接続することでTiDB Dashboardにアクセスできます。アクセスの仕方は本節で説明した「構築したTiDBへ接続する」を参照してください。デフォルトで作成されるServiceリソースはTYPEがClusterIPとなり、Kubernetesクラスター外部からアクセスできません。Kubernetesクラスター外部からアクセスする場合は、TYPEをLoadBalancerやNodePortにしてServiceリソースを定義する必要があります。

TYPEの指定はTidbDashboardリソースのマニフェスト内の.spec.service.typeで指定できます。たとえばNodePortでServiceリソースを作成する場合は次のようにマニフェストファイルを変更し、適用してください(リスト4.14の❶)。

リスト4.14 NodePortを指定したTiDB Dashboardのマニフェストファイルの例

tidb-sample-dashboard.yaml

```
apiVersion: pingcap.com/v1alpha1
kind: TidbDashboard
metadata:
  name: tidb-sample-dashboard
```

```
spec:
  baseImage: pingcap/tidb-dashboard
  version: v8.5.0
  clusters:
    - name: tidb-sample
  requests:
    storage: 1Gi
  service:
    type: NodePort ①
```

■ モニタリング環境を構築する

TiDB Operator は Prometheus と Grafana を使用したモニタリング環境の構築に対応しています。Prometheus と Grafana を構築する場合は TidbMonitor リソースのマニフェストファイルを作成して `kubectl apply` コマンドで適用します。最低限必要な設定項目を記述した TidbMonitor リソースのマニフェストファイルは次のようになります。

リスト4.15 TidbMonitorのマニフェストファイルの例

```
tidb-sample-monitor.yaml
apiVersion: pingcap.com/v1alpha1
kind: TidbMonitor
metadata:
  name: tidb-sample-monitor
spec:
  clusters:
    - name: tidb-sample ①
  prometheus:
    baseImage: prom/prometheus
    version: v2.27.1 ②
  grafana:
    baseImage: grafana/grafana
    version: 7.5.11
  initializer:
    baseImage: pingcap/tidb-monitor-initializer
    version: v8.5.0
  reloader:
    baseImage: pingcap/tidb-monitor-reloader
    version: v1.0.1
```

`.spec.clusters` でモニタリング対象の TiDB (TidbCluster リソースの `.metadata.name`) を指定します (リスト 4.15 の ①)。複数の TiDB を対象にできます。たとえば違う Namespace 上にある TiDB を同時に監視する場合は、次のように Namespace と共に記述します。

複数のTiDBを監視する例

```
apiVersion: pingcap.com/v1alpha1
kind: TidbMonitor
metadata:
  name: tidb-sample-monitor
spec:
  clusters:
    - name: tidb-sample
      namespace: default
    - name: tidb-sample2
      namespace: other
...
(省略)
...
```

.spec.*.versionでそれぞれのコンポーネントのバージョンを指定します(リスト4.15の②)。各コンポーネントのバージョンはGitHubに公開されているexamplesのマニフェスト^{注25}を参考にするとよいでしょう。必須ではありませんが、spec.prometheusReloaderを指定することでPrometheusの設定ファイルをリロードするためのコンポーネントを使用できますので、必要に応じて設定してください。

prometheusReloaderの設定

```
apiVersion: pingcap.com/v1alpha1
kind: TidbMonitor
metadata:
  name: tidb-sample-monitor
spec:
...
(省略)
...
prometheusReloader:
  baseImage: quay.io/prometheus-operator/prometheus-config-reloader
  version: v0.49.0
```

TidbMonitorリソースのマニフェストファイルを適用すると、StatefulSetリソースの<metadata.name>-monitorとそれに紐づくPrometheusやGrafanaのServiceリソースが作成されます。

StatefulSetとServiceの確認

```
$ kubectl get statefulset --selector app.kubernetes.io/component=monitor
NAME                                READY   STATUS    RESTARTS   AGE
tidb-sample-monitor-monitor-0      3/3     Running   0           35m

$ kubectl get service --selector app.kubernetes.io/component=monitor
NAME                                TYPE        CLUSTER-IP    EXTERNAL-IP  ...
tidb-sample-monitor-grafana        ClusterIP   10.96.239.147 <none>       ...
```

注25 <https://github.com/pingcap/tidb-operator/blob/v1.6.1/examples/basic/tidb-monitor.yaml>


```
tidb-sample-monitor-monitor-reloader ClusterIP 10.96.50.137 <none> ...
tidb-sample-monitor-prometheus ClusterIP 10.96.139.185 <none> ...
```

デフォルトでは作成される各種ServiceリソースのTYPEがClusterIPとなっているため、外部からアクセスする場合はLoadBalancerやNodePortでServiceリソースを定義する必要があります。TidbMonitorリソースのマニフェスト内のspec.service.typeで指定できます。たとえばGrafanaにアクセスするためにロードバランサーを使用したい場合は次のように指定します(リスト4.16の❶)。

リスト4.16 GrafanaのServiceにロードバランサーを使用する例

```
tidb-sample-monitor.yaml
apiVersion: pingcap.com/v1alpha1
kind: TidbMonitor
metadata:
  name: tidb-sample-monitor
spec:
  ...
  (省略)
  ...
  grafana:
    baseImage: grafana/grafana
    version: 7.5.11
    service:
      type: LoadBalancer —❶
```

Prometheusに蓄積されるメトリクスデータはデフォルトの設定では永続化されておらず、PrometheusのPodの再起動時にメトリクスデータが消失します。永続化するためにはPrometheusが使用するストレージを永続ボリュームにマウントする必要があります。その場合、TidbMonitorリソースのマニフェスト内で.spec.persistentをtrueにして(リスト4.17の❶)、.spec.storageClassNameと.spec.storageを指定してください(リスト4.17の❷と❸)。.spec.storageClassNameは省略可能で、省略した場合はデフォルトのStorageClassが使用されます。次の例はStorageClassをstandard、容量を10Giに指定して永続化する例です。

リスト4.17 メトリクスデータの永続化の例

```
tidb-sample-monitor.yaml
apiVersion: pingcap.com/v1alpha1
kind: TidbMonitor
metadata:
  name: tidb-sample-monitor
spec:
  persistent: true —❶
  storageClassName: standard —❷
  storage: 10Gi —❸
  prometheus:
```

```
baseImage: prom/prometheus
version: v2.27.1
```

```
...
(省略)
...
```

Grafanaの各種設定を変更したい場合は環境変数を使用して設定を変更できます。たとえばGrafanaのデフォルトのユーザー名とパスワードを変更する場合は次のように設定します(リスト4.18の❶と❷)。

リスト4.18 Grafanaのユーザー名とパスワードの変更

Grafanaのユーザー名とパスワードの変更

```
apiVersion: pingcap.com/v1alpha1
kind: TidbMonitor
metadata:
  name: tidb-sample-monitor
spec:
  ...
  (省略)
  ...
  grafana:
    baseImage: grafana/grafana
    version: 7.5.11
    envs:
      GF_SECURITY_ADMIN_USER: newadmin —❶
      GF_SECURITY_ADMIN_PASSWORD: newpassword —❷
  ...
  (省略)
  ...
```

環境変数の設定の仕方はGrafanaのドキュメントのConfigure Grafana^{注26}を参照してください。

NgMonitoringを有効化する

NgMonitoring(Next Generation Monitoring)はTiDBの継続的なプロファイリングやTop SQL機能を使用する場合に必要なコンポーネントです。TiUPで構築したTiDBやTiDB Playgroundではデフォルトで有効になっていますが、TiDB Operatorによる構築では手動で有効化する必要があります。

NgMonitoringを有効化するにはTidbNGMonitoringリソースを定義します。TidbNGMonitoringリソースのマニフェストファイルは次のようになります。

注26 <https://grafana.com/docs/grafana/latest/setup-grafana/configure-grafana/#override-configuration-with-environment-variables>

リスト4.19 TidbNGMonitoringの manifests ファイルの例

```

tidb-sample-ngmonitoring.yaml
apiVersion: pingcap.com/v1alpha1
kind: TidbNGMonitoring
metadata:
  name: tidb-sample-ngmonitoring
spec:
  clusters:
    - name: tidb-sample ①
  ngMonitoring:
    baseImage: pingcap/ng-monitoring
    version: v8.5.0 ②
    requests:
      storage: 10Gi ③

```

.spec.clusters で対象の TiDB (TidbCluster リソースの .metadata.name) を指定します (リスト 4.19 の ①)。.spec.ngMonitoring.version で NgMonitoring のバージョンを指定します (リスト 4.19 の ②)。.spec.ngMonitoring.requests.storage でストレージの割当量を指定します (リスト 4.19 の ③)。

作成した manifests ファイルを使用して `kubectl apply` コマンドで TidbNGMonitoring リソースを作成します。

TidbNGMonitoring の作成

```

$ kubectl apply -f tidb-sample-ngmonitoring.yaml
tidbngmonitoring.pingcap.com/tidb-sample-ngmonitoring configured

```

NgMonitoring の Pod が起動していることを確認します。

NgMonitoring の Pod の確認

```

$ kubectl get pod --selector app.kubernetes.io/component=ng-monitoring
NAME                                READY   STATUS    RESTARTS   AGE
tidb-sample-ngmonitoring-ng-monitoring-0  1/1     Running   0           72m

```

TiDB Dashboard にアクセスし、「Top SQL」や「Continuous Profiling」を選択します。その後機能を有効化することで TiDB Dashboard で各種情報が表示されるようになります。TiDB Dashboard の詳細は第 7 章を参照してください。

バージョンアップする

本章の冒頭で Kubernetes は宣言型の管理方式であり、あるべき状態を定義することで Kubernetes がその状態に遷移するように制御するという説明をしました。あるべき状態は各種カスタムリソースで定義されています。つまり、バージョンを変更する場合はカスタムリソース内で定義されているバージョンを新しく更新し、Kubernetes へ適用すれば良いということになります。カスタムリソースを更新する方法はさまざまありますが、本項では manifests ファイルを変更して `kubectl apply` コマンド

を使用して適用する方式で紹介します。

▶ TiDBのバージョンアップ

tidb-sampleを作成した際に使用したマニフェストファイル(tidb-sample-cluster.yaml)では次のようにバージョンが定義されていました(リスト4.20の❶)。

リスト4.20 tidb-sampleのマニフェストファイル

```
tidb-sample-cluster.yaml
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: tidb-sample
spec:
  version: v8.5.0 —❶
...
(省略)
...
```

実際に構築されているTiDBからも確認します。

```
実際に稼働しているバージョンの確認
$ kubectl get tidbcluster
NAME          READY  PD          ...  TIKV          ...
tidb-sample   True   pingcap/pd:v8.5.0 ... pingcap/tikv:v8.5.0 ...
```

使用されている各種コンテナイメージのタグがv8.5.0であることが確認できます。

マニフェストファイル(tidb-sample-cluster.yaml)を変更して新しいバージョンを指定します。たとえばバージョンを8.5.1に変更する場合は次のように変更します(リスト4.21の❶)。

リスト4.21 tidb-sampleのバージョンアップ

```
tidb-sample-cluster.yaml
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: tidb-sample
spec:
  version: v8.5.1 —❶
...
(省略)
...
```

変更したマニフェストファイルを使用して kubectl apply コマンドで適用します。

バージョンの変更を適用

```
$ kubectl apply -f tidb-sample-cluster.yaml
tidbcluster.pingcap.com/tidb-sample configured
```

適用するとTiDB Operatorが変更を検知し、各種コンポーネントをバージョンアップしていきます。バージョンアップはPDクラスターのPodから開始され、ストレージクラスター、TiDBクラスターへと続いていきます。この順番はTiDB Operatorによって制御されています。

無事にバージョンアップが終了すると、各コンポーネントのバージョンが変更されていることが確認できます。バージョンアップ中はTiDBClusterリソースのREADYがFalseで表示され、バージョンアップが問題無く終わるとTrueになります。

更新後のバージョンの確認

```
$ kubectl get tidbcluster
NAME      READY  PD      ...  TIKV      ...
tidb-sample  True   pingcap/pd:v8.5.1 ... pingcap/tikv:v8.5.1 ...
```

ダウングレードは執筆時点ではサポートされていないので注意してください。

 TiDB Operatorのバージョンアップ

Helmチャートを使用してTiDB Operatorをインストールする手順を紹介しましたが、TiDB Operatorのバージョンアップもインストールと同様にHelmチャートを使用して行います。

まず初めに現在インストールされているバージョンを確認します。確認はhelm listコマンドで行います。Namespaceを変更してインストールした場合は--namespaceオプションを使用してください。なお、バージョンアップを確認するためにあらかじめHelmチャートのバージョンv1.6.0をインストールしてあります。

現在のバージョンの確認

```
$ helm list
NAME      NAMESPACE  ...  CHART      APP VERSION
tidb-operator  default    ...  tidb-operator-v1.6.0  v1.6.0
```

バージョンアップする場合はhelm upgradeコマンドを使用します。--versionオプションでバージョンを指定します。

バージョンアップ

```
$ helm upgrade tidb-operator pingcap/tidb-operator --version v1.6.1
Release "tidb-operator" has been upgraded. Happy Helming!
NAME: tidb-operator
LAST DEPLOYED: Mon Dec 2 17:58:13 2024
NAMESPACE: default
STATUS: deployed
REVISION: 2
TEST SUITE: None
```

NOTES:

Make sure tidb-operator components are running:

```
kubectll get pods --namespace default -l app.kubernetes.io/instance=tidb-operator
```

先ほどバージョンを確認した `helm list` コマンドでバージョンが変更されていることを確認します。

バージョンアップ後の確認

```
$ helm list
```

NAME	NAMESPACE	...	CHART	APP VERSION
tidb-operator	default	...	tidb-operator-v1.6.1	v1.6.1

`helm upgrade` コマンドに関する詳細は Helm のドキュメントの Helm Upgrade^{注27} を参照してください。

4-4

ログの確認

本節では Kubernetes 上で稼働している各種コンポーネントのログの確認方法を説明します。ログの確認はデバッグやトラブルシューティングの際に役立ちます。また、TiDB のスローログも同様の手順で確認できますので、運用の際にぜひ活用してください。

コンポーネントのログを確認する

Kubernetes 上で稼働している各種コンポーネントのログは `kubectll logs` コマンドで確認できます。`kubectll logs` コマンドは Pod 名や Pod に付与されているラベルを指定してログを表示します。Pod 名の確認は `kubectll get pod` コマンドで行います。

Pod 名の確認

```
$ kubectll get pod
```

NAME	READY	STATUS	RESTARTS	AGE
tidb-sample-discovery-768fcb86f5-vwkz6	1/1	Running	0	10m
tidb-sample-pd-0	1/1	Running	0	10m
tidb-sample-pd-1	1/1	Running	0	10m
tidb-sample-pd-2	1/1	Running	0	10m
tidb-sample-tidb-0	2/2	Running	0	9m17s
tidb-sample-tidb-1	2/2	Running	0	9m17s
tidb-sample-tidb-2	2/2	Running	0	9m17s
tidb-sample-tikv-0	1/1	Running	0	9m47s
tidb-sample-tikv-1	1/1	Running	0	9m47s
tidb-sample-tikv-2	1/1	Running	0	9m47s

注27 https://helm.sh/docs/helm/helm_upgrade/

tidb-sample-pd-0のログを確認する場合は次のようにします。

ログの確認

```
$ kubectl logs tidb-sample-pd-0
Server: 10.96.0.10
Address: 10.96.0.10#53

Name:      tidb-sample-pd-0.tidb-sample-pd-peer.default.svc.cluster.local
Address: 10.244.3.23

nslookup domain tidb-sample-pd-0.tidb-sample-pd-peer.default.svc.svc success
starting pd-server ...
/pd-server --data-dir=/var/lib/pd --name=tidb-sample-pd-0 ...
[2024/12/03 00:47:28.690 +00:00] [INFO] ... ["use physical memory hook"] ...
[2024/12/03 00:47:28.690 +00:00] [INFO] ... ["Welcome to Placement Driver (PD)"]
...
(省略)
...
```

ログをリアルタイムで確認したい場合は--followオプションを使用します。また、直近のログを確認したい場合は--tailオプションを使用します。これらのオプションは組み合わせて使用できます。たとえばtidb-sample-tikv-0の直近の5件のログを表示し、その後のログをリアルタイムで表示する場合は次のように実行します。

リアルタイムでのログの確認

```
$ kubectl logs --follow --tail 5 tidb-sample-tikv-0
[2024/12/03 01:04:14.242 +00:00] [WARN] ... ["[region 1256] 1259 handle ...
[2024/12/03 01:04:17.363 +00:00] [WARN] ... ["[store 1005] handle 1 store ...
[2024/12/03 01:06:13.295 +00:00] [WARN] ... ["[region 1044] 1047 handle 1 ...
[2024/12/03 01:07:51.566 +00:00] [INFO] ... ["connecting to PD endpoint"] ...
[2024/12/03 01:08:24.565 +00:00] [WARN] ... ["[region 1010] 1023 handle 1 ...
```

複数のPodのログを一括で表示したい場合はPodに付与されているラベルを指定します。ラベルの確認はkubectl get pod --show-labels コマンドで行います。

ラベルを指定してのログの確認

```
$ kubectl get pod --show-labels
NAME                                LABELS
tidb-sample-discovery-768fcb86f5-vwkz6 ... app.kubernetes.io/component=discovery, ...
tidb-sample-pd-0                    ... app.kubernetes.io/component=pd, ...
tidb-sample-pd-1                    ... app.kubernetes.io/component=pd, ...
tidb-sample-pd-2                    ... app.kubernetes.io/component=pd, ...
tidb-sample-tidb-0                  ... app.kubernetes.io/component=tidb, ...
tidb-sample-tidb-1                  ... app.kubernetes.io/component=tidb, ...
tidb-sample-tidb-2                  ... app.kubernetes.io/component=tidb, ...
tidb-sample-tikv-0                  ... app.kubernetes.io/component=tikv, ...
```

```
tidb-sample-tikv-1          ... app.kubernetes.io/component=tikv, ...
tidb-sample-tikv-2          ... app.kubernetes.io/component=tikv, ...
```

さまざまなラベルが付与されていることが確認できます。たとえばコンポーネントごとに付与されている `app.kubernetes.io/component` のラベルを使用してログを表示してみましょう。--follow オプションを組み合わせて使用することで、コンポーネント全体でのリアルタイムログの表示も可能です。

PDコンポーネントごとのログの確認

```
$ kubectl logs --selector app.kubernetes.io/component=pd --follow
```

`app.kubernetes.io/component` のラベルがどのように付与されているのかを確認するには `kubectl get` コマンド時に `--label-columns` オプションを使用します。COMPONENT の列に `app.kubernetes.io/component` の値が表示されます。

コンポーネントラベルの確認

```
$ kubectl get pod --label-columns=app.kubernetes.io/component
```

NAME	READY	STATUS	RESTARTS	AGE	COMPONENT
tidb-sample-discovery-768fcb86f5-vwkz6	1/1	Running	0	38m	discovery
tidb-sample-pd-0	1/1	Running	0	38m	pd
tidb-sample-pd-1	1/1	Running	0	38m	pd
tidb-sample-pd-2	1/1	Running	0	38m	pd
tidb-sample-tidb-0	2/2	Running	0	37m	tidb
tidb-sample-tidb-1	2/2	Running	0	37m	tidb
tidb-sample-tidb-2	2/2	Running	0	37m	tidb
tidb-sample-tikv-0	1/1	Running	0	37m	tikv
tidb-sample-tikv-1	1/1	Running	0	37m	tikv
tidb-sample-tikv-2	1/1	Running	0	37m	tikv

スローログを確認する

TiDB のスローログの確認も `kubectl logs` コマンドで取得できます。スローログは TiDB の Pod 内で `slowlog` というコンテナのログとして出力されています。kubectl logs コマンドで `--container` オプションを使用してコンテナ名を指定することでスローログを取得できます。

スローログの確認

```
$ kubectl logs tidb-sample-tidb-0 --container slowlog
# Time: 2024-12-03T01:32:39.135599916Z
# Txn_start_ts: 454345243707572225
# Query_time: 0.488154084
# Parse_time: 0
# Compile_time: 0.460240209
# Rewrite_time: 0.064539417
# Optimize_time: 0.293699208
# Wait_TS: 0.000251458
```



```
# Cop_time: 0.038947043 Process_time: 0.000017209 Wait_time: 0.004187583 ...
# Index_names: [tidb_background_subtask:idx_exec_id,tidb_global_task:state]
# Is_internal: true
# Digest: 8e00fe5c3677d581b104159ff91d824e3be8160c4e4be058288db681abd1a134
# Stats: tidb_global_task:pseudo[10000;0],tidb_background_subtask:pseudo[10000;0]
# Num_cop_tasks: 2
...
(省略)
...
```

4-5 初期パスワードとイニシャライズ

TidbInitializer リソースを使用することで、TiDB の初期パスワードや初期化処理を設定できます。

リスト4.22 TidbInitializer のマニフェストファイルの例

```
tidb-sample-initializer.yaml
apiVersion: pingcap.com/v1alpha1
kind: TidbInitializer
metadata:
  name: tidb-sample-initializer
spec:
  image: tnir/mysqlclient
  cluster:
    name: tidb-sample ①
  initSql: |- ②
    CREATE DATABASE app;
    GRANT ALL PRIVILEGES ON app.* TO 'developer'@'%';
  passwordSecret: tidb-sample-secret ③
```

.spec.cluster で対象の TiDB (TidbCluster リソースの .metadata.name) を指定します (リスト 4.22 の ①)。.spec.initSql には初期化処理をするための SQL を記述します (リスト 4.22 の ②)。.spec.passwordSecret に初期パスワードを記述した Secret リソース名を指定します (リスト 4.22 の ③)。Secret リソースは別途作成しておく必要があります。たとえば次のように Secret リソースを作成します。

Secret の作成

```
$ kubectl create secret generic tidb-sample-secret --from-literal=root=newpassword
secret/tidb-sample-secret created
```

マニフェストファイルを適用すると TidbInitializer リソースが作成されます。TidbInitializer リソースで定義した処理は Kubernetes の Job リソースとして実行されます。

マニフェストファイルの適用

```
$ kubectl apply -f tidb-sample-initializer.yaml
tidbinitializer.pingcap.com/tidb-sample-initializer created
```

```
$ kubectl get job
```

NAME	STATUS	COMPLETIONS	DURATION	AGE
tidb-sample-tidb-initializer	Running	0/1	3s	3s

実行が終了すると初期化処理が完了します。Job リソースのSTATUSがCompleteになっていることを確認してください。

```
$ kubectl get job
```

NAME	STATUS	COMPLETIONS	DURATION	AGE
tidb-sample-tidb-initializer	Complete	1/1	7s	8s

4-6

TiDBの削除

構築したTiDBをKubernetes上から削除するにはTidbClusterリソースを削除します。TiDB Operatorによって作成されたStatefulSetリソースやServiceリソースを直接削除しても、TiDB Operatorはあるべき状態へ戻そうとするために各種リソースを再作成します。そのため、各種リソースを手動で削除せずに、TidbClusterリソースを削除することでTiDBを削除してください。

TidbClusterの削除

```
$ kubectl delete tidbcluster tidb-sample
tidbcluster.pingcap.com "tidb-sample" deleted
```

もしくはkubectl applyコマンド実行時に使用したマニフェストファイルを使用して削除することもできます。

マニフェストファイルを使用した削除

```
$ kubectl delete -f tidb-sample-cluster.yaml
tidbcluster.pingcap.com "tidb-sample" deleted
```

TidbClusterリソース以外にTidbMonitorリソースやTidbDashboardリソースなどを作成している場合は、同様の手順で削除してください。

StatefulSetリソースによって作成されたPersistentVolumeClaimリソースは削除されませんので、必要に応じて手動で削除してください。デフォルトではTiDB OperatorはPersistentVolumeリソースのPersistentVolumeReclaimPolicyをRetainに設定します。そのため、PersistentVolumeClaimリソースを削除してもPersistentVolumeリソースは削除されません。PersistentVolumeReclaimPolicyを変更する場合はTidbClusterリソースの定義で.spec.pvReclaimPolicyを指定してください(リスト4.23の①)。

リスト4.23 PersistentVolumeReclaimPolicyの変更

```

PersistentVolumeReclaimPolicyの変更
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: tidb-sample
spec:
  version: v8.5.0
  timezone: UTC
  pvReclaimPolicy: Delete  — ❶
...
(省略)
...

```

4

4-7**TiDB Operatorのアンインストール**

TiDB Operatorをアンインストールする場合はhelm uninstallコマンドを使用します。helm listコマンドでインストールされているチャートの名前を確認し、helm uninstallコマンドでアンインストールします。インストールした際に指定したNamespaceを指定してアンインストールしてください。

TiDB Operatorのアンインストール

```

$ helm list --namespace tidb-admin
NAME          NAMESPACE ... CHART              APP VERSION
tidb-operator tidb-admin ... tidb-operator-v1.6.1 v1.6.1

$ helm uninstall tidb-operator --namespace tidb-admin
release "tidb-operator" uninstalled

```

kubectl get pod コマンドでTiDB OperatorのPodが削除されていることを確認してください。

Podの削除確認

```

$ kubectl get pod --namespace tidb-admin --selector app.kubernetes.io/instance=tidb-operator
No resources found in tidb-admin namespace.

```

4-8

まとめ

本章ではKubernetes上にTiDBを構築する方法を紹介しました。TiDBは分散システムであるため、Kubernetes上での構築は複雑ですが、TiDB Operatorを使用することで簡単に構築できます。また、TiDB Operatorを使用することでTiDBのバージョンアップやスケールアウトなども容易に行うことができます。しかしながら、Kubernetes自体の構築や運用はそれなりの知識が必要になります。Kubernetesを自身で運用、管理できる体制が整っているかどうかを確認してからTiDB Operatorを使用することをおすすめします。次章ではTiDB Cloudの使い方を紹介します。TiDBの運用コストを抑えたい場合や、運用管理を簡略化したい場合はTiDB Cloudを検討してください。

第 5 章

TiDB Cloudの利用

TiDBのマネージドサービスを利用して
運用負荷を軽減

TiDB CloudはTiDBをフルマネージドの環境で利用できるサービスです。TiDB Cloudを利用することでTiDBに関わる運用コストを軽減し、アプリケーションの開発に集中できます。

TiDB Cloudを使い始める前に、TiDB Cloud ServerlessおよびTiDB Cloud Dedicatedについて理解しておくことが重要です。どちらのサービスがアプリケーションに適しているかを理解し、適切なサービスを選択してください。

本章ではTiDB Cloudで提供されているTiDB Cloud ServerlessとTiDB Cloud Dedicatedのそれぞれの特徴について最初に説明し、続いてTiDB Cloudの始め方、接続方法、およびTiDB Cloud CLIの使い方について説明します。

本章は執筆時点でのTiDB CloudのWebコンソールの画面のスクリーンショットを使用しています。TiDB CloudのWebコンソールのアップデートにより、本書の内容と実際の画面に差異が生じる可能性があります。あらかじめご了承ください。

5-1 TiDB Cloud Serverless

TiDB Cloud Serverlessは、サーバーレスのアーキテクチャーを採用したTiDBのマネージドサービスです。料金体系は従量課金制で、リクエストユニット(RU)単位での利用量とストレージ利用量に応じて料金が発生します。リクエストユニットの詳細については7-3節で説明します。料金の詳細に関してはドキュメントのTiDB Cloud Serverless Pricing Details^{注1}を参照してください。

TiDB Cloud ServerlessはFree ClusterおよびScalable Clusterの2つの利用プランがあります。Free Clusterは支払いのためのクレジットカードの登録無しに利用できますが、次の制限があります。

- 作成できるクラスター数は最大5つまで
- 行ベースと列ベースのストレージ利用量はそれぞれ5GiBまで
- 毎月のリクエストユニット利用量は5000万まで
- クエリあたりのメモリ割り当ては256MiBに制限

リクエストユニットの利用量が5000万を超えた場合、新規接続は拒否されるようになります。既存の接続は維持されますが、さまざまな処理は再びリクエストユニットが使用できるまで制限されますので注意してください。

Scalable ClusterはFree Clusterにあるようなさまざまな制限はありませんが、支払いのためのクレジットカードの登録が必要になります。また、月額限度額(Monthly Spending Limit)の設定が必要になります。月額限度額はScalable Clusterを作成する際に設定でき、既存の月額限度額の設定は後から変更

注1 <https://www.pingcap.com/tidb-serverless-pricing-details/>

できます。なお、Free ClusterからScalable Clusterへの変更は可能ですが、Scalable ClusterからFree Clusterへの変更はできません。

TiDB Cloud Serverlessは利用状況に応じて自動的にスケールアウトおよびスケールインを行います。手動でスケールアウトおよびスケールインはできませんので注意してください。

TiDB Cloud Serverlessは執筆時点ではAWS(Amazon Web Services)環境のみで提供されています。プライベートエンドポイントを有効にして、自身のAWSの環境と接続するにはAWS PrivateLink^{注2}を使用してください。自身のAWSとの接続の手順の詳細はドキュメントのConnect to TiDB Cloud Serverless via Private Endpoint^{注3}を参照してください。TiDB Cloud Serverlessはマルチテナント環境になっています。そのため、払い出されるユーザー名の先頭にそれを区別するためのKeyspace名が付与されます(たとえばrootユーザーは<Keyspace名>.rootというユーザーで払い出されます)。

高可用性を選択する

TiDB Cloud Serverlessでは高可用性の選択肢として、Zonal high availabilityとRegional high availabilityがあります。

Zonal high availabilityは1つのアベイラビリティゾーン内での高可用性を提供します。PDコンポーネント以外のコンポーネントが1つのアベイラビリティゾーンに配置されるため、アベイラビリティゾーンの障害時には利用できない一方、レイテンシーが小さくなります。PDコンポーネントはZonal high availabilityを選択した場合でも、自動的に3つのアベイラビリティゾーンにまたがって構成されます。

Regional high availabilityはコンポーネントが複数のアベイラビリティゾーンにまたがって配置されます。そのため、アベイラビリティゾーンに障害があった場合でもサービスを継続できます。可用性はZonal high availabilityよりも高くなりますが、レイテンシーが大きくなる点とアベイラビリティゾーン間の転送料金が加算される点に注意が必要です。

執筆時点ではRegional high availabilityはベータ版の機能であり、AWS東京リージョン(ap-northeast-1)でのみ有効です。また、Scalable Clusterの作成時のみ選択できます。

注2 <https://aws.amazon.com/jp/privatelink/>

注3 <https://docs.pingcap.com/tidbcloud/set-up-private-endpoint-connections-serverless>

パブリッククラウドのリージョンとアベイラビリティゾーン

column

パブリッククラウドではリージョンとアベイラビリティゾーン(AZと略されることもあります)が連携して高い可用性を提供しています。リージョンは地理的なエリアを表しており、たとえば日本やアメリカなどの特定の地域を指します。アベイラビリティゾーンは1つのリージョン内に複数配置され、地理的に独立したデータセンターを表しており、それぞれが独立した電力設備やネットワークを持っています。リージョンは地理的規模の信頼性を提供し、アベイラビリティゾーンは其中でさらなる安定性を確保するための仕組みとなっています。

なお、TiDBが扱うデータの単位もリージョンという単語を使用しますが、パブリッククラウドにおけるリージョンとは別の単語になります。

TiDB Cloud Serverless Branchingを使用してデータやスキーマを別々に管理する

TiDB Cloud Serverless Branchingはデータをバージョン管理システムのブランチのように分岐させてデータやスキーマを別々に管理する機能です。アプリケーション開発者にとって便利な機能で、テスト時に別々のスキーマを試すことが可能になります。ブランチはTiDB Cloud Serverlessのみで使用でき、執筆時点ではベータ版となっています。ブランチについては5-3節の「ブランチを使用してデータとスキーマを別々に管理する」で説明します。

ベクトル検索機能を使用する

TiDB Cloud Serverlessではベクトル検索を使用できます。近年は生成AIが目覚ましく発展し、大規模言語モデルや画像生成モデルなどのモデルが登場しています。これらのモデルは非構造化データを扱うため、従来のデータベースでは扱いづらかった非構造化データを効率よく扱うための機能としてベクトル検索は注目されています。TiDB Cloud Serverlessのベクトル検索は、Dify^{注4}やGPTs^{注5}などのさまざまなエコシステムとの連携も可能です。詳細はドキュメントのIntegrate a Data App with Third-Party Tools^{注6}を参照してください。ベクトル検索機能については第7章で説明します。

TiDB Cloud Serverlessのアーキテクチャーを理解する

TiDB Cloud ServerlessのアーキテクチャーはAWS Storage Blogに寄稿されている「How PingCAP transformed TiDB into a serverless DBaaS using Amazon S3 and Amazon EBS^{注7}」の記事内で詳細に解説

注4 <https://docs.dify.ai/guides/tools>

注5 <https://openai.com/blog/introducing-gpts>

注6 <https://docs.pingcap.com/tidbcloud/data-service-integrations>

注7 <https://aws.amazon.com/jp/blogs/storage/how-pingcap-transformed-tidb-into-a-serverless-dbaas-using-amazon-s3-and-amazon-efs/>

されています。

記事の内容によれば、TiDBへデータが書き込まれるとTiKVノードのAmazon EBSにWAL(Write Ahead Log)が書き込まれ、同時にデータの更新内容がRaftログとして書き込まれます。次にRaftログは他のTiKVノードに伝播^{でんぱ}されます。データはTiKVノードでキャッシュされ、その後非同期でAmazon S3へ書き込まれます。最後に他のTiKVノードにデータを読み込むためのシグナルが送信され、Amazon S3からデータを読み込んでレプリケーションが完了するという流れです(記事内ではmulti-replica replicationと表現されています)。データの処理の流れ以外にも興味深い内容が多く含まれているので、興味のある方はぜひ記事を読んでみてください。

5-2 TiDB Cloud Dedicated

TiDB Cloud Dedicatedは、専用の環境を払い出してTiDBを利用できるリソース専有型のマネージドサービスです。料金体系はこちらも従量課金制ですが、TiDB Cloud Serverlessとは違い、リクエストユニットではなく作成するインスタンスタイプやストレージサイズ、トラフィック量などに応じて料金が発生します。TiDB Cloud Dedicatedには無料利用枠はありませんので、支払いのためのクレジットカードの登録が必要です。料金の詳細に関してはドキュメントのTiDB Cloud Dedicated Pricing Details^{注8}を参照してください。

TiDB Cloud Dedicatedは検証目的の場合に一定のクレジットを利用できる仕組みもあります。詳細はPerform a Proof of Concept (PoC) with TiDB Cloud^{注9}のドキュメントを参照してください。PingCAP 日本人への問い合わせはホームページの問い合わせフォーム^{注10}より可能です。

TiDB Cloud Dedicatedはカスタマイズ性が高いのが特徴です。インスタンスタイプやストレージサイズを自由に設定できます。ただしオートスケーリング機能が無いため、モニタリングツールと連携して負荷状況に応じてTiDB CloudのAPIを利用してスケールする仕組みをご自身で用意する必要があります。

TiDB Cloud Dedicatedは執筆時点ではAWSおよびGoogle Cloudで提供されています。プライベートエンドポイント経由で接続するために、AWS PrivateLink^{注11}またはPrivate Service Connect^{注12}を使用できます。また、VPC Peering^{注13}にも対応しています。

TiDB Cloud Dedicatedでは各種コンポーネントは自動で複数のアベイラビリティゾーンにまたがって

注8 <https://www.pingcap.com/tidb-dedicated-pricing-details/>

注9 <https://docs.pingcap.com/tidbcloud/tidb-cloud-poc>

注10 <https://pingcap.co.jp/contact-us/>

注11 <https://docs.pingcap.com/tidbcloud/set-up-private-endpoint-connections>

注12 <https://docs.pingcap.com/tidbcloud/set-up-private-endpoint-connections-on-google-cloud>

注13 <https://docs.pingcap.com/tidbcloud/set-up-vpc-peering-connections>

配置されます。また、TiKVコンポーネントは最低3台配置する必要があり、スケールアウトおよびスケールインする場合は3台単位で行う必要があります。

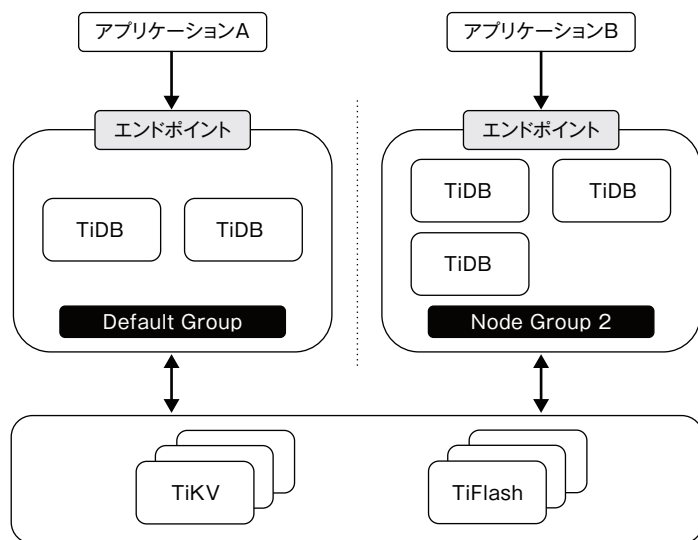
■ ビルトインアラート機能

TiDB Cloud Dedicatedではあらかじめ用意されているアラート機能を使用できます。クラスター概要画面より「Alerts」をクリックすると現在のアラートの設定状況が確認できます。アラートの内容や設定方法は次節の「ビルトインアラート機能を使用する」で説明します。

■ ノードグループ機能

ノードグループは単一TiDB Cloud Dedicatedクラスター内のTiDBサーバーを物理的に分離して配置し、それぞれ別の役割を持たせることができる機能です。ノードグループは1台以上のTiDBサーバーで構成され、接続用のエンドポイントがノードグループごとに払い出されます(図5.1)。

図5.1 TiDB Cloud Dedicatedのノードグループ機能



Default GroupはTiDB Cloud Dedicatedを作成すると自動的に作成されるデフォルトのノードグループで、削除できません。

ノードグループを追加することにより、接続するアプリケーションごとにTiDBサーバーのリソースを分離させることが可能になります。そのため、特定のアプリケーションによるTiDBサーバーのリソースの占有が、他のアプリケーションへ影響してしまう問題がある場合に特に有効です。なお、ノー

ドグループ機能はTiDBサーバーを分離しますが、ストレージクラスターであるTiKVやTiFlashを分離する機能ではありません。ストレージクラスターのリソースは変わらずアプリケーションで共有されますので注意してください。ストレージクラスターのリソース制御を実現したい場合は、リクエストユニットによるリソース制御を検討してください。リクエストユニットによるリソース制御については7-3節で説明します。

ノードグループ機能はTiDB Cloud Serverlessでは使用できません。

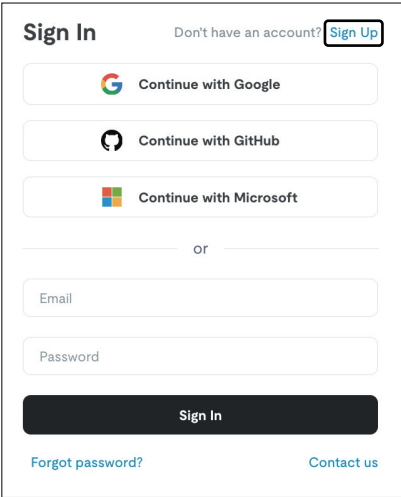
5-3 TiDB Cloudの始め方

本節ではTiDB Cloudを使うにあたっての基本を解説します。

サインインして初期クラスターを確認する

TiDB Cloudを始めるためには、TiDB Cloudのサイト^{注14}にアクセスし、アカウントを作成する必要があります。TiDB Cloudのサイトにアクセスするとまず最初にサインイン画面が表示されます。すでにアカウントがある場合は図5.2の画面でサインインしてください。アカウントがない場合は「Sign Up」をクリックしてアカウントを作成してください。執筆時点ではGoogleアカウント、GitHubアカウント、MicrosoftアカウントによるSSO(Single Sign-On)にも対応しています。

図5.2 TiDB Cloudサインイン画面



Sign In

Don't have an account? [Sign Up](#)

 Continue with Google

 Continue with GitHub

 Continue with Microsoft

or

Email

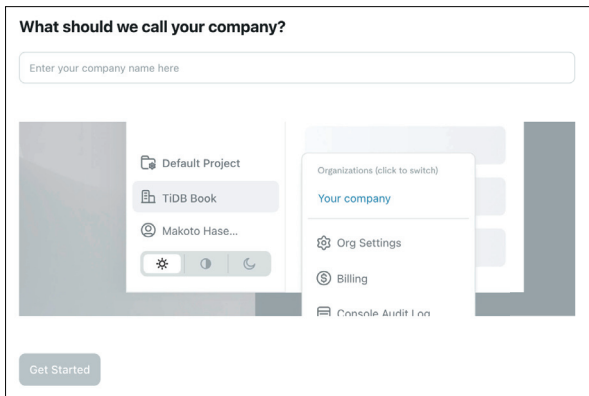
Password

Sign In

[Forgot password?](#) [Contact us](#)

注14 <https://tidbcloud.com>

図5.3 TiDB Cloudサインアップ後



サインインすると、初回は図5.3のような画面が表示されます。「Enter your company name here」に会社名を入力し、「Get Started」をクリックしてください。個人で登録する場合は分かりやすい名前にしておくといいでしょう。入力した会社名は次項で説明する組織名として使用されます。

「Get Started」をクリック後、図5.4のような画面に遷移します。自動的に Free Cluster が作成され、クラスターの詳細が表示されます。自動で作成されたクラスターは Cluster0 として表示されています。

画面右上の「Connect」ボタンをクリックすることで接続用のエンドポイントが表示されます(図5.5)。「Connection Type」では Public と Private Endpoint が選択できますが、現時点では Public のみです。Private Endpoint を使用するためには AWS PrivateLink を設定する必要があります。TiDB Cloud Dedicated の場合は VPC Peering も利用可能です。

図5.4 TiDB Cloud最初期画面

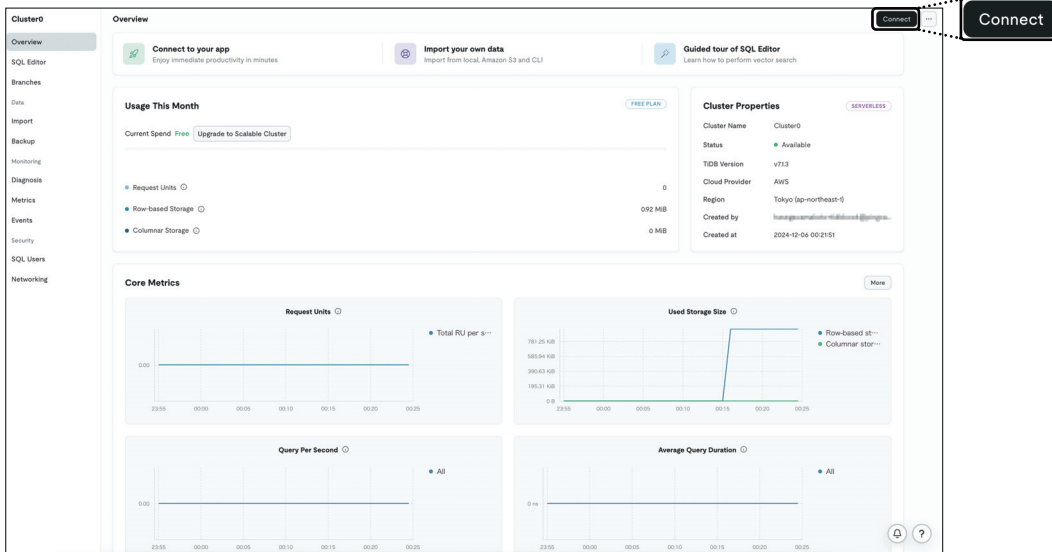
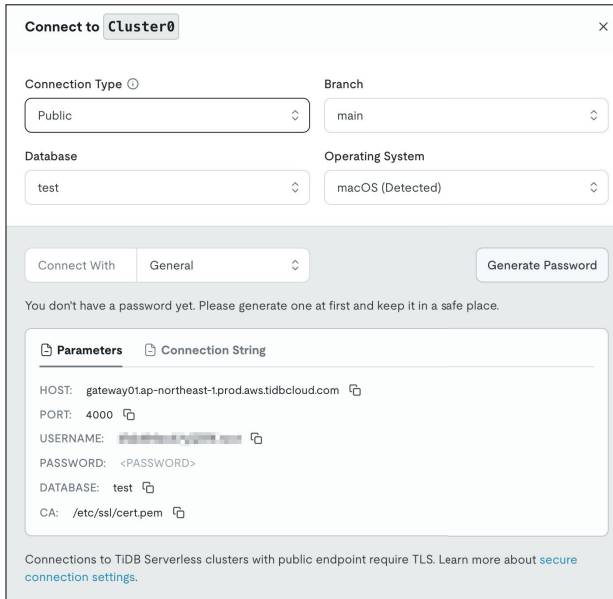


図5.5 TiDB Cloud接続画面



The screenshot shows the 'Connect to Cluster0' dialog box. It has fields for 'Connection Type' (Public), 'Branch' (main), 'Database' (test), and 'Operating System' (macOS (Detected)). There is a 'Connect With' dropdown set to 'General' and a 'Generate Password' button. Below these is a section for 'Parameters' and 'Connection String'. The 'Parameters' section shows: HOST: gateway01ap-northeast-1prod.aws.tidbcloud.com, PORT: 4000, USERNAME: [redacted], PASSWORD: <PASSWORD>, DATABASE: test, and CA: /etc/ssl/cert.pem. A note at the bottom states: 'Connections to TiDB Serverless clusters with public endpoint require TLS. Learn more about [secure connection settings](#).'

「Parameters」に表示されているHOSTやUSERNAMEの情報をを使用してTiDBに接続できますが、初期パスワードを生成する必要があります。「Generate Password」をクリックするとパスワードが生成されますので、忘れずに保存するか、ログイン後にパスワードを変更してください。「Connection String」にはさまざまなクライアントツールや言語の接続用のスニペットが表示されます。たとえば、macOS環境で「Connect With」の選択肢から「MySQL CLI」を選ぶとmysqlコマンドで接続するコマンドラインのスニペットが次のように表示されます。

払い出されたエンドポイントに接続するmysqlコマンドのスニペット

```
mysql --comments -u '<USERNAME>' -h gateway01.ap-northeast-1.prod.aws.tidbcloud.com \
-P 4000 -D 'test' --ssl-mode=VERIFY_IDENTITY --ssl-ca=/etc/ssl/cert.pem -p'<PASSWORD>'
```

これを実行した後、接続を確認します。

```
TiDB Cloud接続
mysql> SELECT VERSION();
+-----+
| VERSION() |
+-----+
| 5.7.28-TiDB-v7.1.3-serverless |
+-----+
1 row in set (0.01 sec)
```

このようにTiDB Cloudはアカウント作成後にすぐに接続できるクラスターが自動的に払い出されますので、非常に簡単にTiDBを使い始めることができます。

組織とプロジェクトを確認する

TiDB Cloudは組織とプロジェクトの概念を持っています。Webコンソールでは組織はOrganization、プロジェクトはProjectとして表記されます。組織は複数のプロジェクトを持つことができ、作成するクラスターはプロジェクトで管理されます。

組織の情報を表示するには、Webコンソール画面左下の組織名をクリックすると表示される「Organization Settings」をクリックします(図5.6)。

プロジェクトの情報を表示するには、Webコンソール画面左下のプロジェクト名をクリックすると表示される「Project Settings」をクリックしてください(図5.7)。

なお、複数の組織に所属している場合は組織をクリックすると表示される組織一覧から選ぶことで表示する組織を切り替えることができます。プロジェクトも同様に切り替えることができます。

▶ タイムゾーンの設定

Webコンソール上で表示される日時はデフォルトでUTC(協定世界時)で表示されます。そのため、クラスター作成日時やバックアップの取得日時などの表示はJST(日本標準時)より9時間遅れて表示されます。

Webコンソールのタイムゾーンを変更する場合は、Organization Settingsの画面で行います。JSTで表示するためには「Time Zone」の項目をUTC+09:00に設定してください(図5.8)。

なお、このタイムゾーンの設定はWebコンソールの表示に使用されるのみです。作成するTiDBのタイムゾーンの設定は影響されず、デフォルトのUTCのままです。

図5.6 組織の設定

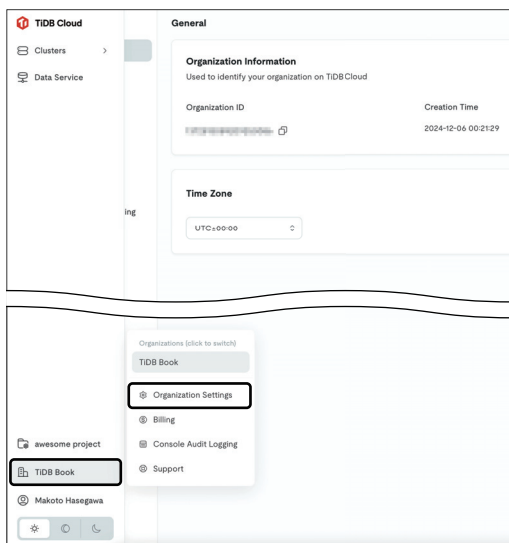


図5.7 プロジェクトの設定

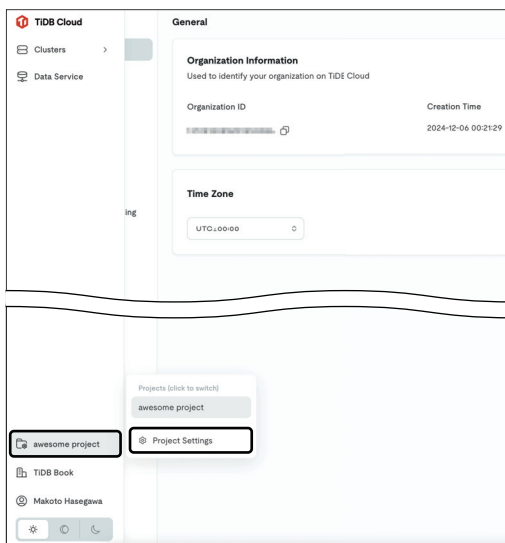


図5.8 タイムゾーンの設定

TiDB Book

General

Projects

Users

API Keys

Authentication

Billing

Console Audit Logging

Support

General

Organization Information

Used to identify your organization on TiDB Cloud

Update

Organization ID

Creation Time

2024-12-06 00:21:29

Organization Name *

TiDB Book

Time Zone

Update

UTC+09:00

Current Time: 2025-03-08 12:50:39 UTC+09:00

TiDBのタイムゾーンの確認

```
mysql> SELECT @@global.time_zone, @@global.system_time_zone;
+-----+-----+
| @@global.time_zone | @@global.system_time_zone |
+-----+-----+
| SYSTEM             | UTC                       |
+-----+-----+
1 row in set (0.01 sec)
```

▶ ユーザーとロールの設定

組織やプロジェクトの情報にアクセスするためには、組織内にユーザーとして存在しなければならず、ユーザーは適切なロールを持っていなければなりません。ロールには組織に関するロール(表5.1)とプロジェクトに関するロール(表5.2)があります。

表5.1 組織のロール一覧

組織のロール名	説明
Organization Owner	組織のすべての設定権限とProject Owner権限を持つ
Organization Viewer	組織の閲覧権限と、プロジェクトに所属している場合プロジェクトの操作権限を持つ
Organization Billing Manager	請求情報の閲覧権限と設定権限を持つ
Organization Billing Viewer	請求情報の閲覧権限を持つ
Organization Console Audit Manager	組織のコンソール操作ログの閲覧権限を持つ

表5.2 プロジェクトのロール一覧

プロジェクトのロール名	説明
Project Owner	プロジェクト内ですべての操作権限を持つ
Project Viewer	プロジェクト内でクラスター情報の閲覧権限を持つ。クラスターのデータはアクセス不可
Project Data Access Read-Write	プロジェクト内のクラスターのデータ閲覧および操作権限を持つ
Project Data Access Read-Only	プロジェクト内のクラスターのデータ閲覧権限を持つ

図5.9 組織のロールの変更

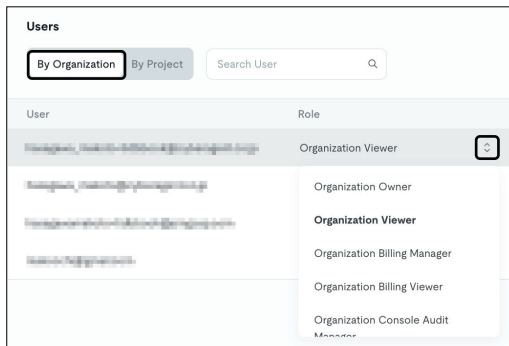
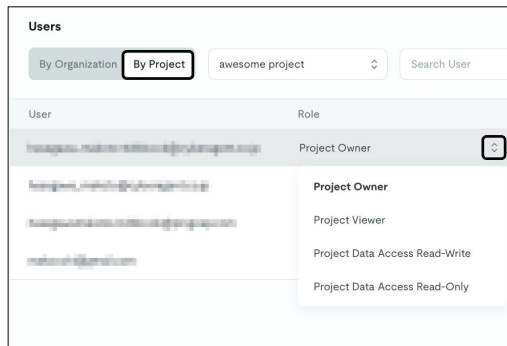


図5.10 プロジェクトのロールの変更



組織またはプロジェクトへのユーザーの招待は、Organization Settings の画面から「Users」をクリック後、画面右上にある「Invite」から行います。招待にはメールアドレスが必要になります。招待する際にどのロールを割り当てるのかも同時に設定します。組織に追加するだけであれば「By Organization」タブが選択されている状態で招待して組織のロールを割り当てます。その場合は招待されたユーザーは組織内のプロジェクトに対する権限をまだ持っていない状態になります（Organization Owner ロールを付与した場合はすべてのプロジェクトの操作が可能になります）。プロジェクトのロールを割り当てるには「By Project」タブが選択されている状態で招待してプロジェクトのロールを割り当てます。組織内に居ないユーザーをプロジェクトに追加すると、組織のロールとしては Organization Viewer が割り当てられます。

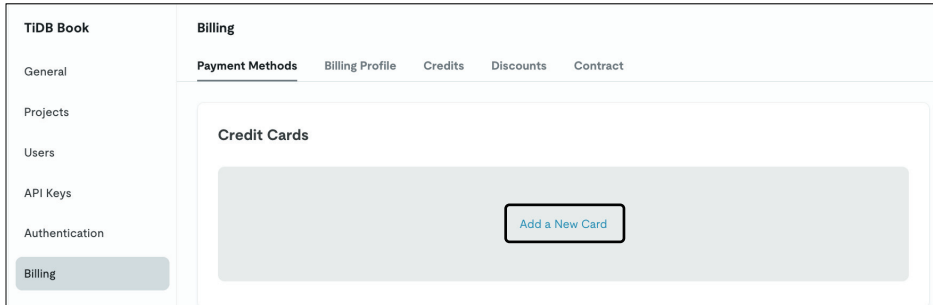
割り当てたロールの変更も可能です。Users のページから、Role の列の右側にある矢印をクリックすることでロールを変更できます（図5.9、図5.10）。ユーザーの組織のロールを変更する場合は Organization Owner のロールが必要です。また、ユーザーのプロジェクトのロールの変更にはそのプロジェクトで Project Owner のロールを保持している必要があります。

組織内には複数のプロジェクトを作成できます。Organization Settings の画面から「Projects」をクリックして、画面右上にある「Create New Project」をクリックします。プロジェクト名を入力して作成しますが、請求情報の設定をしていない場合は組織内にプロジェクトは1つしか作れないため、デフォルトで用意されている「default project」以外は作成できませんので注意してください。請求情報の設定は次節で説明します。

プロジェクト名は自由に変更できます。Projects の画面で名前を変更したいプロジェクトの「Actions」の列の3点リーダーをクリックして表示される「Rename」から変更してください。初期で用意されている「default project」も名前を変更できます。なお、「Rename」以外に「Invite」も表示されますので、こちらの画面からユーザーをプロジェクトに招待することもできます。

▶ 請求情報の設定

支払いのためのクレジットカードを登録するためには、Organization Settings の画面から「Billing」を

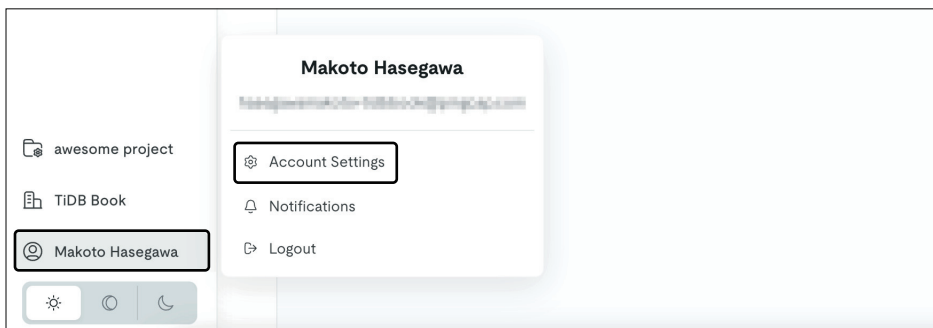
図5.11 クレジットカードの登録

クリックしてください。「Add a New Card」をクリックするとクレジットカードの情報を入力する画面が表示されます。クレジットカードの情報を入力して「Save Card」をクリックしてください(図5.11)。なお、「Billing」画面はOrganization Owner ロールを持つユーザーのみ表示されます。

アカウントを設定する

アカウントに関する設定は、Web コンソール画面の左下に表示されているアカウント名をクリックすると表示される「Account Settings」で行います(図5.12)。

Account Settings では表示上の名前を変更できます。メールアドレスの変更はできません。SSO を使用していない場合はパスワードの変更もこちらの画面から可能です。また、アカウントのセキュリティ向上のために2要素認証(Two Factor Authentication)も有効にできます。2要素認証を有効にすると、次回ログイン時に追加の認証をするためのQRコードが表示されます。QRコードは認証アプリケーションである Google Authenticator^{注15}や Microsoft Authenticator^{注16}などで読み取ります。QRコードを読み取った後に表示されるコードを入力することで2要素認証が有効になります。

図5.12 アカウントの設定

注15 <https://support.google.com/accounts/answer/1066447?hl=JA>

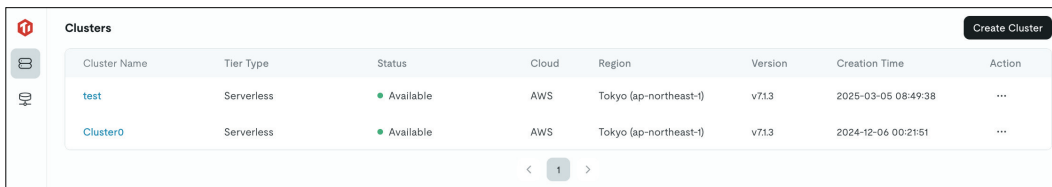
注16 <https://www.microsoft.com/ja-jp/security/mobile-authenticator-app>

クラスター概要画面を表示する

Web コンソール画面左側にある「Clusters」アイコンをクリックするとプロジェクト内に存在するクラスターの一覧が表示されます(図5.13)。

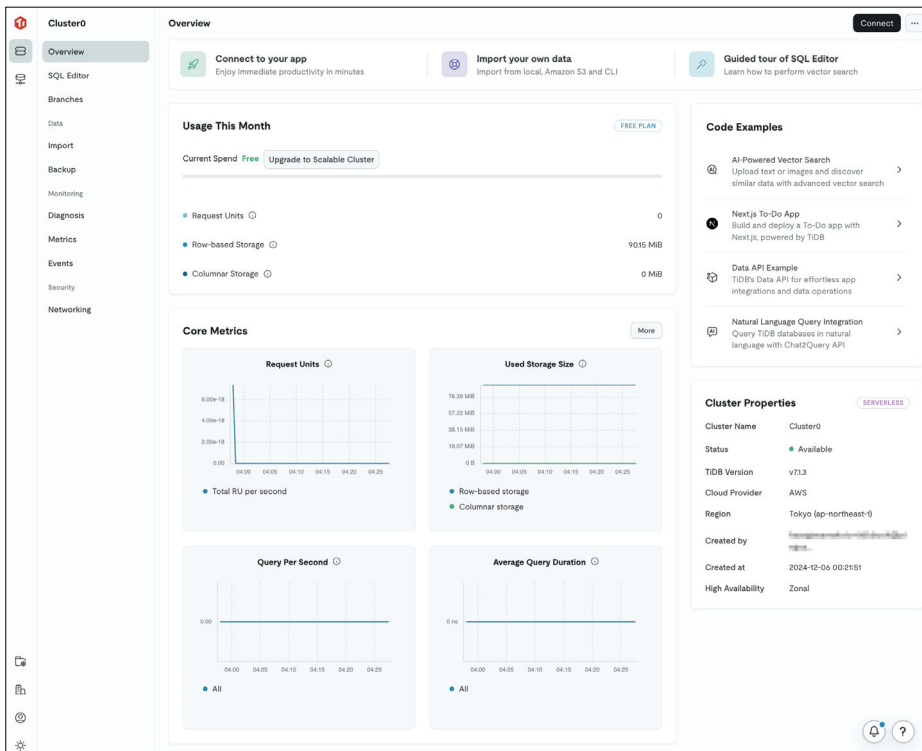
クラスター名をクリックするとクラスターの概要画面が表示されます(図5.14)。概要画面では「Usage This Month」「Cluster Properties」「Core Metrics」が表示されます。「Usage This Month」では今月の利用状況が表示されます。「Cluster Properties」ではクラスターの名前やステータス、TiDBのバージョンなどが表示されます。「Core Metrics」ではリクエストユニットの使用状況や秒間のリクエスト数、ストレージの使用量、平均クエリ実行時間が表示されます。

図5.13 クラスター一覧画面



Cluster Name	Tier Type	Status	Cloud	Region	Version	Creation Time	Action
test	Serverless	Available	AWS	Tokyo (ap-northeast-1)	v71.3	2025-03-05 08:49:38	...
Cluster0	Serverless	Available	AWS	Tokyo (ap-northeast-1)	v71.3	2024-12-06 00:21:51	...

図5.14 クラスター概要画面



SQL Editorを使用する

TiDB CloudのWebコンソールにはSQL Editorが搭載されており、ブラウザ上でSQLを実行できます。また、SQL Editor内ではAIによる自然言語からSQLへの変換機能(Chat2Query)を搭載しており、自然言語からSQLを生成できます。生成したSQLはそのままSQL Editorで実行可能です。

SQL Editorを使うためには、まず始めにデータベースを選択する必要があります。ここでは「test」データベースを選択します。SQL EditorでUSE test;と入力し、その後に`[Ctrl] + [I]`(macOSの場合は`[Command] + [I]`)を入力するとChat2Queryの入力画面が表示されます(図5.15)。

入力欄に「IDと名前を持つ従業員テーブルを作成して」と入力して`[Enter]`を押してみましょう。するとSQLが生成されます(図5.16)。

IDはAUTO_RANDOMを使うように指示してみましょう。AUTO_RANDOMは自動的にランダムかつ一意な値を生成するTiDB独自の列属性です。先ほどの入力欄に続けて「IDはBIGINTでAUTO_RANDOMを使用して」と入力して`[Enter]`を押してみましょう。SQLが再生成されます(図5.17)。

IDがAUTO_RANDOMに変更されたSQLが生成されました。実際にSQLを実行してテーブルを作成してみましょう。生成されたSQLを採用する場合は入力欄の下にある「Accept」をクリックします。その後画面右上にある「Run」をクリックするか、`[Ctrl] + [Enter]`(macOSの場合は`[Command] + [Enter]`)を入力するとSQLが実行されます。実行結果は画面下部に表示されます。SQLを実行したことによりtestデータベースにemployeeテーブルが作成されました。SQL入力画面の左側のSchemasでデータベース内の情報を確認できます(図5.18)。なお、日本語のSQLのコメント文は分かりやすいように後から追加したものになります。Chat2Queryによる出力ではありませんのでご了承ください。

図5.15 Chat2Queryの入力1

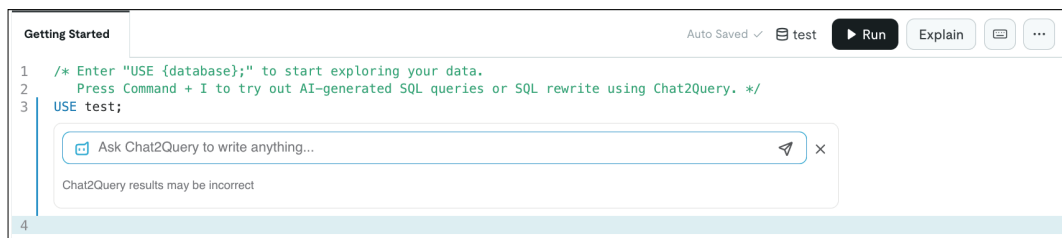


図5.16 Chat2Queryの入力2

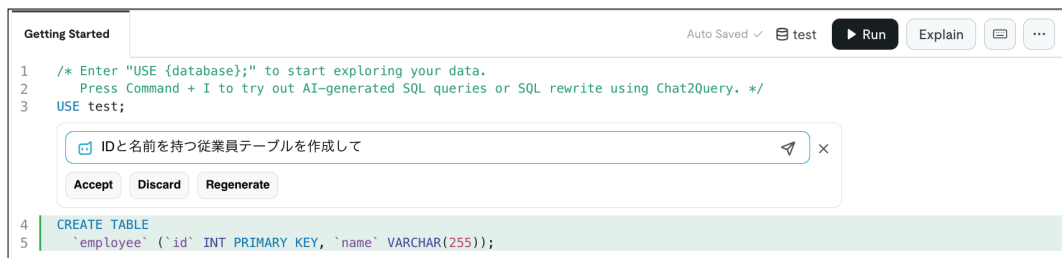


図5.17 Chat2Queryの入力3

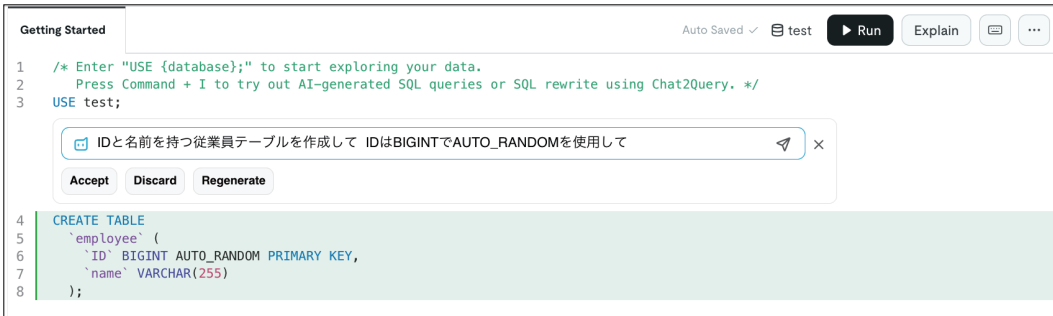
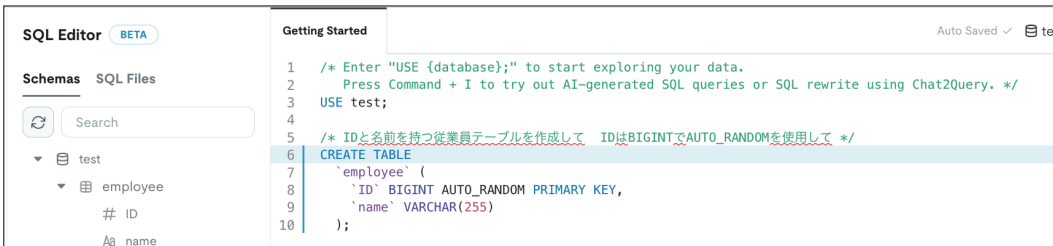


図5.18 Chat2Queryの入力4



それでは次に、実際にデータを投入してみましょう。データの投入も Chat2Query を使って行います。SQL Editor 内の `);` の右に移動し、`Enter` を押して新しい行に移動します。その後 Chat2Query を呼び出します。Chat2Query には「従業員テーブルに 10 件の名前を投入して 名前は日本人の姓をランダムに使用して」と入力して `Enter` を押します (図 5.19)。

図5.19 Chat2Queryの入力5



図 5.20 Chat2Queryの入力6

Getting Started

Auto Saved ✓ test ▶ Run Explain ⓘ ⋮

```

1  /* Enter "USE (database);" to start exploring your data.
2  Press Command + I to try out AI-generated SQL queries or SQL rewrite using Chat2Query. */
3  USE test;
4
5  /* IDと名前を持つ従業員テーブルを作成して IDはBIGINTでAUTO_RANDOMを使用して */
6  CREATE TABLE
7  `employee` (
8    `id` BIGINT AUTO_RANDOM PRIMARY KEY,
9    `name` VARCHAR(255)
10 );
11
12 /* 従業員テーブルに10件の名前を投入して 名前は日本人の名前ランダムに使用して */
13 INSERT INTO
14 `employee` (`name`)
15 VALUES
16 ('佐藤'),
17 ('鈴木'),
18 ('高橋'),
19 ('田中'),
20 ('渡辺'),
21 ('伊藤'),
22 ('山本'),
23 ('中村'),
24 ('小林'),
25 ('加藤');
26
27 /* 従業員テーブルのデータを表示して */
28 SELECT
29 `id`,
30 `name`
31 FROM
32 `employee`;

```

Query log

Result Chart

1 SELECT 'id', 'name' FRO...

Duration 28 ms

id	name
8935141660703064065	佐藤
8935141660703064066	鈴木
8935141660703064067	高橋
8935141660703064068	田中
8935141660703064069	渡辺
8935141660703064070	伊藤
8935141660703064071	山本
8935141660703064072	中村
8935141660703064073	小林
8935141660703064074	加藤

先ほどと同じように「Accept」を押した後に実行します。なお、実行されるSQLは行番号の右側が青くなっている部分だけになります。ですので先ほど生成したCREATE TABLE ... は実行されません。実際にINSERTされたデータを表示してみましょう。こちらもChat2Queryを使って「従業員テーブルのデータを表示して」と入力して実行します(図5.20)。結果は画面下部に表示されます。

次は少し複雑な構造を読み込ませて、SQLも複雑になる例を見てみましょう。TiUPを使用して仮想オンライン書店アプリケーションのデータを準備します。詳細は3-10節を参照してください。データベースはbookshopを使用するようにします。ユーザー名とパスワードはTiDB Cloudのクラスターのものを使用します。

TiUPを使用してデモ用データセットを作成

```

$ tiup demo bookshop prepare --db bookshop --host gateway01.ap-northeast-1.prod.aws.tidbcloud.com --port 4000 --user '<USERNAME>' --password '<PASSWORD>'
INFO[0000] Creating the tables if not existed...      dataset=bookshop
INFO[0000] Creating table books.                      dataset=bookshop
INFO[0000] Creating table users.                     dataset=bookshop
INFO[0000] Creating table authors.                   dataset=bookshop

```

```
INFO[0001] Creating table book_authors. dataset=bookshop
INFO[0001] Creating table orders. dataset=bookshop
INFO[0001] Creating table ratings. dataset=bookshop
INFO[0001] Finished creating tables! dataset=bookshop
INFO[0001] Clearing the old data... dataset=bookshop
INFO[0002] Loading users data... dataset=bookshop
INFO[0003] Loading books data... dataset=bookshop
INFO[0006] Loading authors data... dataset=bookshop
INFO[0007] Loading book authors data... dataset=bookshop
INFO[0008] Loading book orders data... dataset=bookshop
INFO[0031] Loading book ratings data... dataset=bookshop
INFO[0054] Finished! dataset=bookshop
```

データ投入後、SQL EditorのSchemasでbookshopデータベースと各種テーブルが作成されていることを確認します。表示されない場合はリフレッシュを押してください(図5.21)。

データの中身を確認する前に、新しいSQL Editorの作業スペース(SQL File)を用意します。SQL Fileを作成するために「SQL Files」のタブを選択し、プラスマークのボタンをクリックします(図5.22)。

クリックすると新しいSQL Fileが作成されます。新規SQL FileはNew Queryという名前で作成されます。SQL Fileのタイトルを分かりやすいように「bookshop」に変更します。New Queryの右側にある3点リーダーをクリックし「Rename」を選択します(図5.23)。その後「bookshop」と入力して **Enter** を押します。

新しい作業用のSQL Fileが作成されたので、先ほどと同じようにUSE bookshop; と入力して **Enter** を押します。その後にChat2Queryを呼び

図5.21 Chat2Queryの入力 bookshop編1

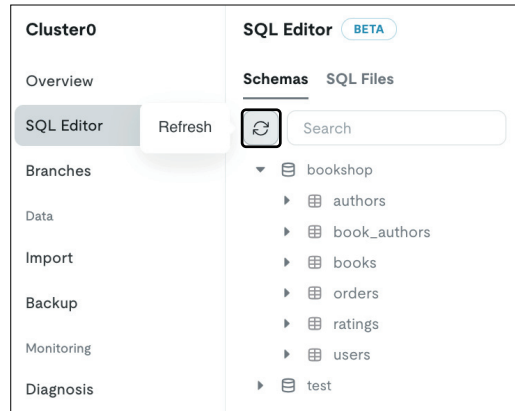


図5.22 Chat2Queryの入力 bookshop編2

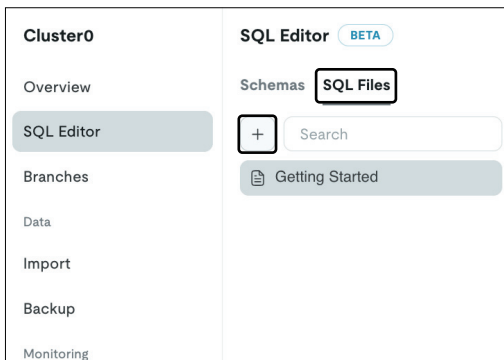


図5.23 Chat2Queryの入力 bookshop編3

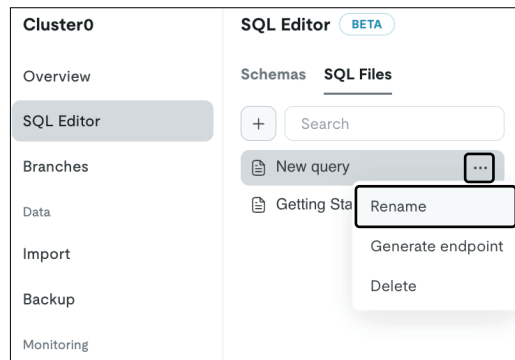


図 5.24 Chat2Queryの入力 bookshop編4

The screenshot shows the Chat2Query interface with a SQL query entered in the editor. The query is a SELECT statement that joins the 'books', 'ratings', 'book_authors', and 'authors' tables to find the top 10 books by total score. The results are displayed in a table below the query editor.

```

1 /* Enter "USE {database};" to start exploring your data.
2 Press Command + I to try out AI-generated SQL queries or SQL rewrite using Chat2Query. */
3 USE bookshop;
4
5 /* 合計スコアが高い順に本と著者を10件表示して */
6 SELECT
7   b.`title`,
8   a.`name`,
9   SUM(r.`score`) AS total_score
10 FROM
11   `books` b
12 JOIN `ratings` r ON b.`id` = r.`book_id`
13 JOIN `book_authors` ba ON b.`id` = ba.`book_id`
14 JOIN `authors` a ON ba.`author_id` = a.`id`
15 GROUP BY
16   b.`id`,
17   a.`id`
18 ORDER BY
19   total_score DESC
20 LIMIT
21   10;

```

Query log	Result	Chart
1 SELECT b.`title`, a.`name`...	title	name
Duration 363 ms	The Documentary of spider	Leola Grady
	The History of Civic Impulse LLC	Kameron Leffler
		92

出し、「合計スコアが高い順に本と著者を10件表示して」と入力して **Enter** を押します。生成されたSQLを実行すると、本のタイトルと著者名が表示されます(図 5.24)。

続いて「著者の性別が男であり生存している最も料金の高い本をオーダーしたユーザーをオーダー日が新しい順に5人抽出 最終結果にオーダー日も表示」と入力してみましょう(図 5.25)。

このように多少複雑なSQLでもChat2Queryを使うことで簡単にSQLを生成できます。

Chat2Queryは執筆時点ではベータ版の機能ですので、予告なく仕様変更が行われる可能性があります。使用する際は注意してください。

図 5.25 Chat2Queryの入力 bookshop編5

The screenshot shows the Chat2Query interface with a complex SQL query entered in the editor. The query is a SELECT statement that joins the 'users', 'orders', 'books', 'book_authors', and 'authors' tables to find the top 5 users by the highest price of books they ordered, filtered by male authors and non-null death years. The results are displayed in a table below the query editor.

```

1 /* Enter "USE {database};" to start exploring your data.
2 Press Command + I to try out AI-generated SQL queries or SQL rewrite using Chat2Query. */
3 USE bookshop;
4
5 /* 著者の性別が男であり生存している最も料金の高い本をオーダーしたユーザーをオーダー日が新しい順に5人抽出 最終結果にオーダー日も表示 */
6 SELECT
7   b.`title`,
8   a.`name`,
9   SUM(r.`score`) AS total_score
10 FROM
11   `books` b
12 JOIN `ratings` r ON b.`id` = r.`book_id`
13 JOIN `book_authors` ba ON b.`id` = ba.`book_id`
14 JOIN `authors` a ON ba.`author_id` = a.`id`
15 GROUP BY
16   b.`id`,
17   a.`id`
18 ORDER BY
19   total_score DESC
20 LIMIT
21   10;
22
23 /* 著者の性別が男であり生存している最も料金の高い本をオーダーしたユーザーをオーダー日が新しい順に5人抽出 最終結果にオーダー日も表示 */
24 SELECT
25   `users`.`id`,
26   `users`.`nickname`,
27   `orders`.`ordered_at`
28 FROM
29   `users`
30 JOIN `orders` ON `users`.`id` = `orders`.`user_id`
31 JOIN `books` ON `orders`.`book_id` = `books`.`id`
32 JOIN `book_authors` ON `books`.`id` = `book_authors`.`book_id`
33 JOIN `authors` ON `book_authors`.`author_id` = `authors`.`id`
34 WHERE
35   `authors`.`gender` = 0
36 AND `authors`.`death_year` IS NULL
37 AND `books`.`price` = {
38   SELECT
39     MAX(`price`)
40   FROM
41     `books` b
42 JOIN `book_authors` ba ON b.`id` = ba.`book_id`
43 JOIN `authors` a ON ba.`author_id` = a.`id`
44 WHERE
45   a.`gender` = 0
46 AND a.`death_year` IS NULL
47 }
48 ORDER BY
49   `orders`.`ordered_at` DESC
50 LIMIT
51   5;

```

Query log	Result	Chart
1 SELECT `users`.`id`, `user`...	id	nickname
Duration 74 ms	1932379220	Kirin4773
	2626934635	Romaguera7032
	1082337882	Terry7357
	1739179061	Smith6612
	2626934635	Romaguera7032
		ordered_at
		2024-04-16 04:42:58
		2020-02-21 04:12:19
		2019-08-26 01:30:04
		2019-12-05 05:52:47
		2017-10-08 01:19:42

ブランチを使用してデータとスキーマを別々に管理する

TiDB Cloud Serverlessでは、データおよびスキーマ情報をコピーオンライトを使用して分岐させることが可能なTiDB Cloud Serverless Branchingが提供されています。この機能を使用することで、バージョン管理システムにおけるブランチのように分離された環境が提供されるため、元のクラスターへの影響を心配することなく自由にデータの操作が可能になります。なお、TiDB Cloud Serverless Branchingは執筆時点ではベータ版の機能です。

TiDB Cloud Serverless Branchingを使用するには、クラスター概要画面の左側にある「Branches」をクリックします。Branches画面では現在のブランチの一覧が表示されます(図5.26)。新しいブランチを作成するには画面右上にある「Create Branch」をクリックします。ブランチ名を入力し、「Create」をクリックします。ブランチが作成されるとBranches画面に作成したブランチが表示されますので、Statusの列がActiveになるまで待ちます。

ブランチを作成する際には、分岐元のデータの任意の時点を指定できますが、Free Clusterの場合は過去24時間以内で選択可能です。Scalable Clusterの場合は過去14日間以内で選択可能です。

作成されたブランチへ接続してみましょう。Action列の3点リーダーをクリックして「Connect」を選択します(図5.27)。

図5.26 ブランチ一覧画面

Branches BETA						Create Branch	Connect to GitHub
Name	Parent	Status	Row Storage	Creation Time	Action		
test branch	main	Active	8915 MiB	2024-12-09 07:48:11	...		

図5.27 ブランチへの接続

Branches BETA						Create Branch	Connect to GitHub
Name	Parent	Status	Row Storage	Creation Time	Action		
test branch	main	Active	8915 MiB	2024-12-09 07:48:11	...		
						Connect Create branch Reset Rename Delete	

コピーオンライトとは

column

コピーオンライト(Copy-On-Write)は、データを複製する際に実体をコピーするのではなく、元のデータを参照するポインタをコピーする技術です。コピー先でデータが変更されると、そのデータのみをコピーして新しいデータを作成します。この技術を使用することで、データの複製にかかる時間やストレージの使用量を削減できます。COW、またはCoWと略されることもあります。

本節の始めに確認したユーザー名とは異なるユーザー名が払い出されていることが確認できます。新しく払い出されたブランチ接続用のユーザー情報を使用して接続しますが、その前に「Generate Password」をクリックしてブランチ接続用のユーザーのパスワードを生成してください。

ブランチへ接続したら test データベースを DROP (削除) してみます。

ブランチに接続して test データベースを DROP

```
$ mysql -u '<ブランチ接続用のUSERNAME>' -h gateway01.ap-northeast-1.prod.aws.tidbcloud.com -P 4000 \
--ssl-mode=VERIFY_IDENTITY --ssl-ca=/etc/ssl/cert.pem -p'<ブランチ接続用のPASSWORD>'

mysql> DROP DATABASE test;
Query OK, 0 rows affected (0.35 sec)

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| INFORMATION_SCHEMA |
| PERFORMANCE_SCHEMA |
| mysql |
+-----+
3 rows in set (0.01 sec)
```

その後、最初に払い出された接続情報で再度接続します。test データベースが存在し、分岐元のデータには影響が無いことが確認できます。

ブランチ元のデータに接続

```
$ mysql -u '<USERNAME>' -h gateway01.ap-northeast-1.prod.aws.tidbcloud.com -P 4000 --ssl-mode=V \
ERIFY_IDENTITY --ssl-ca=/etc/ssl/cert.pem -p'<PASSWORD>'

mysql> show databases;
+-----+
| Database |
+-----+
| INFORMATION_SCHEMA |
| PERFORMANCE_SCHEMA |
| mysql |
| test |
+-----+
4 rows in set (0.02 sec)
```

作成したブランチは、それを元にしてさらにブランチを作成できます。「Create Branch」をクリックした後に「Parent Branch」の部分をも main からブランチに変更してください。またはブランチ一覧画面から、分岐させたいブランチの Action 列にある 3 点リーダーをクリックし「Create Branch」を選択してください。

ブランチは組織内ですべてのクラスターに渡って最大 5 つまで作成できます。また、Free Cluster では各ブランチ 10GiB まで、Scalable Cluster では各ブランチ 100GiB までストレージを使用できます。

図5.28 ブランチの削除

Branches BETA Create Branch Connect to GitHub					
Name	Parent	Status	Row Storage	Creation Time	Action
test branch 5	test branch 4	Active	88.81 MiB	2024-12-09 08:41:24	...
test branch 4	test branch 3	Active	8919 MiB	2024-12-09 08:38:57	...
test branch 3	test branch 2	Active	8917 MiB	2024-12-09 08:36:42	...
test branch 2	test branch	Active	8916 MiB	2024-12-09 08:33:25	...
test branch	main	Active	8915 MiB	2024-12-09 07:48:11	...

ブランチ内のデータを分岐元の最新のデータに同期するには、リセットを使用します。先ほど test データベースを DROP したブランチをリセットしてみましょう。リセットしたいブランチの Action 列にある 3 点リーダーをクリックし、「Reset」を選択します。Status が Restoring になり、しばらくすると Active になります。リセットに伴い、パスワードも初期化されていますので再度「Generate Password」でパスワードを生成してください。再度接続すると、test データベースが存在していることを確認できます。

その他の操作として、ブランチのリネームや削除もできます。ブランチを削除する際は、削除したいブランチを派生元としているブランチが存在していると削除できませんので注意してください。たとえば図5.28のようなブランチを持っている場合、削除できるブランチは「test branch 5」のみです。

TiDB Cloud Serverless Branching はアプリケーション開発者にとって非常に便利な機能です。GitHub と連携し、Pull Request ごとにブランチを作成してテストする仕組みも可能です。詳細はドキュメントの Integrate TiDB Cloud Serverless Branching (Beta) with GitHub^{注17} を参照してください。

データをバックアップおよび復元する

バックアップの設定変更およびバックアップからの復元はクラスター概要画面の左側にある「Backup」より行います。TiDB Cloud ではデータのバックアップ、復元をサポートしていますが、使用しているクラスターの種類によってできることが異なります。

▶ TiDB Cloud Serverless の場合

TiDB Cloud Serverless ではデータのバックアップは自動で行われ、手動で行うことはできません。また、バックアップの無効化もできません。Free Cluster のバックアップは設定された時刻に毎日実行されます。バックアップの開始時刻は変更できません。Scalable Cluster のバックアップも Free Cluster と同様に設定された時刻に毎日実行されますが、開始時刻は設定変更できます。

バックアップの保持期間は Free Cluster は 1 日、Scalable Cluster は 14 日です。どちらも保持日数は変更できません。執筆時点では料金は無料です。

注17 <https://docs.pingcap.com/tidbcloud/branch-github-integration>

バックアップからの復元は、稼働中のクラスターまたは新規のクラスターに対して行えます。稼働中のクラスターに対して復元を行った場合、復元中はクラスターに接続できなくなりますので注意してください。

Scalable Cluster はバックアップの保持期間内であればポイントインタイムリストア(一般的にはポイントインタイムリカバリーと呼ばれます)に対応しています。Free Cluster はバックアップからのスナップショット復元のみサポートしています。

TiDB Cloud Dedicated の場合

TiDB Cloud Dedicated ではスナップショットバックアップによるフルバックアップと、ログバックアップによる更新データのバックアップが可能です。ログバックアップはポイントインタイムリストアをする場合に必須になります。デフォルトではスナップショットバックアップは有効になっており、ログバックアップは無効になっています。

スナップショットバックアップを有効にしている場合は定期的に行われ、デフォルトではクラスターが作成されたリージョンのタイムゾーンにおける午前2時に取得されます。取得時刻は設定で変更可能で、スナップショットバックアップ自体の無効化もできます。

ポイントインタイムリストアのためにログバックアップを有効にするためには、Backup 画面の右上にある「Backup Setting」ボタンをクリックして「Point-in-time Restore」を有効にします。注意点として、有効にしてもすぐポイントインタイムリストアが可能になるわけではなく、一度スナップショットバックアップが実行される必要があります。そのため、ログバックアップを有効にした際は、手動ですぐにスナップショットバックアップを取得しておくことをおすすめします。手動でスナップショットバックアップを取得する場合は、Backup 画面の右上にある3点リーダーをクリックして「Manual Backup」を選択してください。

デフォルトではバックアップデータはクラスターと同じパブリッククラウド上のリージョン内に保存されますが、異なるリージョンへバックアップデータを複製するデュアルリージョンバックアップもサポートしています。有効にする場合はBackup Setting 画面の「Dual Region Backup」を有効にしてください。

バックアップの保存期間は7日間から90日間の間で設定可能で、デフォルトは7日間です。バックアップは別途料金がかかりますので注意してください。料金の詳細はドキュメントの Data Backup Cost^{注18}を参照してください。

稼働しているクラスターに対してバックアップデータの復元はできませんので注意してください。復元時は新規クラスターを作成する必要があります。

ビルトインアラート機能を使用する

TiDB Cloud Dedicated では、あらかじめ組み込まれているアラート機能を使用できます。現在のアラ

注18 <https://www.pingcap.com/tidb-dedicated-pricing-details/#backup-storage-cost>

図5.29 TiDB Cloudビルトインアラート一覧画面

ートの設定状況は画面の左側の「Alerts」で確認できます(図5.29)。

アラートは3つのカテゴリに分かれており、それぞれ「Resource Usage」「Data Migration」「Changefeed」に分類されています。それぞれのアラートは個別に有効、無効(ON、OFF)が選択できるようになっており、しきい値の調整も可能です。しきい値を調整する場合はアラート項目の右側の矢印を選択し、新しい値を入力後に「Save」をクリックします(図5.30)。

表5.3はビルトインアラートの項目とデフォルトの状態を一覧にしたものです。運用状況に合わせて適切なしきい値を設定してください。

図5.30 TiDB Cloudビルトインアラート編集画面

表5.3 ビルトインアラートの項目とデフォルトの状態

カテゴリ	アラート項目	デフォルトの状態
Resource Usage	Total TiDB node memory utilization across cluster exceeded 70% for 10 minutes	ON
Resource Usage	Total TiKV node memory utilization across cluster exceeded 70% for 10 minutes	ON
Resource Usage	Total TiFlash node memory utilization across cluster exceeded 70% for 10 minutes	ON
Resource Usage	Total TiDB node CPU utilization exceeded 80% for 10 minutes	ON
Resource Usage	Total TiKV node CPU utilization exceeded 80% for 10 minutes	ON
Resource Usage	Total TiFlash node CPU utilization exceeded 80% for 10 minutes	ON
Resource Usage	TiKV storage utilization exceeds 80%	ON
Resource Usage	TiFlash storage utilization exceeds 80%	ON
Resource Usage	Max memory utilization across TiDB nodes exceeded 70% for 10 minutes	OFF
Resource Usage	Max memory utilization across TiKV nodes exceeded 70% for 10 minutes	OFF
Resource Usage	Max CPU utilization across TiDB nodes exceeded 80% for 10 minutes	OFF
Resource Usage	Max CPU utilization across TiKV nodes exceeded 80% for 10 minutes	OFF
Data Migration	Data migration job met error during data export	ON
Data Migration	Data migration job met error during data import	ON
Data Migration	Data migration job met error during incremental data migration	ON
Data Migration	Replication lag is larger than 600 seconds and stilling increasing for more than 20 minutes	ON
Changefeed	Changefeed latency is more than 600 seconds	ON
Changefeed	Changefeed status is failed	ON
Changefeed	Changefeed status is warning	ON

アラートの通知はメールで受信できます。アラート通知の設定は「Project Settings」から「Alert Subscription」を選択します。画面右上の「Add Subscriber」をクリックしてメールアドレスを登録します。プロジェクト内に複数のクラスターが存在している場合はすべてのアラートが通知されます。

ビルトインアラート以外にアラートを設定したい場合は次項の「サードパーティメトリクスと連携する」を参照してください。

サードパーティメトリクスと連携する

サードパーティメトリクス連携を使用すると、Datadog^{注19}やNew Relic^{注20}と連携が可能になり、自由にアラートをカスタマイズできます。サードパーティ側でAPIキーを作成し、「Project Settings」の「Integrations」画面で入力してください。どのようなメトリクスが収集されるかは次のドキュメントを参照してください。

注19 <https://www.datadoghq.com/>

注20 <https://newrelic.com/>

- Integrate TiDB Cloud with Datadog (Beta)^{注21}
- Integrate TiDB Cloud with New Relic (Beta)^{注22}

また、Prometheus用の設定も払い出すことができますので、外部のPrometheusからメトリクスを収集できます。収集できるメトリクスの詳細は次のドキュメントを参照してください。

- Integrate TiDB Cloud with Prometheus and Grafana (Beta)^{注23}

収集したメトリクスを表示するためのGrafanaのダッシュボードの設定もGitHub上で公開^{注24}されていますので、Prometheus連携をする際には参考にしてください。

サードパーティメトリクス連携はTiDB Cloud Dedicatedのみサポートされています。また、サードパーティメトリクス連携は執筆時点ではベータ版となっています。

■ クラスターを削除する

クラスターの削除はクラスター一覧画面のAction列にある3点リーダーをクリックし、「Delete」を選択します。またはクラスター概要画面の右上にある3点リーダーをクリックし、「Delete」を選択します(図5.31、図5.32)。

TiDB Cloud Serverlessは、クラスターを削除するとバックアップデータも含めて削除され復元できなくなりますので注意してください。データを保存しておきたい場合はダンプを取得して、TiDB Cloud外へ保存しておく必要があります。

TiDB Cloud Dedicatedでは、クラスターを削除したとしても自動で取得されたバックアップデータは

注21 <https://docs.pingcap.com/tidbcloud/monitor-datadog-integration>

注22 <https://docs.pingcap.com/tidbcloud/monitor-new-relic-integration>

注23 <https://docs.pingcap.com/tidbcloud/monitor-prometheus-and-grafana-integration>

注24 <https://github.com/pingcap/docs/blob/master/tidb-cloud/monitor-prometheus-and-grafana-integration-grafana-dashboard-UI.json>

図5.31 クラスター削除1

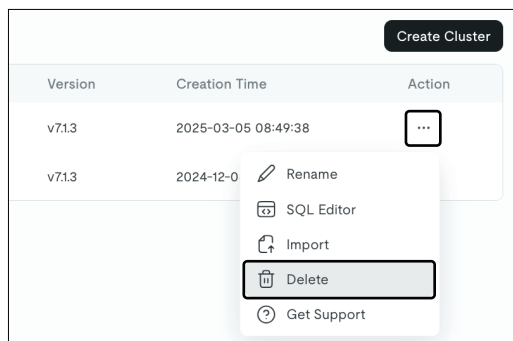
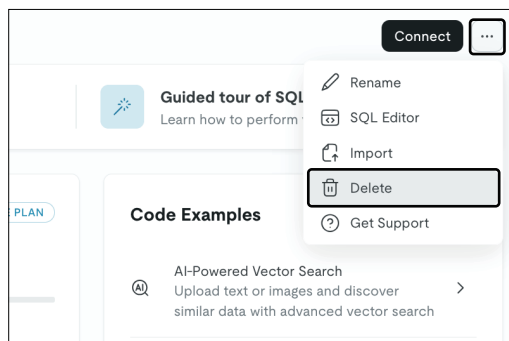


図5.32 クラスター削除2



設定された期間保持されます。手動でバックアップしたデータは削除されず、手動で削除するかアカウントが削除されるまで保持されます。そのため、クラスターを削除してもデータを復元できますが、バックアップデータは削除されるまで料金が発生し続けますので注意してください。

5-4 TiDB Cloudの稼働状況の確認

TiDB Cloudはサービスの稼働状況を確認するためのステータスページを提供しています。ステータスページはstatus.tidbcloud.com^{注25}からアクセスできます。ステータスページでは過去90日間の各サービスの稼働状況と障害情報が表示されます。各サービスは次の3つになります。

- TiDB Cloud Portal Backend
- TiDB Cloud Portal Frontend
- TiDB Cloud Serverless

また、ステータスページの「SUBSCRIBE TO UPDATES」から、ステータスの更新情報をリアルタイムで受け取ることができますので必要に応じて設定してください。

ステータスページではTiDB Cloud Dedicatedの情報は表示されません。TiDB Cloud Dedicatedの稼働状況を確認するには、TiDB Cloud Dedicatedのクラスター概要画面の「Core Metrics」を確認するか、TiDB Cloud Dedicatedのビルトインアラート機能を使用してください。

5-5 TiDB Cloud CLIの使い方

TiDB CloudではCLI(Command Line Interface)が提供されています。CLIを使用することで、コマンドラインからTiDB Cloudを操作できます。執筆時点ではTiDB Cloud CLIはベータ版ですので、今後のアップデートで仕様が変更される可能性があります。

インストールする

TiDB Cloud CLIをインストールする方法は2つあります。1つ目の手段は用意されたインストールスクリプトを使用してインストールします。2つ目の手段はTiUPを使用してインストールします。

注25 <https://status.tidbcloud.com/>

▶ インストールスクリプトを使用してインストール

インストールスクリプトを使用してインストール

```
$ curl https://raw.githubusercontent.com/tidbcloud/tidbcloud-cli/main/install.sh | sh
Detected shell: bash
Shell profile: /home/tidb/.bash_profile
/home/tidb/.bash_profile has been modified to add ticloud to PATH
open a new terminal or source /home/tidb/.bash_profile to use it
Installed path: /home/tidb/.ticloud/bin/ticloud
=====
Have a try:      ticloud
=====
```

インストールスクリプトを実行したらシェルのプロファイルを読み込みするか、新しい端末を開いてください。インストールが成功するとticloudコマンドが使用できるようになります。

ticloudコマンドの確認

```
$ source ~/.bash_profile

$ which ticloud
~/ticloud/bin/ticloud
```

デフォルトでは実行ユーザーのホームディレクトリ内の.ticloudディレクトリにインストールされます。インストール先を変更したい場合は、環境変数TICLOUD_HOMEを指定してインストールスクリプトを実行してください。

TICLOUD_HOMEを指定してインストール

```
$ curl https://raw.githubusercontent.com/tidbcloud/tidbcloud-cli/main/install.sh | TICLOUD_HOME= /srv/data/ticloud sh
```

▶ TiUPを使用してインストール

TiUPを使用してインストールする場合は次のようにTiUPを実行します。

TiUPを使用してインストール

```
$ tiup install cloud

$ tiup cloud -h
Starting component cloud: ...
cloud is a CLI library for communicating with TiDB Cloud's API.

Usage:
  cloud [command]

Available Commands:
  branch      Manage branches in your serverless cluster
  cluster     Manage clusters for your project
```



```
completion  Generate the autocompletion script for the specified shell
config      Configure and manage your user profiles
connect     Connect to TiDB Cloud
help        Help about any command
import      Import data into TiDB Cloud cluster
project     Manage projects
update      Update the CLI to the latest version
```

Flags:

```
-D, --debug      Enable debug mode
-h, --help       help for cloud
--no-color       Disable color output
-P, --profile string Profile to use from your configuration file
```

Use "cloud [command] --help" for more information about a command.

.tiupディレクトリ内にticloudコマンドのバイナリーがインストールされ、tiup cloudコマンドを使用して実行できるようになります。

執筆時点ではインストールスクリプトを使用するとバージョン1.0.0-beta.2がインストールされます。TiUPを使用した場合はバージョン0.5.2がインストールされます。手動でバイナリーをダウンロードすることでもインストールできます。その場合はGitHubのリリースページ^{注26}からダウンロードしてください。

本節の手順はticloudコマンド(バージョン1.0.0-beta.2)を使用しています。TiUPを使用してインストールした場合はtiup cloudコマンドに置き換えて実行してください。

プロファイルを作成する

ticloudコマンドを使用するためには、プロファイルを作成する必要があります。プロファイルはticloud config createコマンドを使用して作成します。公開鍵、秘密鍵を使用した認証は後述する「APIキーによる認証を使用する」の項で説明しますので、現時点ではプロファイル名のみを指定してプロファイルを作成します。

プロファイルの作成

```
$ ticloud config create
Check the https://docs.pingcap.com/tidbcloud/api/ ...
> Profile Name must not contain periods
> Public Key (optional)
> Private Key (optional)

[ Submit ]
```

注26 <https://github.com/tidbcloud/tidbcloud-cli/releases>

「Profile Name must not contain periods」の部分にプロファイル名を入力します。プロファイル名にはピリオドを含めることはできません。入力したら「Submit」にカーソルを移動してEnterを押します。次の例はtidb-sampleというプロファイルを作成した場合です。ticloud config describeコマンドでプロファイルの詳細を確認できます。現時点では何も設定されていません。

プロファイルの確認

```
$ ticloud config list
Profile Name
tidb-sample (active)

$ ticloud config describe tidb-sample
{
  "private-key": "",
  "public-key": ""
}
```

なお、ticloud config create コマンドはデフォルトではインタラクティブモードで実行されますが、--profile-name オプションを指定することで非インタラクティブモードでプロファイルを作成できます。

非インタラクティブモードでプロファイルの作成

```
$ ticloud config create --profile-name tidb-sample
```

プロファイルを直接編集する

プロファイルの設定はticloud コマンドの実行ユーザーのホームディレクトリ(たとえば/home/tidb)内の.ticloud/config.tomlファイルに保存されます。ticloud config edit コマンドを実行するとエディターが立ち上がり、直接設定ファイルを編集できます。

設定ファイルの編集

```
$ ticloud config edit
```

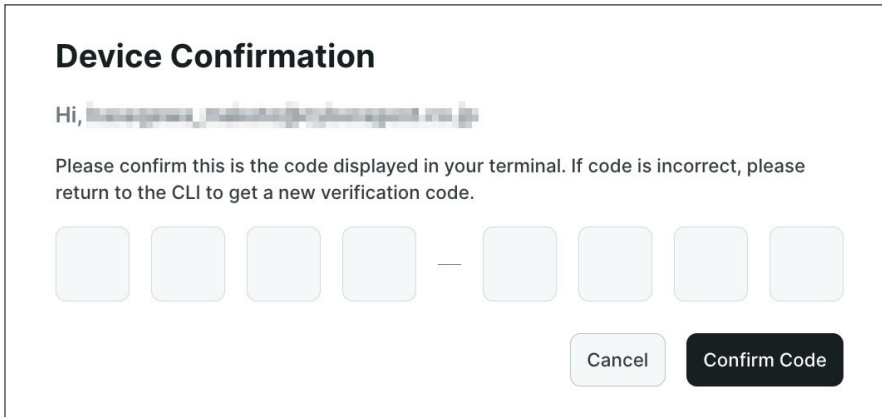
OAuthによる認証を使用する

OAuth による認証を使用する場合はticloud auth login コマンドを使用します。実行するとConfirmation Codeが表示されます。ブラウザが起動可能な状態であればブラウザが起動し、Device Confirmationの画面が表示されます(図5.33)。ブラウザが起動できない環境でticloud auth login コマンドを実行した場合は、別途ブラウザを起動して表示されているURLへアクセスしてください。

ticloud auth login コマンドはデータを安全に保存するために、内部でgo-keyring^{注27}を使用してい

注27 <https://github.com/zalando/go-keyring>

図5.33 TiDB CloudのDevice Confirmation画面



ます。go-keyringは特定のコマンドが別途必要となりますので、GitHubリポジトリのREADME内に記載されているDependencies^{注28}を参考にセットアップしてください。なお、keyringの仕組みを使用しない場合は`--insecure-storage`オプションを指定できますが、その場合はセキュリティ上のリスクが伴いますので注意してください。

OAuthによる認証

```
$ ticloud auth login
Attempting to automatically open the authorization page in your default browser.
If the browser does not open or you wish to use a different device to authorize ...

https://tidbcloud.com/oauth/device?user_code=xxxx-xxxx

Confirmation Code: xxxx-xxxx
```

ブラウザで「Confirm Code」をクリックすると認証が完了します。認証が完了するとプロフィール内の情報が更新されます。

OAuth設定後

```
$ ticloud config describe tidb-sample
{
  "private-key": "",
  "public-key": "",
  "token-expired-at": "2025-01-09T08:32:43.317316+09:00",
  "token-type": "Bearer"
}
```

`--insecure-storage`オプションを使用した場合はaccess-tokenが表示されます。

注28 <https://github.com/zalando/go-keyring?tab=readme-ov-file#dependencies>

--insecure-storageオプション利用時のOAuth設定後

```
$ ticloud config describe tidb-sample
{
  "access-token": "ticloud_oa_.....",
  "private-key": "",
  "public-key": "",
  "token-expired-at": "2025-01-09T08:32:43.317316+09:00",
  "token-type": "Bearer"
}
```

これでticloudコマンドによる操作が可能になりました。TiDB Cloud Serverlessのクラスター一覧を表示してみます。

TiDB Cloud Serverlessのクラスター一覧

```
$ ticloud serverless list
ID                               DisplayName State  Version Cloud Region ...
10302667634043741050 Cluster0  ACTIVE v7.1.3  aws    Tokyo  ...
```

TiDB Cloudからログアウトする場合はticloud auth logoutコマンドを使用します。--insecure-storageオプションを指定してログインしていた場合はプロファイルからaccess-tokenが削除されます。

ログアウト

```
$ ticloud auth logout
You have successfully logged out.
```

APIキーによる認証を使用する

OAuthによる認証以外にAPIキーを払い出して認証する方法もあります。APIキーはTiDB CloudのWebコンソールから払い出します。「Organization Settings」から「API Keys」の画面を表示し、右上にある「Create API Key」をクリックします。なお、APIキーの払い出しはOrganization Ownerの権限が必要です。また、払い出されたAPIキーを使用した場合はOrganization Ownerと同等の権限で各種操作が可能になります。

APIキーの払い出しにはDescriptionが必要になりますので、分かりやすい説明を入れておくことをおすすめします。APIキーを発行するとPublic Key(公開鍵)およびPrivate Key(秘密鍵)が払い出されます。Private Keyは一度しか表示されませんので、必ず安全な場所に保存してください。

払い出されたAPIキーを使用してプロファイルを作成します。ticloud config createコマンドを使用してプロファイルを作成し、Public KeyおよびPrivate Keyを入力します。このあとプロファイルの切り替え作業を説明するので、ここでは先ほど作成したプロファイル(tidb-sample)とは別のプロファイル(tidb-sample-apikey)として作成します。

APIキーによる認証

```
$ ticloud config create
Check the https://docs.pingcap.com/tidbcloud/api/v1beta ...
> tidb-sample-apikey
> xxxxxxxx
> .....

[ Submit ]
```

なお、ticloud config create コマンド時にプロファイル名と共に Public Key および Private Key を指定できます。

認証情報を指定してプロファイルを作成

```
$ ticloud config create --profile-name tidb-sample-apikey --public-key xxx --private-key xxx
```

新規プロファイルとして tidb-sample-apikey を作成しました。OAuth の時と同じように TiDB Cloud Serverless クラスタ一覧を表示してみます。

TiDB Cloud Serverlessのクラスタ一覧

```
$ ticloud serverless list
```

ID	DisplayName	State	Version	Cloud	Region	...
10302667634043741050	Cluster0	ACTIVE	v7.1.3	aws	Tokyo (ap-northeast-1)	...

クラスタ一覧が取得できていれば認証に成功しています。

API キーを使用した認証は OAuth に比べてセキュリティが低いため、API キーを使用する場合はプロファイルの設定ファイル(ホームディレクトリ内の .ticloud/config.toml)のアクセス権限に注意してください。設定ファイル内に平文で Public Key および Private Key が保存されています。

APIキーの表示

```
$ ticloud config describe tidb-sample-apikey
{
  "private-key": ".....",
  "public-key": "xxxxxxx"
}

$ cat ~/.ticloud/config.toml
current-profile = 'tidb-sample-apikey'

[tidb-sample-apikey]
private-key = '.....'
public-key = 'xxxxxxx'
```

プロファイルを切り替える

各種 ticloud コマンドを実行する際に使用するプロファイルは `--profile` オプションで指定します。

プロファイルの切り替え

```
$ ticloud serverless list --profile tidb-sample
$ ticloud serverless list --profile tidb-sample-apikey
```

`--profile` オプションを指定しない場合は、アクティブなプロファイルが使用されます。現在アクティブなプロファイルを確認する場合は `ticloud config list` コマンドを実行します。(active) が表示されているプロファイルがアクティブなプロファイルです。

アクティブなプロファイルの確認

```
$ ticloud config list
Profile Name
tidb-sample      (active)
tidb-sample-apikey
```

アクティブなプロファイルを切り替える場合は `ticloud config use` コマンドを使用します。

アクティブなプロファイルの切り替え

```
$ ticloud config use tidb-sample-apikey
Current profile has been changed to tidb-sample-apikey

$ ticloud config list
Profile Name
tidb-sample
tidb-sample-apikey (active)
```

プロファイルを更新する

プロファイルの更新は `ticloud config set` コマンドを使用します。`--profile` オプションを指定しない場合はアクティブなプロファイルの内容が更新されます。

アクティブなプロファイルの更新

```
$ ticloud config set public-key xxxxxxxx
```

特定のプロファイルの更新

```
$ ticloud config set public-key xxxxxxxx --profile tidb-sample-apikey
```

更新可能な項目は次の通りです。

- `public-key`
- `private-key`

- api-url
- serverless-endpoint
- iam-endpoint
- oauth-endpoint
- oauth-client-id
- oauth-client-secret
- telemetry-enabled

プロファイルを削除する

プロファイルを削除する場合は `ticloud config delete` コマンドを使用します。

プロファイルの削除

```
$ ticloud config delete tidb-sample-apikey
? Please type yes to confirm:
```

確認を求められますので `yes` を入力します。次のメッセージが表示されてプロファイルが削除されます。

```
Profile tidb-sample-apikey deleted successfully
```

TiDB Cloud Serverlessのクラスターを操作する

`ticloud serverless` コマンドを使用して TiDB Cloud Serverless のクラスターを操作できます。

作成

TiDB Cloud Serverless のクラスターを作成する場合は次のように実行します。

TiDB Cloud Serverlessのクラスター作成

```
$ ticloud serverless create
```

何もオプションを指定せずに実行すると、インタラクティブモードでクラスターの作成に必要な情報を入力する必要があります。

インタラクティブモードでのTiDB Cloud Serverlessのクラスター作成

```
$ ticloud serverless create
Choose the cloud region:
> [x] Tokyo (ap-northeast-1)
> from-cli-cluster

[ Submit ]
```

```
? Set spending limit monthly in USD cents (Example: 10, default is 0)? 0
••• Waiting for cluster to be ready
Cluster 10275900787964872131 is ready.
```

必要な情報はリージョン、クラスターの名前です。実行時にオプションで指定するとインタラクティブモードでの入力を省略できます。--spending-limit-monthly オプションで月額限度額も同時に設定できます(デフォルトでは0になります)。

月額限度額を省略、または0を指定した場合はFree Clusterが作成されます。

0より大きい値を指定した場合はScalable Clusterが作成されます。請求情報を設定していない場合は作成時にエラーになりますので注意してください。

オプション指定でのTiDB Cloud Serverlessクラスターの作成

```
$ ticloud serverless create --display-name from-cli-cluster --region aws-ap-northeast-1
••• Waiting for cluster to be ready
Cluster 10275900787964872131 is ready.
```

無事に作成されるとticloud serverless listコマンドで確認できます。

TiDB Cloud Serverlessクラスターの一覧表示

```
$ ticloud serverless list
```

ID	DisplayName	State	Version	Cloud	Region	CreateTime
10275900787964872131	from-cli-cluster	ACTIVE	v7.1.3	aws	Tokyo (ap-northeast-1)	2024-12-10T00:23:07Z
10302667634043741050	Cluster0	ACTIVE	v7.1.3	aws	Tokyo (ap-northeast-1)	

削除

TiDB Cloud Serverlessのクラスターを削除する場合は次のように実行します。オプションを指定せずに実行すると、インタラクティブモードでクラスターを選択する必要があります。

TiDB Cloud Serverlessクラスターの削除

```
$ ticloud serverless delete
Choose the cluster:
> [x] from-cli-cluster(10275900787964872131)
? Please type yes to confirm: yes
cluster 10275900787964872131 deleted
```

--cluster-id オプションでクラスターのIDを指定することでインタラクティブモードでの入力を省略できます。クラスターのIDはticloud serverless listで確認してください。

オプション指定でのTiDB Cloud Serverlessのクラスターの削除

```
$ ticloud serverless delete --cluster-id 10275900787964872131
? Please type yes to confirm: yes
cluster 10275900787964872131 deleted
```

削除時の確認は--force オプションを指定することで省略できます。

確認を省略してのTiDB Cloud Serverlessのクラスターへの削除

```
$ ticloud serverless delete --cluster-id 10275900787964872131 --force
cluster 10275900787964872131 deleted
```

▶ クラスターへの接続

ticloud serverless shellコマンドを使用してTiDB Cloud Serverlessのクラスターに接続できます。ユーザー名を自動で取得してくれるので非常に便利です。オプションでクラスターIDを指定しない場合はインタラクティブモード(対話モード)でクラスターを選択する必要があります。パスワードは接続時に入力するか、オプションで指定可能です。

TiDB Cloud Serverlessのクラスターへの接続

```
$ ticloud serverless shell
Choose the cluster:
> [x] Cluster0(10302667634043741050)
? Use the default user? Yes
Current user: xxxxxxxxxxxx.root
Enter password:
Connected with driver mysql (5.7.28-TiDB-v7.1.3-serverless)
Type "help" for help.

xxxxxxxxxxxxx.root@Cluster0=>SELECT VERSION();
      VERSION()
-----
5.7.28-TiDB-v7.1.3-serverless
(1 row)
```

ユーザー名やクラスターIDなどをオプションで指定して非インタラクティブモードで接続できます。

オプション指定でのTiDB Cloud Serverlessのクラスターへの接続

```
$ ticloud serverless shell --cluster-id 10302667634043741050 --user xxxxxxxxxxxx.root --password ㊄
xxxxxxxxxxxxx
Connected with driver mysql (5.7.28-TiDB-v7.1.3-serverless)
Type "help" for help.

xxxxxxxxxxxxx.root@Cluster0=>
```

▶ ブランチの操作

ブランチはticloud serverless branchコマンドを使用して操作します。ブランチの一覧表示はticloud serverless branch listコマンドで行います。

ブランチの一覧表示

```
$ ticloud serverless branch list
Choose the cluster:
> [x] Cluster0(10302667634043741050)
```

ID	DisplayName	ParentID	ParentDisplayName	State	CreateTime
bran-cbmdmg4r7vbbxklbnj2pof6mkm	test branch	10302667634043741050	Cluster0	ACTIVE	2024-12-09T07:48:11Z
bran-ffx4fxwlgzedbjnk6lmm6hkvq	test branch 2	bran-cbmdmg4r7vbbxklbnj2pof6mkm	test branch	ACTIVE	2024-12-09T08:33:25Z
bran-tjnj23u4c5agflrua5l2egc4z4	test branch 3	bran-ffx4fxwlgzedbjnk6lmm6hkvq	test branch 2	ACTIVE	2024-12-09T08:36:42Z

クラスターIDを指定しない場合はインタラクティブモードでクラスターを選択する必要があります。指定する場合は`--cluster-id`オプションを使用します。

オプション指定でのブランチの一覧表示

```
$ ticloud serverless branch list --cluster-id 10275900787964872131
ID                               DisplayName  ParentID          ParentDisplayName  State  CreateTime
bran-6aki6xytmzapzaxlunkw4yyui  branch2     10275900787964872131  from-cli-cluster  ACTIVE 2024-12-10T01:13:13Z
bran-ou3osgvjobc5pjg3rfcmntdusi  branch1     10275900787964872131  from-cli-cluster  ACTIVE 2024-12-10T01:11:39Z
```

ブランチを作成する場合は`ticloud serverless branch create`コマンドを使用します。オプションを指定しない場合はインタラクティブモードでブランチの作成に必要な情報を入力する必要があります。入力後に「Submit」を選択して を押します。

ブランチの作成

```
$ ticloud serverless branch create
Choose the cluster:
> [x] Cluster0(10302667634043741050)
Choose the parent:
> [x] Cluster0(main)
> Display Name
> Parent Timestamp (optional, e.g., 2024-12-10T10:18:54+09:00)

[ Submit ]
•●• Waiting for branch to be ready
Branch bran-ihpifxvghvh3zc7g3lb4wolkoe is ready.
```

作成に必要な情報は各種オプションで指定できます。`--parent-id`オプションは元となるクラスターIDまたはブランチIDを指定します。クラスターIDを指定した場合はクラスターの現在のデータを元にブランチが作成されます。ブランチIDを指定した場合はブランチのデータから新規のブランチが作成されます。ブランチIDは`ticloud serverless branch list`コマンドで確認できます。`--parent-id`オプションを省略した場合はクラスターIDを指定した時と同じ挙動になります。`--parent-timestamp`オプションを指定すると、指定したタイムスタンプのデータを元にブランチが作成されます。タイムスタンプはRFC 3339形式(例:2024-12-01T00:00:00Z)で指定します。指定できるタイムスタンプはFree Clusterの場合は24時間以内、Scalable Clusterの場合は14日以内になります。

現在のクラスターデータからブランチ作成

```
$ ticloud serverless branch create --cluster-id 10302667634043741050 \
--display-name newbranch
```

特定のブランチからブランチを作成

```
$ ticloud serverless branch create --cluster-id 10302667634043741050 \
--display-name newbranch2 --parent-id bran-ihpifxvghvh3zc7g3lb4wolcoe
```

特定の時刻のデータからブランチを作成

```
$ ticloud serverless branch create --cluster-id 10302667634043741050 \
--display-name newbranch2 --parent-timestamp 2024-12-14T00:00:00Z
```

ブランチを削除する場合はticloud serverless branch deleteコマンドを使用します。オプションを指定しない場合はインタラクティブモードでブランチを選択する必要があります。

ブランチの削除

```
$ ticloud serverless branch delete
Choose the cluster:
> [x] from-cli-cluster(10275900787964872131)
Choose the branch:
> [x] branch2(bran-6aki6xytmzapzaxlunkkw4yyui)
? Please type yes to confirm: yes
branch bran-6aki6xytmzapzaxlunkkw4yyui deleted
```

--cluster-id オプションと --branch-id オプションを使用することで非インタラクティブモードになり入力を省略できます。削除の確認は--force オプションを指定することで省略できます。

オプション指定でのブランチの削除

```
$ ticloud serverless branch delete --cluster-id 10275900787964872131 --branch-id bran-6aki6xytmzapzaxlunkkw4yyui --force
branch bran-6aki6xytmzapzaxlunkkw4yyui deleted
```

指定したブランチに接続するにはticloud serverless branch shellを使用します。こちらも同様にオプションを指定しない場合はインタラクティブモードでブランチを選択する必要があります。オプションを使用する場合は--cluster-id オプションや --branch-id オプションなどを使用してください。

ブランチへの接続

```
$ ticloud serverless branch shell --cluster-id 10275900787964872131 --branch-id bran-ou3osgvjbc5 pjg3rfcmntdusi --user xxxxxxxxxxxx.root --password xxxxxxxxxxxx
Connected with driver mysql (5.7.28-TiDB-v7.1.3-serverless)
Type "help" for help.

xxxxxxxxxxxxx.root@branch1=>
```

SQLユーザーの操作

TiDB Cloud Serverlessに接続する際に使用するユーザー(SQLユーザー)の作成やパスワード変更、削除の操作ができます。なお、SQLユーザーはTiDB Cloudの組織のユーザーとは別の概念です。TiDB Cloud Serverlessのクラスター内のデータベースへアクセスするためのユーザーとなります。

SQLユーザーの一覧表示は`ticloud serverless sql-user list`コマンドで行います。`--cluster-id`オプションを指定しない場合はインタラクティブモードになり、クラスターを選択する必要があります。

SQLユーザーの一覧表示

```
$ ticloud serverless sql-user list --cluster-id 10275900787964872131
UserName          Role
xxxxxxxxxxxxxxx.root  role_admin
```

SQLユーザーの作成は`ticloud serverless sql-user create`コマンドを使用します。オプションを指定しなくても実行できますが、インタラクティブモードになりますので適宜入力が必要になります。`--role`オプションで指定できるロールの種類は`role_admin`、`role_readwrite`、`role_readonly`の3つになります。また、`--user`オプションで指定したユーザー名は最終的にプリフィックスが付与されます。

SQLユーザーの作成

```
$ ticloud serverless sql-user create --cluster-id 10275900787964872131 \
--user newuser --password newpassword --role role_readwrite
SQL user xxxxxxxxxxxxxxx.newuser is created
```

パスワードの変更やロールの変更は`ticloud serverless sql-user update`コマンドを使用します。

SQLユーザーのパスワード変更

```
$ ticloud serverless sql-user update --cluster-id 10275900787964872131 \
--user xxxxxxxxxxxxxxx.newuser --password newpassword2
```

SQLユーザーのロール変更

```
$ ticloud serverless sql-user update --cluster-id 10275900787964872131 \
--user xxxxxxxxxxxxxxx.newuser --role role_readonly
```

SQLユーザーの削除は`ticloud serverless sql-user delete`コマンドを使用します。

SQLユーザーの削除

```
$ ticloud serverless sql-user delete --cluster-id 10275900787964872131 --user xxxxxxxxxxxxxxx.newuser
? Please type yes to confirm:
```

確認を求められますので`yes`を入力して を押します。次のメッセージが表示されてSQLユーザーが削除されます。

```
SQL user xxxxxxxxxxxxxxx.newuser is deleted
```

▶ 月額限度額の設定

TiDB Cloud Serverlessの月額限度額を設定する場合は`ticloud serverless spending-limit`コマンドを使用します。`--cluster-id`オプションでクラスターIDを指定します。`--monthly`オプションで月額の予算をUSセントで指定します。USドルではないので注意してください。デフォルトは0です。

月額限度額の設定

```
$ ticloud serverless spending-limit --cluster-id 10275900787964872131 \  
--monthly 1000  
set spending limit to 1000 cents success
```

設定されている月額限度額はticloud serverless describeコマンドで確認できます。

月額限度額の確認

```
$ ticloud serverless describe --cluster-id 10275900787964872131 | \  
grep -A 2 spendingLimit  
"spendingLimit": {  
  "monthly": 1000  
},
```

5

GitHub ActionsでTiDB Cloud CLIを使用する

GitHub Actions のワークフロー内でTiDB Cloud CLIをセットアップする必要がある場合はsetup-tidbcloud-cli^{注29}を使用すると便利です。setup-tidbcloud-cliを使用する場合は、GitHubのシークレットにAPIキーの情報を保存してください。ワークフローの定義は次のようになります。

GitHub Actionsでの使用

```
uses: tidbcloud/setup-tidbcloud-cli@v0.2.0  
with:  
  api_public_key: ${ secrets.TIDB_CLOUD_API_PUBLIC_KEY }  
  api_private_key: ${ secrets.TIDB_CLOUD_API_PRIVATE_KEY }
```

ticloudのバージョンを指定しない場合は最新のバージョンが使用されます。バージョンを指定する場合はversionを指定してください。

GitHub Actionsでのバージョン指定での使用

```
uses: tidbcloud/setup-tidbcloud-cli@v0.2.0  
with:  
  api_public_key: ${ secrets.TIDB_CLOUD_API_PUBLIC_KEY }  
  api_private_key: ${ secrets.TIDB_CLOUD_API_PRIVATE_KEY }  
  version: 0.5.2
```

注29 <https://github.com/tidbcloud/setup-tidbcloud-cli>

5-6

まとめ

TiDB CloudはTiDBをフルマネージドの環境で利用できるサービスです。TiDBに関わる運用コストを軽減し、アプリケーションの開発に集中できますので、運用コストをなるべく減らしたいニーズがある場合は有力な選択肢になります。

TiDB Cloud ServerlessとTiDB Cloud Dedicatedはそれぞれ特徴の違いがあります。TiDB Cloud Serverlessは先進的な機能が入り入れられている一方で、制約の多さやカスタマイズできる部分が少ないなどの側面を持っています。TiDB Cloud Dedicatedはカスタマイズ性が高く、アプリケーションに合わせて柔軟に運用できる一方で、先進的なブランチ機能やベクトル検索機能が使用できないなどの側面を持っています。それぞれの特徴をしっかりと理解し、適切なサービスを選択することが大切です。

なお、ベクトル検索機能はTiDBのバージョン8.4.0で実験的に実装されました。将来的にTiDB Cloud DedicatedにおいてTiDBのバージョン8.4.0以降がサポートされた場合、TiDB Cloud Dedicatedでもベクトル検索機能を使用できます。執筆時点ではTiDB Cloud Dedicatedの新規クラスター作成時に使用されるバージョンは8.1.2です。

それぞれの特徴の比較を表5.4にまとめましたので、判断する時の参考にしてください。

表5.4 TiDB Cloudのサービス比較

特徴	TiDB Cloud Serverless	TiDB Cloud Dedicated
提供形態	リソース共有型	リソース専有型
料金形態	リクエストユニットによる従量課金	インスタンスタイプ、ストレージサイズなどによる従量課金
無料利用枠	有り	無し（検証目的でクレジットの利用が可能な仕組みあり）
カスタマイズ性	低い	高い
オートスケーリング	有り	無し
対応クラウドプロバイダー	AWS	AWS、Google Cloud
ベクトル検索機能	有り（ベータ版）	執筆時点では無し
ブランチ機能	有り（ベータ版）	無し
ビルトインアラート機能	無し	有り
サードパーティメトリクス連携	無し	有り

第 6 章

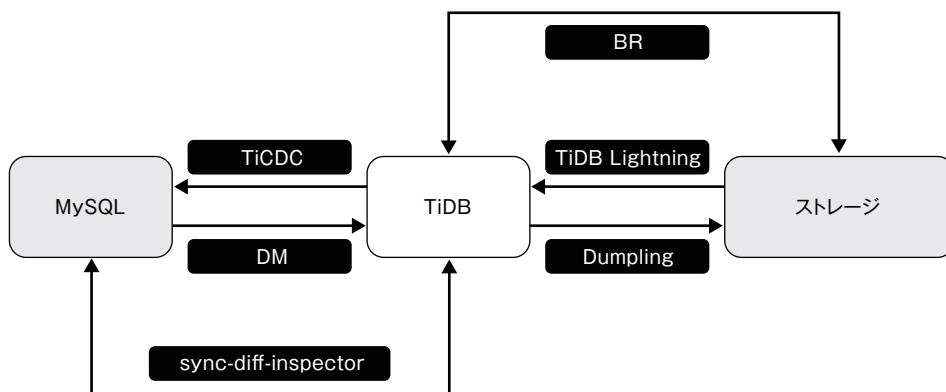
TiDBの データ運用ツール

バックアップとリストア、同期、移行

本章ではTiDBを運用していくにあたり、データの管理に役立つツールを紹介します。ツールを使用することでTiDBのデータのバックアップやリストア、データの移行を効率的に行うことができます。また、TiDB外へのデータ同期についても紹介します。

本章で紹介するツールの関連図は図6.1の通りです。

図6.1 データ運用ツールの関連図



BRはストレージクラスターのTiKVのデータをバックアップおよびリストアするツールです。DumplingはTiDBからデータをエクスポートするツールで、MySQLにも対応しています。TiDB LightningはDumplingでエクスポートされたデータをインポートするツールです。TiCDCはTiDB外部へデータを同期するツールで、DMはMySQLおよびMySQL互換のデータベース（MariaDBやAurora MySQLなど）からデータを移行するツールです。sync-diff-inspectorはTiDBとTiDB、もしくはMySQLとTiDBのデータを比較するツールです。

6-1

バックアップとリストア

TiDBに限らず、データベースを運用していく上で大切なのはバックアップとリストアです。TiDBではバックアップとリストアを行うツールとして、次のツールが用意されています。

- BR
- Dumpling
- TiDB Lightning

本節ではこれらのツールについて使い方を説明していきます。バックアップ先のストレージとしてAmazon S3のバケットtidb-book-sampleを使用します。バケットの作成の仕方はAWSのドキュメント

のCreating a bucket^{注1}を参照してください。また、パケットにアクセスするための認証情報はAWSのドキュメントのManaging access keys for IAM users^{注2}を参照して用意してください。AWSへの認証情報は環境変数として設定して各ツールが使用できるようにしておきます。

AWSの認証情報の設定

```
$ export AWS_ACCESS_KEY_ID=xxxxxxxxxxxxx
$ export AWS_SECRET_ACCESS_KEY=xxxxxxxxxxxxx
```

環境変数を使用しない場合、~/aws/credentials ファイルまたは環境変数AWS_SHARED_CREDENTIALS_FILEに設定されたファイルを使用できます。各種ツールはこのファイルを参照して認証情報を取得します。

BRを使用する

BRはTiDBのバックアップとリストアを行うためのツールです。BRはストレージクラスターのTiKVと通信することで高速にバックアップおよびリストアを実行します。バックアップの保存先はローカルファイルシステムに加えて、Amazon S3やGoogle Cloud Storageなどのオブジェクトストレージに保存できます。

BRはバックアップの種類としてスナップショットバックアップとログバックアップをサポートしています。

BRはTiKVサーバーと通信する必要があるため、TiDB Cloud Serverlessでは使用できません。TiDB Cloud ServerlessにおけるバックアップはTiDB CloudのWeb コンソールから行う(5-3節の「データをバックアップおよび復元する」参照)か、後述のDumplingを使用してください。

BR実行時のパフォーマンスへの影響

BRによるバックアップやリストアは稼働しているTiDBのパフォーマンスに対して影響を与えます。TiDBのリソースが十分にある場合はパフォーマンスへの影響は限りなく低くなりますが、リソースが不足している場合はパフォーマンスに影響を与える可能性がありますのでBR実行時には注意してください。パフォーマンスに影響を与えないようにするためには、バックアップやリストアに使用するリソース量、スレッド数などを調整する必要があります。詳細はドキュメントのSnapshot Backup and Restore Guide^{注3}を参照してください。

BRによるバックアップの仕組み

BRはバックアップ時に、PDサーバーからバックアップ取得時刻のタイムスタンプと共にTiKVサー

注1 <https://docs.aws.amazon.com/AmazonS3/latest/userguide/create-bucket-overview.html>

注2 https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_access-keys.html

注3 <https://docs.pingcap.com/tidb/stable/br-snapshot-guide#performance-and-impact>

バーの情報を取得します。その後バックアップ対象(単一テーブルもしくはデータベース全体)のキーの範囲を取得し、TiKVサーバーに対してバックアップ要求を送信します。TiKVはバックアップ要求を受け取ると、バックアップ対象のキー範囲からリージョンを特定してデータをSST(Sorted String Table)と呼ばれるファイルに保存します。保存が完了するとTiKVサーバーはBRにメタデータを返します。BRはメタデータを収集してbackupmetaファイルとして保存します。

▶ BRのインストール

BRはTiUPを使用してインストールします。

TiUPによるBRインストール

```
$ tiup install br
```

BRを使用する場合はtiup brコマンドを実行します。

TiUPからBRを使用する

```
$ tiup br --version
Starting component br: ...
Release Version: v8.5.0
Git Commit Hash: d13e52ed6e22cc5789bed7c64c861578cd2ed55b
Git Branch: HEAD
Go Version: go1.23.3
UTC Build Time: 2024-12-18 02:28:02
Race Enabled: false
```

インストール可能なBRのバージョンはtiup list brコマンドで確認できます。

BRのバージョンリスト確認

```
$ tiup list br
Available versions for br:
Version                Installed  Release           ...
-----
nightly -> v9.0.0-alpha-nightly      2024-12-28T07:26:50Z ...
...
(省略)
...
v8.5.0                  YES       2024-12-19T06:05:58Z ...
v9.0.0-alpha-nightly    2024-12-28T07:26:50Z ...
```

▶ スナップショットバックアップ

実際にスナップショットバックアップを取得してみましょう。TiDB PlaygroundでTiDBを起動し、デモ用のデータを投入します。TiDB Playgroundの使い方は3-2節を参照してください。デモ用データの投入の仕方は3-10節を参照してください。

TiDB Playgroundを起動

```
$ tiup playground
```

別端末からデモ用データの投入

```
$ tiup demo bookshop prepare
```

デモ用のデータの投入により bookshop データベースが作成されました。まず最初に bookshop データベース内にある books テーブルを対象にスナップショットを取得します。実行時には --pd オプションで PD サーバーを指定します。バックアップの保存先は --storage オプションで指定します。--s3.region オプションでバケットのリージョンを指定します。--send-credentials-to-tikv オプションは、各 TiKV ノードがオブジェクトストレージにアクセスするための認証情報を TiKV ノードに送信するためのオプションです。

BRによるテーブルスナップショット取得

```
$ tiup br backup table --db bookshop --table books \
  --pd 127.0.0.1:2379 --send-credentials-to-tikv \
  --storage s3://tidb-book-sample/backup/table/books --s3.region ap-northeast-1
Starting component br: ...
Detail BR log in /tmp/br.log.2024-12-13T20.59.26Z
Table Backup <-----> 100.00%
Checksum <-----> 100.00%
[2024/12/13 20:59:32.697 +00:00] [INFO] ... ["Table Backup success summary"] ...
```

バケットに格納された books テーブルのスナップショット内容を確認します。

スナップショット内容の確認

```
$ aws s3 ls --recursive s3://tidb-book-sample/backup/table/books/
2024-12-14 05:59:32      756486 backup/table/books/1/14_71..._write.sst
2024-12-14 05:59:28         78 backup/table/books/backup.lock
2024-12-14 05:59:33        368 backup/table/books/backupmeta
2024-12-14 05:59:32        236 backup/table/books/backupmeta.datafile.000000001
2024-12-14 05:59:33       3789 backup/table/books/backupmeta.schema.000000002
```

指定したバケット内に SST ファイルと backupmeta ファイルが保存されていることが確認できます。

次に bookshop データベース全体をスナップショットバックアップします。実行内容は先ほどのテーブルとほぼ同じですが、tiup br backup db コマンドを使用します。

BRによるデータベーススナップショット取得

```
$ tiup br backup db --db bookshop \
  --pd 127.0.0.1:2379 --send-credentials-to-tikv \
  --storage s3://tidb-book-sample/backup/database/bookshop --s3.region ap-northeast-1
Starting component br: ...
Detail BR log in /tmp/br.log.2024-12-13T21.10.12Z
Database Backup <-----> 100.00%
Checksum <-----> 100.00%
[2024/12/13 21:10:22.308 +00:00] ... ["Database Backup success summary"] ...
```

バケットに格納された bookshop データベース全体のスナップショット内容を確認します。テーブル単体のスナップショットバックアップとは違い、ファイル数が増えていることが確認できます。

データベーススナップショット内容の確認

```
$ aws s3 ls --recursive s3://tidb-book-sample/backup/database/bookshop
2024-12-14 06:22:47 9328584 backup/database/bookshop/1/143_67_2c...sst
2024-12-14 06:22:46 3749593 backup/database/bookshop/1/143_67_32...sst
2024-12-14 06:22:45 5776513 backup/database/bookshop/1/145_68_90...sst
2024-12-14 06:22:46 5873113 backup/database/bookshop/1/145_68_a3...sst
2024-12-14 06:22:46 271594 backup/database/bookshop/1/147_69_45...sst
2024-12-14 06:22:47 528353 backup/database/bookshop/1/149_70_f5...sst
2024-12-14 06:22:45 756486 backup/database/bookshop/1/14_71_5e...sst
2024-12-14 06:22:44 256763 backup/database/bookshop/1/151_71_42...sst
2024-12-14 06:22:47 112148 backup/database/bookshop/1/151_71_53...sst
2024-12-14 06:22:44 178775 backup/database/bookshop/1/151_71_66...sst
2024-12-14 06:22:40      78 backup/database/bookshop/backup.lock
2024-12-14 06:22:49    370 backup/database/bookshop/backupmeta
2024-12-14 06:22:48   2343 backup/database/bookshop/backupmeta.datafile.000000001
2024-12-14 06:22:49  19277 backup/database/bookshop/backupmeta.schema.000000002
```

TiDB 全体のスナップショットバックアップを取得する場合は `tiup br backup full` コマンドを使用します。

BRによるTiDB全体のスナップショット取得

```
$ tiup br backup full --pd 127.0.0.1:2379 \
--send-credentials-to-tikv \
--storage s3://tidb-book-sample/backup/cluster --s3.region ap-northeast-1
Starting component br: ...
Detail BR log in /tmp/br.log.2024-12-13T21.59.43Z
Full Backup <-----> 100.00%
Checksum <-----> 100.00%
[2024/12/13 21:59:55.789 +00:00] [INFO] ... ["Full Backup success summary"] ...
```

特定のデータベースやテーブルのみをバックアップ対象にしたい場合は、TiDB 全体のスナップショットを取得する際に `--filter` オプションを使用します。`--filter` オプションは複数指定できます。たとえば bookshop データベース内全部のテーブルと、test データベース内の sample テーブルを指定する場合は次のように指定します。

TiDB 全体のスナップショット取得時に特定のテーブルのみ選択

```
$ tiup br backup full --filter "bookshop.*" --filter "test.sample" ...
```

BR はデフォルトでは統計情報を保存しません。統計情報も同時にバックアップする場合は `--ignore-stats` オプションで `false` を指定します。

SSTファイル名について

column

SST ファイル名は次のように決定されます。

```
{regionID}_{regionEpoch}_{keyHash}_{timestamp}_{cf}.sst
```

Google Cloud Storage または Azure Blob Storage に保存される場合は storeID が追加されます。

```
{storeID}_{regionID}_{regionEpoch}_{keyHash}_{timestamp}_{cf}.sst
```

各フィールドの説明は表Aの通りです。

表A SSTファイル名のフィールド

フィールド名	説明
storeID	TiKV ノードID
regionID	リージョンID
regionEpoch	リージョンのバージョン番号
keyHash	リージョンの開始キーのハッシュ (sha256) 値
timestamp	TiKVによって生成されたSSTファイルのUnixタイムスタンプ
cf	RocksDBのカラムファミリー

統計情報もバックアップに含める

```
$ tiup br backup full --pd 127.0.0.1:2379 \
--ignore-stats false \
...
```

ログバックアップ

ログバックアップはTiDBに対して更新されたデータをログファイルとして指定された場所に保存します。スナップショットバックアップとは違い、一度設定するとタスクとしてバックグラウンドで実行され続け、止めるまで定期的に取得され続けます。ログバックアップとスナップショットバックアップを使用することでポイントインタイムリストアが可能になります。

ログバックアップはtiup br log コマンドで設定します。--task-name オプションでタスク名を指定します。タスク名は一意である必要があります。--pd オプションでPDサーバーを指定します。--storage オプションでバックアップの保存先を指定します。--s3.region オプションでバケットのリージョンを指定します。

ログバックアップ設定

```
$ tiup br log start --pd 127.0.0.1:2379 \
--task-name sample-task \
--storage s3://tidb-book-sample/log_backup/ --s3.region ap-northeast-1
Starting component br: ...
Detail BR log in /tmp/br.log.2024-12-15T05.10.26Z
```

```
[2024/12/15 05:10:27.357 +00:00] [INFO] ... ["log start"] ...  
[2024/12/15 05:10:27.361 +00:00] [INFO] ... ["log start success summary"] ...
```

ログバックアップの設定を確認するには `tiup br log status` コマンドを使用します。

```
ログバックアップ設定確認  
$ tiup br log status --pd 127.0.0.1:2379  
Starting component br: ...  
Detail BR log in /tmp/br.log.2024-12-15T05.12.43Z  
● Total 1 Tasks.  
> #1 <  
      name: sample-task  
      status: ● NORMAL  
      start: 2024-12-15 05:10:27.045 +0000  
      end: 2090-11-18 14:07:45.624 +0000  
      storage: s3://tidb-book-sample/log_backup  
      speed(est.): 0.00 ops/s  
checkpoint[global]: 2024-12-15 05:10:27.045 +0000; gap=2m18s
```

設定したバケット内にログバックアップが保存されていることを確認します。

```
ログバックアップ内容確認  
$ aws s3 ls --recursive s3://tidb-book-sample/log_backup/  
2024-12-15 14:10:28      78 log_backup/backup.lock  
2024-12-15 14:10:28      36 log_backup/backupmeta  
2024-12-15 14:12:54    550 log_backup/v1/20241215/05/1/...-2f38f820e08b.log  
2024-12-15 14:15:19    486 log_backup/v1/20241215/05/1/...-ad6153cacb4d.log  
2024-12-15 14:12:54    411 log_backup/v1/backupmeta/...-b5280e6ac44d.meta  
2024-12-15 14:15:19    385 log_backup/v1/backupmeta/...-a93db69fc0bc.meta  
2024-12-15 14:15:24       8 log_backup/v1/global_checkpoint/1.ts
```

ログバックアップの定期実行を一時的に停止する場合は `tiup br log pause` コマンドを使用します。

```
ログバックアップ一時停止  
$ tiup br log pause --pd 127.0.0.1:2379 --task-name sample-task  
Starting component br: ...  
Detail BR log in /tmp/br.log.2024-12-15T05.35.35Z  
[2024/12/15 05:35:35.406 +00:00] [INFO] ... ["log pause"] ...  
[2024/12/15 05:35:35.410 +00:00] [INFO] ... ["log pause success summary"] ...
```

ログバックアップのステータスを確認すると PAUSE になっています。

```
一時停止中のログバックアップタスク  
$ tiup br log status --pd 127.0.0.1:2379  
Starting component br: ...  
Detail BR log in /tmp/br.log.2024-12-15T05.35.47Z  
● Total 1 Tasks.  
> #1 <  
      name: sample-task
```

```

status: ● PAUSE
start: 2024-12-15 05:10:27.045 +0000
end: 2090-11-18 14:07:45.624 +0000
storage: s3://tidb-book-sample/log_backup
speed(est.): 0.00 ops/s
checkpoint[global]: 2024-12-15 05:34:38.844 +0000; gap=1m10s

```

再開する場合は `tiup br log resume` コマンドを使用します。

ログバックアップ再開

```

$ tiup br log resume --pd 127.0.0.1:2379 --task-name sample-task
Starting component br: ...
Detail BR log in /tmp/br.log.2024-12-15T05.38.51Z
[2024/12/15 05:38:51.524 +00:00] [INFO] ... ["log resume"] ...
[2024/12/15 05:38:51.526 +00:00] [INFO] ... ["log resume success summary"] ...

```

再びステータスが `NORMAL` になっていることを確認します。

再開後のログバックアップタスク

```

$ tiup br log status --pd 127.0.0.1:2379
Starting component br: ...
Detail BR log in /tmp/br.log.2024-12-15T05.39.08Z
● Total 1 Tasks.
> #1 <
    name: sample-task
    status: ● NORMAL
    start: 2024-12-15 05:10:27.045 +0000
    end: 2090-11-18 14:07:45.624 +0000
    storage: s3://tidb-book-sample/log_backup
    speed(est.): 0.84 ops/s
    checkpoint[global]: 2024-12-15 05:34:38.844 +0000; gap=4m32s

```

ログバックアップを停止する場合は `tiup br log stop` コマンドを使用します。

ログバックアップ停止

```

$ tiup br log stop --pd 127.0.0.1:2379 --task-name sample-task
Starting component br: ...
Detail BR log in /tmp/br.log.2024-12-15T05.44.42Z
[2024/12/15 05:44:42.840 +00:00] [INFO] ... ["log stop"] ...
[2024/12/15 05:44:42.843 +00:00] [INFO] ... ["log stop success summary"] ...

```

ステータスには、タスクが表示されず「No Task Yet.」が表示されることを確認します。

停止後の確認

```

$ tiup br log status --pd 127.0.0.1:2379
Starting component br: ...
Detail BR log in /tmp/br.log.2024-12-15T05.44.56Z
○ No Task Yet.

```

`tiup br log stop` コマンドはログバックアップのタスク定義を削除しますので、タスク定義を残し

たままログバックアップを停止したい場合は `tiup br log pause` を使用してください。

過去に取得したログバックアップを削除する場合は `tiup br log truncate` コマンドを実行します。スナップショットバックアップの保持期間よりも古いログバックアップのみ削除することをおすすめします。ログバックアップはポイントインタイムリストアに必要ですが、そのためにはスナップショットバックアップも必要になります。ある特定の日時のスナップショットバックアップを削除する場合、その日時より前のログバックアップは削除しても問題ありません。

スナップショットバックアップのタイムスタンプは `tiup br validate decode` コマンドで取得できます。表示されるタイムスタンプはTSOで18桁の数字になります。TSOについては2-3節を参照してください。

バックアップのタイムスタンプ確認

```
$ tiup br validate decode --field="end-version" \  
  --storage s3://tidb-book-sample/backup/cluster  
...  
(省略)  
...  
454622064107585537
```

`tiup br log truncate` コマンド実行時に、先ほど取得したタイムスタンプを `--until` オプションで設定してください。

ログバックアップ削除

```
$ tiup br log truncate --pd 127.0.0.1:2379 \  
  --until 454622064107585537 \  
  --storage s3://tidb-book-sample/log_backup/ --s3.region ap-northeast-1  
Starting component br: ...  
Detail BR log in /tmp/br.log.2024-12-15T06.53.02Z  
Reading log backup metadata... DONE { take = 442ms }  
We are going to truncate 1 files, up to TS 2024-12-15 06:52:24.8440.  
Are you sure?(y/N)
```

削除の確認が表示されますので、問題なければ `y` を入力してください。実行時に `--yes` オプションを指定することで確認を省略できます。削除されると次のようなメッセージが表示されます。

```
Truncating Data Files and Metadata :: DONE { take = 171ms, kv-count = 4, kv-size = 588(588B) }
```

実際には削除しない `--dry-run` オプションもあります。どれくらいファイルが削除されるのか、事前に確認したい場合に使用してください。

スナップショットからのリストア

スナップショットバックアップからのリストアは `tiup br restore` コマンドで行います。テーブル単位、データベース単位、TiDB全体のリストアが可能です。リストアを実行するためにはログバックア

ップを停止する必要があります。ログバックアップを停止していない場合は次のようなエラーメッセージが表示されますので、tiup br log stop コマンドでログバックアップを停止してください。

```
Error: failed to check task exists: log backup task is running: <task-name>, please stop the task
before restore, and after PITR operation finished, create log-backup task again and create a full
backup on this cluster
```

TiDB 全体をリストアする場合は tiup br restore full コマンドを使用します。--pd オプションで PD サーバーを指定します。--send-credentials-to-tikv オプションを指定することで TiKV ノードに認証情報を送信します。--storage オプションでスナップショットバックアップの保存先を指定してください。

TiDB 全体のリストア

```
$ tiup br restore full --pd 127.0.0.1:2379 \
--send-credentials-to-tikv \
--storage s3://tidb-book-sample/backup/cluster --s3.region ap-northeast-1
Starting component br: ...
Detail BR log in /tmp/br.log.2024-12-15T07.08.50Z
Full Restore <-----> 100.00%
[2024/12/15 07:08:56.978 +00:00] [INFO] ... ["Full Restore success summary"] ...
```

同様にデータベースをリストアする場合は tiup br restore db コマンドを使用します。復元するデータベースは --db オプションで指定します。リストア時にスナップショットを取得した際のデータベースの名前と違う場合、リストアできませんので注意してください。

データベースのリストア

```
$ tiup br restore db --pd 127.0.0.1:2379 \
--db bookshop \
--send-credentials-to-tikv \
--storage s3://tidb-book-sample/backup/database/bookshop \
--s3.region ap-northeast-1
```

テーブル単位でリストアする場合は tiup br restore table コマンドを使用します。--db オプションでデータベース名、--table オプションでテーブル名を指定してください。

テーブルのリストア

```
$ tiup br restore table --pd 127.0.0.1:2379 \
--db bookshop --table books \
--send-credentials-to-tikv \
--storage s3://tidb-book-sample/backup/table/books \
--s3.region ap-northeast-1
```

リストアを実行する場合は復元するデータベースやテーブルが存在しないことを確認してください。存在する場合は次のようなエラーが表示されます。

```
#####
# the target cluster is not fresh, cannot restore.
```

```
# you can drop existing databases and tables and start restore again
#####
```

TiDB 全体をリストアする場合、BR は一部の mysql スキーマ内のシステムテーブルも復元します。復元されるシステムテーブルは表 6.1 を参照してください。

表 6.1 復元可能なシステムテーブル

システムテーブル名
mysql.columns_priv
mysql.db
mysql.default_roles
mysql.global_grants
mysql.global_priv
mysql.role_edges
mysql.tables_priv
mysql.user
mysql.bind_info

復元しない場合は `--with-sys-table` オプションで `false` を指定してください。

システムテーブルを復元しない

```
$ tiup br restore full --pd 127.0.0.1:2379 \
--with-sys-table=false \
...
```

スナップショットバックアップに統計情報がある場合は自動的に統計情報も復元されます。統計情報を復元しない場合は `--load-stats` オプションで `false` を指定してください。

統計情報を復元しない

```
$ tiup br restore full --pd 127.0.0.1:2379 \
--load-stats=false \
...
```

▶ ポイントインタイムリストア

スナップショットバックアップとログバックアップを両方使用することでポイントインタイムリストア(一般的にはポイントインタイムリカバリーと呼ばれます)が可能になります。ポイントインタイムリストアは `tiup br restore point` コマンドで行います。リストアする時刻を確認するため、ログバックアップファイルを参照し、TSO(Time Stamp Oracle)を確認します。

TSOの確認

```
$ aws s3 ls --recursive s3://tidb-book-sample/log_backup/
...
(省略)
```

```
...
2024-12-15 17:22:17 385 log_backup/v1/backupmeta/454623474416091137-...f5d6.meta
2024-12-15 17:24:42 384 log_backup/v1/backupmeta/454623505873371138-...7fd9.meta
2024-12-15 17:27:07 385 log_backup/v1/backupmeta/454623537343758337-...e7ba.meta
2024-12-15 17:27:12 8 log_backup/v1/global_checkpoint/1.ts
2024-12-15 15:53:06 18 log_backup/v1_stream_truncate_safecheckpoint.txt
```

.meta ファイルの先頭部分が TSO です。TSO を確認し、ポイントインタイムリストアの際に `--restored-ts` オプションで指定します。なお `--restored-ts` オプションは TSO の値ではなく 2024-12-15 17:26:04+0900 のような日付でも問題ありません。

最新のスナップショットバックアップを `--full-backup-storage` オプションで指定し、ログバックアップを `--storage` オプションで指定してポイントインタイムリストアを実行します。

ポイントインタイムリストア

```
$ tiup br restore point --pd 127.0.0.1:2379 \
--restored-ts 454623537343758337 \
--storage s3://tidb-book-sample/log_backup \
--full-backup-storage s3://tidb-book-sample/backup/cluster \
--s3.region ap-northeast-1
Starting component br: ...
Detail BR log in /tmp/br.log.2024-12-15T08.46.39Z
Full Restore <-----> 100.00%
[2024/12/15 08:46:53.972 +00:00] [INFO] ... ["Full Restore success summary"] ...
Restore Meta Files <-----> 100.00%
Restore KV Files <-----> 100.00%
[2024/12/15 08:47:00.699 +00:00] [INFO] ... ["restore log success summary"] ...
```

Kubernetes環境でBRを使用してバックアップとリストアをする

Kubernetesに構築されたTiDB環境ではBackupリソースおよびBackupScheduleリソースを使用してBRによるバックアップを実行できます。Backupリソースを定義するとバックアップが即時に1回取得されます。定期的取得する場合はBackupScheduleリソースを定義してください。リストア時にはRestoreリソースを定義します。

準備として各種リソースが参照するSecretリソースを作成し、Amazon S3へのアクセスキーとシークレットキーを保存します。kubectlコマンドで作成するか、マニフェストを作成してkubectl applyコマンドで適用してください。Secretリソースの名前はaws-secretとします。

Secretの作成

```
$ kubectl create secret generic aws-secret \
--from-literal=access_key=xxxx --from-literal=secret_key=xxxx
secret/aws-secret created
```

Secretリソースのマニフェストの例はリスト6.1になります。stringDataを使用していますが、Data

を使用する際は値をBase64エンコードしてください(リスト6.1の❶)。

リスト6.1 Secretリソースのマニフェスト例

```
backup-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: aws-secret
stringData: ❶
  access_key: xxxx
  secret_key: xxxx
```

次にバックアップとリストア時に使用するServiceAccountリソースを作成し、適切な権限を割り当てます。マニフェストはリスト6.2のようになります。ServiceAccountの名前はtidb-backup-managerとし(リスト6.2の❶)、Roleリソース(リスト6.2の❷)とRoleBindingリソース(リスト6.2の❸)も作成して権限をServiceAccountに付与します。

リスト6.2 ServiceAccountリソースのマニフェスト例

```
backup-sa.yaml
---
kind: ServiceAccount
apiVersion: v1
metadata:
  name: tidb-backup-manager ❶
---
kind: Role ❷
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: tidb-backup-manager
  labels:
    app.kubernetes.io/component: tidb-backup-manager
rules:
- apiGroups: [""]
  resources: ["events"]
  verbs: ["*"]
- apiGroups: ["pingcap.com"]
  resources: ["backups", "restores"]
  verbs: ["get", "watch", "list", "update"]
---
kind: RoleBinding ❸
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: tidb-backup-manager
  labels:
    app.kubernetes.io/component: tidb-backup-manager
```

```
subjects:
- kind: ServiceAccount
  name: tidb-backup-manager
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: tidb-backup-manager
```

マニフェストを適用して ServiceAccount リソースと関連リソースを作成します。

ServiceAccountリソースの作成

```
$ kubectl apply -f backup-rbac.yaml
serviceaccount/tidb-backup-manager created
role.rbac.authorization.k8s.io/tidb-backup-manager created
rolebinding.rbac.authorization.k8s.io/tidb-backup-manager created
```

▶ Backupリソースの定義

TiDB 全体のスナップショットバックアップを Amazon S3 に取得する Backup リソースのマニフェストの例はリスト 6.3 になります。

.spec.serviceAccount には前項で作成した tidb-backup-manager を指定します(リスト 6.3 の❶)。BR の定義は .spec.br で行い、バックアップ対象のクラスター(TidbCluster リソース)の .metadata.name と .metadata.namespace を指定します(リスト 6.3 の❷)。.spec.s3 ではバケットの指定や認証情報の Secret リソースを指定します(リスト 6.3 の❸)。

.spec.backupType は full、db、table のいずれかを指定します(リスト 6.3 の❹)。full はクラスター全体のバックアップ、db はデータベースのバックアップ、table はテーブルのバックアップを行います。.spec.backupMode は snapshot、volume-snapshot、log のいずれかを指定します(リスト 6.3 の❺)。snapshot はスナップショットバックアップ、log はログバックアップ、volume-snapshot は Amazon EBS のスナップショット機能^{注4}を使用する場合に指定します。デフォルトは snapshot です。

リスト6.3 Backupリソースのマニフェスト例

```
backup.yaml
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: tidb-sample-backup
spec:
  backupType: full      ❹
  backupMode: snapshot  ❺
  cleanPolicy: OnFailure
  serviceAccount: tidb-backup-manager  ❶
```

注4 https://docs.aws.amazon.com/ja_jp/ebs/latest/userguide/ebs-snapshots.html

```
br: — ❷
  cluster: tidb-sample
  clusterNamespace: default
  logLevel: info
  sendCredToTikv: true
s3: — ❸
  provider: aws
  region: ap-northeast-1
  bucket: tidb-book-sample
  prefix: k8s/backup/cluster/tidb-sample
  secretName: aws-secret
```

マニフェストを適用し、`kubectl get backup` コマンドでリソースが作成されていることを確認してください。実際のバックアップ処理はJobリソースとして実行されます。

Backupリソースの作成

```
$ kubectl apply -f backup.yaml
backup.pingcap.com/tidb-sample-backup created

$ kubectl get backup
NAME                TYPE    MODE    STATUS    ...
tidb-sample-backup  full    snapshot Scheduled ...

$ kubectl get job
NAME                STATUS    COMPLETIONS  DURATION  AGE
backup-tidb-sample-backup  Running    0/1          3s        3s
```

実行が終了するとSTATUSがCompleteになります。

Backupリソースの確認

```
$ kubectl get backup
NAME                TYPE    MODE    STATUS    ...
tidb-sample-backup  full    snapshot Complete ...
```

Amazon S3のバケット内にバックアップファイルが保存されていることを確認してください。

バックアップファイルの確認

```
$ aws s3 ls --recursive s3://tidb-book-sample/k8s/backup/cluster/tidb-sample/
...
(省略)
...
2024-12-15 19:11:08 2290 k8s/backup/cluster/tidb-sample/1/76_110_..._write.sst
2024-12-15 19:11:04 78 k8s/backup/cluster/tidb-sample/backup.lock
2024-12-15 19:11:09 370 k8s/backup/cluster/tidb-sample/backupmeta
2024-12-15 19:11:08 3023 k8s/backup/cluster/tidb-sample/backupmeta.datafile.000000001
2024-12-15 19:11:09 288287 k8s/backup/cluster/tidb-sample/backupmeta.schema.000000002
```

データベースのみバックアップ対象にしたい場合は、次のどちらかの方法でBackupリソースを定義

してください。test データベースをバックアップ対象にした例になります。spec.backupType を db にする場合は対象のデータベースを spec.br.db で指定します。spec.backupType を full にする場合は spec.tableFilter を使用することでデータベースを指定してバックアップできます。

データベースを指定してバックアップ パターン1

```
...
(省略)
...
spec:
  backupType: db
  backupMode: snapshot
  cleanPolicy: OnFailure
  serviceAccount: tidb-backup-manager
  br:
    db: test
...
(省略)
...
```

データベースを指定してバックアップ パターン2

```
...
(省略)
...
spec:
  backupType: full
  backupMode: snapshot
  cleanPolicy: OnFailure
  serviceAccount: tidb-backup-manager
  tableFilter:
    - "test.*"
  br:
...
(省略)
...
```

特定のテーブルを対象にしたい場合も同様に次のどちらかの方法でBackup リソースを定義してください。spec.backupType を table にする場合は対象のテーブルを spec.br.db と spec.br.table で指定します。spec.backupType を full にする場合は spec.tableFilter を使用して対象のテーブルを指定してします。

テーブルを指定してバックアップ パターン1

```
...
(省略)
...
spec:
  backupType: table
```

```
backupMode: snapshot
cleanPolicy: OnFailure
serviceAccount: tidb-backup-manager
br:
  db: test
  table: sample
...
(省略)
...
```

テーブルを指定してバックアップ パターン2

```
...
(省略)
...
spec:
  backupType: full
  backupMode: snapshot
  cleanPolicy: OnFailure
  serviceAccount: tidb-backup-manager
  tableFilter:
    - "test.sample"
  br:
...
(省略)
...
```

.spec.tableFilterは正規表現を使用できます。たとえばtest.sample以外のすべてのテーブルをバックアップ対象にしたい場合は次のように記述します。

```
tableFilter:
  - "*,*"
  - "!test.sample"
```

ログバックアップはBackup リソースの .spec.backupMode を log に設定します(リスト6.4の❶)。

リスト6.4 ログバックアップのマニフェスト例

```
backup-log.yaml
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: tidb-sample-backup-log
spec:
  backupMode: log —❶
  cleanPolicy: OnFailure
  serviceAccount: tidb-backup-manager
  br:
    cluster: tidb-sample
```



```

clusterNamespace: default
logLevel: info
sendCredToTikv: true
s3:
  provider: aws
  region: ap-northeast-1
  bucket: tidb-book-sample
  prefix: k8s/log_backup/cluster/tidb-sample
  secretName: aws-secret

```

マニフェストを適用し、バックアップが正常に完了していることを確認してください。.spec.backupModeをlogに設定した場合はSTATUSがRunningのままになり、定期的にログバックアップが行われます。

ログバックアップリソースの作成と確認

```

$ kubectl apply -f backup-log.yaml
backup.pingcap.com/tidb-sample-backup-log configured

$ kubectl get backup
NAME                                TYPE    MODE    STATUS    ...
tidb-sample-backup-log              log     log     Running   ...

```

ログバックアップの定期実行を止める場合はBackupリソース内で.spec.logStopをtrueに設定します。マニフェストを更新して適用するか、kubectl patchコマンドで更新してください。

ログバックアップの停止

```

$ kubectl patch backup tidb-sample-backup-log \
  --type='json' -p='[{"op": "replace", "path": "/spec/logStop", "value": true}]'
backup.pingcap.com/tidb-sample-backup-log patched

```

ログバックアップのクリーンアップは.spec.logTruncateUntilに削除する日時またはTSOを指定します。指定した日時より前のログが削除されます。.spec.logTruncateUntilは文字列型で指定する必要があるため、TSOの値を指定する場合は"454623537343758337"のように"を使用して指定してください。日時で指定する場合は"2024-12-14T12:00:00+0900"のように指定してください。

ログバックアップのクリーンアップ

```

apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: tidb-sample-backup-log
spec:
  backupMode: log
  logTruncateUntil: "454623537343758337"
...
(省略)
...

```

BackupSchedule リソースの定義

定期的にスナップショットバックアップを取得する場合は BackupSchedule リソースを使用します(リスト 6.5)。前項で定義した Backup リソースを BackupSchedule リソースで毎時間定期実行する例になります。

BackupSchedule リソース内では `.spec.backupTemplate` で定期実行させたい Backup リソースの内容を定義します(リスト 6.5 の ❶)。`.spec.schedule` には Cron 形式でバックアップのスケジュールを設定します(リスト 6.5 の ❷)。スケジュールは UTC で記述する必要がありますので注意してください。`.spec.maxBackups` は保持するバックアップの数を指定します(リスト 6.5 の ❸)。

リスト 6.5 BackupSchedule リソースのマニフェスト例

```
backup-schedule.yaml
apiVersion: pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: tidb-sample-schedule-backup
spec:
  schedule: "0 * * * *" — ❷
  maxBackups: 5 — ❸
  backupTemplate: — ❶
    backupType: full
    backupMode: snapshot
    cleanPolicy: OnFailure
    serviceAccount: tidb-backup-manager
    br:
      cluster: tidb-sample
      clusterNamespace: default
      logLevel: info
      sendCredToTikv: true
    s3:
      provider: aws
      region: ap-northeast-1
      bucket: tidb-book-sample
      prefix: k8s/schedule-backup/cluster/tidb-sample
      secretName: aws-secret
```

マニフェストを適用し、`kubectl get backupschedule` コマンドでリソースが作成されていることを確認してください。

BackupSchedule リソースの作成

```
$ kubectl apply -f backup-schedule.yaml
backupschedule.pingcap.com/tidb-sample-schedule-backup created

$ kubectl get backupschedule
NAME                                SCHEDULE    MAXBACKUPS    MAXRESERVEDTIME    AGE
tidb-sample-schedule-backup        0 * * * *    5              3s
```

指定したバケット内に実行時刻ごとにフォルダが作成され、バックアップされていることを確認します。

バックアップファイルの確認

```
$ aws s3 ls s3://tidb-book-sample/k8s/schedule-backup/cluster/tidb-sample/
PRE tidb-sample-pd.default-2379-2024-12-14t20-00-00/
PRE tidb-sample-pd.default-2379-2024-12-14t21-00-00/
PRE tidb-sample-pd.default-2379-2024-12-14t22-00-00/
```

定期的な実行を止める場合はBackupSchedule リソース内で .spec.pause を true に設定します。マニフェストを更新して適用するか、kubectl patch コマンドで更新してください。

バックアップスケジュールの停止

```
$ kubectl patch backupschedule tidb-sample-schedule-backup \
  --type='json' -p='[{"op": "replace", "path": "/spec/pause", "value": true}]'
backupschedule.pingcap.com/tidb-sample-schedule-backup patched
```

定期的な実行の有効／無効状態を確認する場合は kubectl describe backupschedule <バックアップスケジュール名> コマンドや kubectl get backupschedule <バックアップスケジュール名> -o yaml コマンドで PAUSE 部分を確認してください。または次の kubectl get backupschedule コマンドで確認できます。

バックアップスケジュールの有効／無効確認

```
$ kubectl get backupschedule -o 'custom-columns=NAME:.metadata.name,PAUSE:.spec.pause'
NAME                                PAUSE
tidb-sample-schedule-backup        true
```

kubectl get backupschedule コマンドは --output オプションに wide を指定すると最後にバックアップした日時が LASTBACKUP として表示されますので、LASTBACKUP を確認してスケジュールが停止されているか確認しても良いでしょう。

バックアップスケジュールの最終実行日時確認

```
$ kubectl get backupschedule --output wide
... LASTBACKUP                                LASTBACKUPTIME  AGE
... tidb-sample-schedule-backup-2024-12-14t22-00-00  70m             90m
```

▶ Restore リソースの定義

Backup リソースで取得したバックアップをリストアする場合は Restore リソースを使用します(リスト 6.6)。Restore リソース定義はほぼ Backup リソースと同じです。バックアップされたデータの場所を .spec.s3.bucket と .spec.s3.prefix で指定します(リスト 6.6 の ❶ と ❷)。

リスト6.6 Restoreリソースのマニフェスト例

```
restore.yaml
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: tidb-sample-restore
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  br:
    cluster: tidb-sample
    clusterNamespace: default
    logLevel: info
    sendCredToTikv: true
  s3:
    provider: aws
    region: ap-northeast-1
    bucket: tidb-book-sample — ❶
    prefix: k8s/schedule-backup/cluster/tidb-sample/... — ❷
    secretName: aws-secret
```

バックアップの保存先を確認する場合は `kubectl get backup` コマンドで確認すると良いでしょう。
`BACKUPPATH` に表示されます。

バックアップの保存先確認

```
$ kubectl get backup
NAME                TYPE    MODE    STATUS    BACKUPPATH    ...
tidb-sample-backup  full    snapshot Complete  s3://tidb-book-sample/k8s/...  ...
```

Restore リソースによるリストアは、ログバックアップが有効になっているとエラーが発生してリストアできません。ログバックアップを停止してからリストアを行ってください。ログバックアップの Backup リソースを削除するか、`.spec.logStop` を `true` に設定することで停止できます。

リストアの状況は `kubectl get restore` コマンドで確認できます。正常にリストアが終了すると `STATUS` が `Complete` になります。

```
$ kubectl get restore
NAME                STATUS    TIMETAKEN    COMMITTS    AGE
tidb-sample-restore Complete    6s           454627142363250689  75s
```

執筆時点では Restore リソースによるポイントインタイムリストアはサポートされていません。こちらに関しては関連 issue が [GitHub](https://github.com/pingcap/tidb-operator/issues/3017)^{注5} にありますので、最新の情報を確認してください。

注5 <https://github.com/pingcap/tidb-operator/issues/3017>

Dumplingを使用してデータをエクスポートする

DumplingはTiDBからデータをエクスポートするツールです。TiDBだけではなくMySQLに対しても利用可能で、CSVまたはSQL形式でエクスポート可能です。エクスポート先はローカルファイルとパブリッククラウドのオブジェクトストレージに対応しています。エクスポートする際にさまざまな条件を指定でき、サイズによって出力ファイルを分割したりなど高機能かつ高速なエクスポートが可能なツールです。エクスポートされたデータはTiDB Lightningを使用してインポートできます。TiDB Lightningについてはこの後説明します。

▶ Dumplingのインストール

DumplingをインストールするにはTiUPを使用します。

```
TiUPを使用したDumplingのインストール
```

```
$ tiup install dumpling
```

Dumplingを実行する場合はtiup dumplingコマンドを使用します。

```
Dumplingの実行確認
```

```
$ tiup dumpling --version
Starting component dumpling: ...
Release version: v8.5.0
Git commit hash: 1a9f0fa25580907d3266dc3edf000ba24379a600
Git branch:      HEAD
Build timestamp: 2024-11-07 03:18:43Z
Go version:      go version go1.23.2 linux/amd64
```

インストール可能なDumplingのバージョンはtiup list dumplingコマンドで確認できます。

```
Dumplingのバージョンリスト確認
```

```
$ tiup list dumpling
Available versions for dumpling:
Version          Installed Release
-----
nightly -> v9.0.0-alpha-nightly 2024-12-28T07:27:14Z ...
...
(省略)
...
v8.5.0           YES      2024-12-19T06:06:21Z ...
v9.0.0-alpha-nightly 2024-12-28T07:27:14Z ...
```

▶ TiDBまたはMySQLからのエクスポート

Dumplingの動作確認のためにデモデータを使用します。TiDB PlaygroundでTiDBを起動し、デモ用

のデータを投入します。TiDB Playgroundの使い方は3-2節を参照してください。デモ用データの投入の仕方は3-10節を参照してください。

TiDB Playgroundを起動

```
$ tiup playground
```

別端末からデモ用データの投入

```
$ tiup demo bookshop prepare
```

デモ用データが投入されたらDumplingを使用してデータをエクスポートします。デフォルトではエクスポートされたデータはローカルディスクに保存されます。Dumpling実行時にはTiDBまたはMySQLのホスト名、認証情報を指定する必要があります。--host、--port、--user、--passwordなどのオプションを適宜使用してください。本項ではTiDB Playgroundに接続しますので--host 127.0.0.1 --port 4000 --user rootのように指定します。

Dumplingによるエクスポート

```
$ tiup dumpling --host 127.0.0.1 --port 4000 --user root
```

エクスポートされたデータは、デフォルトでは現在のディレクトリのexport-<実行日時>ディレクトリに出力されます。出力先のディレクトリは--outputオプションで指定できます。なお、ディレクトリが無い場合は作成されます。

エクスポートされたデータの確認

```
$ ls ./export-2024-12-16T07:10:34Z
bookshop.authors.0000000000.sql      bookshop.ratings.0000000000.sql
bookshop.authors-schema.sql          bookshop.ratings-schema.sql
bookshop.book_authors.0000000000.sql  bookshop-schema-create.sql
bookshop.book_authors-schema.sql      bookshop.users.0000000000.sql
bookshop.books.0000000000.sql         bookshop.users-schema.sql
bookshop.books-schema.sql            metadata
bookshop.orders.0000000000.sql        test-schema-create.sql
bookshop.orders-schema.sql
```

出力されるファイルの内容は次の通りです。

- **metadata**
出力の開始と終了日時およびSHOW MASTER STATUSの情報
- **<データベース>-schema-create.sql**
データベースのスキーマ(CREATE DATABASE)
- **<データベース>-<テーブル>-schema.sql**
テーブルのスキーマ(CREATE TABLE)
- **<データベース>.<テーブル>.<ファイルインデックス>.(sql|csv)**
テーブルのデータ(INSERT)

テーブルデータが出力されるファイル名は`--output-filename-template`オプションで変更できます。ファイル名に指定できる変数として`{{.DB}}`、`{{.Table}}`、`{{.Index}}`があり、`{{.DB}}`はデータベース名、`{{.Table}}`はテーブル名、`{{.Index}}`はファイルインデックスで、000000000から始まる単調増加の数字になります。デフォルトは`{{.DB}}`、`{{.Table}}`、`{{.Index}}`となります。

CSV形式で出力させる場合は`--filetype`オプションに`csv`を指定します。デフォルトは`sql`です。

DumplingによるCSV形式でのエクスポート

```
$ tiup dumpling --host 127.0.0.1 --port 4000 --user root --filetype csv
```

CSV形式でエクスポートした場合はテーブルのデータがCSV形式になります。各種スキーマ情報はSQL形式のままになります。

CSV形式でエクスポートされたデータの確認

```
$ ls ./export_dir
bookshop.authors.000000000.csv      bookshop.ratings.000000000.csv
bookshop.authors-schema.sql         bookshop.ratings-schema.sql
bookshop.book_authors.000000000.csv bookshop-schema-create.sql
bookshop.book_authors-schema.sql    bookshop.users.000000000.csv
bookshop.books.000000000.csv        bookshop.users-schema.sql
bookshop.books-schema.sql           metadata
bookshop.orders.000000000.csv       test-schema-create.sql
bookshop.orders-schema.sql
```

CSVファイルに使用する区切り文字を変更したい場合は`--csv-separator`オプションで指定してください。デフォルトは`,`です。

CSVファイルの区切り文字を変更

```
$ tiup dumpling --host 127.0.0.1 --port 4000 --user root \
  --filetype csv --csv-separator '|'

$ head export-2024-12-16T07:31:09Z/bookshop.authors.000000000.csv
"id"|"name"|"gender"|"birth_year"|"death_year"
211345|"Tad Renner"|0|1988|\N
261367|"Dedric Towne"|0|1934|1953
773077|"Robyn Fisher"|1|1981|1998
1081956|"Brandt Greenholt"|0|1989|\N
1276498|"Toney Littel"|1|1987|2013
1283251|"Katheryn Buckridge"|0|1948|1973
1606911|"Shanie Rempel"|1|2000|\N
1701931|"Gabriella Cremin"|0|1997|\N
1783085|"Casimer Hamill"|1|1953|1977
```

データベースを指定してエクスポートする場合は`--database`オプションを使用します。

データベースを指定してエクスポート

```
$ tiup dumpling --host 127.0.0.1 --port 4000 --user root --database bookshop
```

テーブルを指定してエクスポートする場合は `--tables-list` オプションを使用します。複数のテーブルを指定する場合はカンマ区切りで指定してください。bookshop データベース内の books テーブルと orders テーブルをエクスポートする場合は次のように実行します。

テーブルを指定してエクスポート

```
$ tiup dumping --host 127.0.0.1 --port 4000 --user root \  
--tables-list bookshop.books,bookshop.orders
```

より複雑な条件で対象をフィルターしたい場合は `--filter` オプションを使用してください。たとえば bookshop データベース内で books テーブル以外を対象にしたい場合は次のように指定します。`--database` オプションや `--tables-list` オプションと同時に使用できませんので注意してください。フィルターの詳細な書き方は Table Filter ドキュメント^{注6}を参照してください。

複雑なフィルターを使用したエクスポート

```
$ tiup dumping --host 127.0.0.1 --port 4000 --user root \  
--filter 'bookshop.*' --filter '!bookshop.books'
```

エクスポートしたファイルを圧縮したい場合は `--compress` オプションを使用します。指定できる圧縮形式としては gzip、snappy、zstd があります。デフォルトは no-compression となっており、圧縮されません。なお、圧縮は個々のファイルに対して行われます。

エクスポートファイルの圧縮

```
$ tiup dumping --host 127.0.0.1 --port 4000 --user root --compress gzip
```

```
$ ls export-2024-12-16T07:34:55Z  
bookshop.authors.0000000000.sql.gz      bookshop.ratings.0000000000.sql.gz  
bookshop.authors-schema.sql.gz          bookshop.ratings-schema.sql.gz  
bookshop.book_authors.0000000000.sql.gz  bookshop-schema-create.sql.gz  
bookshop.book_authors-schema.sql.gz      bookshop.users.0000000000.sql.gz  
bookshop.books.0000000000.sql.gz        bookshop.users-schema.sql.gz  
bookshop.books-schema.sql.gz            metadata  
bookshop.orders.0000000000.sql.gz        test-schema-create.sql.gz  
bookshop.orders-schema.sql.gz
```

巨大なデータをエクスポートする場合は出力ファイルを分割できます。`--filesize` オプションを指定すると、そのサイズでファイルが分割されて保存されます。10KB や 1GiB などの指定が可能で、単位を省略すると MiB として扱われます。

エクスポートファイルの分割

```
$ tiup dumping --host 127.0.0.1 --port 4000 --user root --filesize 5MiB
```

bookshop データベース内では orders テーブルと ratings テーブルが 5MiB を超えているので分割され

注6 <https://docs.pingcap.com/tidb/stable/table-filter>

て保存されます。分割されたファイルは数字部分(000000000や000000001など)がインクリメントされて保存されます。

```
$ ls export-2024-12-16T07:48:20Z
bookshop.authors.000000000.sql      bookshop.orders-schema.sql
bookshop.authors-schema.sql         bookshop.ratings.000000000.sql
bookshop.book_authors.000000000.sql bookshop.ratings.000000001.sql
bookshop.book_authors-schema.sql    bookshop.ratings.000000002.sql
bookshop.books.000000000.sql        bookshop.ratings-schema.sql
bookshop.books-schema.sql           bookshop-schema-create.sql
bookshop.orders.000000000.sql       bookshop.users.000000000.sql
bookshop.orders.000000001.sql       bookshop.users-schema.sql
bookshop.orders.000000002.sql       metadata
bookshop.orders.000000003.sql       test-schema-create.sql
```

エクスポートする際にSQLでデータをフィルターする機能もあり、SQLのWHERE句を--whereオプションで渡すことができます。たとえば生まれた年が1990年以降である著者をエクスポートする場合は次のように指定します。テーブルを指定しない場合は対象のテーブルすべてにWHERE句が発行されます。

```
$ tiup dumpling --host 127.0.0.1 --port 4000 --user root \
--tables-list "bookshop.authors" --where "birth_year >= 1990"

$ head export-2024-12-16T08:37:12Z/bookshop.authors.000000000.sql
/*!40014 SET FOREIGN_KEY_CHECKS=0*/;
/*!40101 SET NAMES binary*/;
INSERT INTO `authors` VALUES
(1606911,'Shanie Rempel',1,2000,NULL),
(1701931,'Gabriella Cremin',0,1997,NULL),
(3107282,'Adelia Lebsack',1,1991,1997),
(3155041,'Rey Effertz',1,1991,NULL),
(3442626,'Haylee Macejkovic',1,1996,NULL),
(3549059,'Kavon Simonis',1,1990,NULL),
(3990661,'Cameron Breitenberg',1,2000,NULL),
```

Dumplingは--consistencyオプションを使用することで、エクスポート時の一貫性の保証を制御できます。flushを指定した場合は一貫性を確保するためにFLUSH TABLES WITH READ LOCKが使用されます。snapshotを指定した場合は指定されたタイムスタンプの一貫したスナップショットを取得してエクスポートします。lockを指定した場合はエクスポートするすべてのテーブルに読み取りロックが追加されます。TiDBに対しても指定できますが、TiDBはデフォルトではテーブルロックが無効になっていますので、enable-table-lock^{注7}を有効にする必要があります。noneを指定した場合は一貫性は保証されません。autoを指定した場合、MySQLではflush、TiDBではsnapshotが使用されます。デフォルト

注7 <https://docs.pingcap.com/tidb/stable/tidb-configuration-file#enable-table-lock-new-in-v400>

はautoになります。

一貫性の保証を制御する

```
$ tiup dumpling --host 127.0.0.1 --port 4000 --user root --consistency snapshot
```

なお、`--consistency` オプションの `flush` は TiDB に対して指定できません。指定した場合、Dumpling は次のようなエラーを返します。

```
dump failed: 'flush table with read lock' cannot be used to ensure the consistency in TiDB
```

`--consistency` オプションが `snapshot` の場合のみ、`--snapshot` オプションでスナップショットを取得する時刻を指定できます。指定する時刻は `YYYY-MM-DD HH:MM:SS` の形式か TSO の値を指定してください。

スナップショットの取得時刻を指定

```
$ tiup dumpling --host 127.0.0.1 --port 4000 --user root \  
--consistency snapshot --snapshot 454657866903846913  
  
$ tiup dumpling --host 127.0.0.1 --port 4000 --user root \  
--consistency snapshot --snapshot "2024-12-16 20:48:41"
```

TiDB Cloud に接続して Dumpling を実行する場合は、GC (ガベージコレクション) の保持期間の設定を変更してください。PD サーバーにアクセス可能な環境であれば、Dumpling はバックアップ中にデータが GC されてしまうことを防ぐために GC をブロックしますが、TiDB Cloud 環境では PD サーバーにアクセスできないため、GC の保持期間の設定を手動で変更してください。

設定変更は次のように行います。最初に保持期間を 30 日間 (720 時間) に設定していますが、Dumpling の実行時間よりも長ければ問題ありません。Dumpling が終了したら元の設定に戻すことを忘れないでください。デフォルトは 10 分です。

GC の設定変更 (TiDB Cloud 環境のみ)

```
mysql> SET GLOBAL tidb_gc_life_time = '720h';  
  
mysql> SET GLOBAL tidb_gc_life_time = '10m';
```

外部ストレージにエクスポートする

Dumpling はエクスポート先としてパブリッククラウドのオブジェクトストレージに対応しています。執筆時点では Amazon S3、Google Cloud Storage、Azure Blob Storage に対応しています。外部ストレージを出力先に指定する場合は `--output` オプションで URI 形式で指定します。Amazon S3 の認証情報の設定の仕方は本節冒頭の説明を参照してください。URI 形式のパラメーターの詳細はドキュメントの [URI Formats of External Storage Services](https://docs.pingcap.com/tidb/stable/external-storage-uri)^{注8}を参照してください。Amazon S3 に対してエクスポートす

注8 <https://docs.pingcap.com/tidb/stable/external-storage-uri>

る場合は次のように指定します。

オブジェクトストレージにエクスポート

```
$ tiup dumpling --host 127.0.0.1 --port 4000 --user root \
--output s3://tidb-book-sample/export
```

TiDB Lightningを使用してデータをインポートする

TiDB Lightning は Dumpling でエクスポートされたデータを TiDB にインポートするツールです。Dumpling でエクスポートされたデータ以外にも、CSV ファイルや Amazon Aurora によって生成された Apache Parquet ファイルをインポートできます。

TiDB Lightning は物理インポートモードと論理インポートモードの2つのモードがあります。物理インポートモードはデータをキーとバリューの値にエンコードして TiKV サーバーにアップロードし、TiKV に直接データを書き込むため高速なインポートが可能です。物理インポートモードは対象のテーブルにデータが存在していると実行できないため、TiDB を構築後に初期インポートを実行する場合に適しています。論理インポートモードはデータを SQL にエンコードして、INSERT としてデータを書き込むため物理インポートモードより遅いですが、インポート対象のテーブルにすでにデータが存在している場合でも実行できます。それぞれのモードの比較を表6.2に示します。

表6.2 TiDB Lightningの実行モードの比較

	物理インポートモード	論理インポートモード
インポート速度	速い(時間あたり100~500GiB)	遅い(時間あたり10~50GiB)
リソース消費量	多い	少ない
ネットワーク帯域消費量	多い	少ない
インポート時のACID準拠	無し	有り
対象テーブル	テーブル内データが空である必要がある	テーブル内データが存在していても実行可能
インポート中のTiDBの利用	限定的	可能

物理インポートモード時の TiDB の利用は制約事項があり、限定的になります。詳細は Physical Import Mode ドキュメント内の Limitations^{注9}を参照してください。

TiDB Lightningのインストール

TiDB Lightning をインストールするには TiUP を使用します。

TiUPを使用したTiDB Lightningのインストール

```
$ tiup install tidb-lightning
```

注9 <https://docs.pingcap.com/tidb/stable/tidb-lightning-physical-import-mode#limitations>

TiDB Lightning を実行する場合は `tiup tidb-lightning` コマンドを使用します。

TiDB Lightning の実行確認

```
$ tiup tidb-lightning -V
Starting component tidb-lightning: ...
Release Version: v8.5.0
Git Commit Hash: d13e52ed6e22cc5789bed7c64c861578cd2ed55b
Git Branch: HEAD
Go Version: go1.23.3
UTC Build Time: 2024-12-18 02:28:25
Race Enabled: false
```

インストール可能な Dumpling のバージョンは `tiup list tidb-lightning` コマンドで確認できます。

TiDB Lightning のバージョンリスト確認

```
$ tiup list tidb-lightning
Available versions for tidb-lightning:
Version                Installed  Release                ...
-----                -
nightly -> v9.0.0-alpha-nightly      2024-12-28T07:27:36Z ...
...
(省略)
...
v8.5.0                  YES       2024-12-19T06:06:44Z ...
v9.0.0-alpha-nightly    2024-12-28T07:27:36Z ...
```

▶ IMPORT INTO 構文と TiDB Lightning

TiDB バージョン 7.5.0 から TiDB Lightning における物理インポートモードを利用するための `IMPORT INTO` 構文が導入されました。これにより TiDB Lightning をインストールする必要なく高速にインポートできます。`IMPORT INTO` を使用するためには `tidb_enable_dist_task` を有効にする必要があります。TiDB バージョン 8.1.0 からは `tidb_enable_dist_task` はデフォルトで有効になっています。

tidb_enable_dist_task の確認

```
mysql> SELECT @@global.tidb_enable_dist_task;
+-----+
| @@global.tidb_enable_dist_task |
+-----+
| 1 |
+-----+
1 row in set (0.00 sec)
```

`IMPORT INTO` については第 7 章の「SQL によるバックアップとリストア」で説明します。
論理インポートモードを使用する場合は引き続き TiDB Lightning を使用してください。

▶ 物理インポートモード

TiDB Lightning で物理インポートモードを使用する場合は `-backend` オプションで `local` を指定します。TiDB への接続情報は `-tidb-host`、`-tidb-port`、`-tidb-user`、`-tidb-password` オプションを使用して指定します。

物理インポートモードは一時的なデータを保存するディレクトリを `-sorted-kv-dir` で指定する必要があります。パフォーマンスの問題を回避するために、高速なストレージを使用すること、データソースディレクトリのストレージとは別のストレージにあることが推奨されています。また、TiDB Lightning 実行時にはディレクトリ内は空である必要があります。

次の例は、Dumpling でエクスポートした bookshop データベースのデータを bookshop_export ディレクトリに配置し、一時的な作業ディレクトリとして `/tmp/sorted-kv-dir` を指定して TiDB Lightning を実行します。実行すると事前チェックが行われ、問題がなければインポートが開始されます。

物理インポートモードの実行

```
$ tiup tidb-lightning -backend local \
  -tidb-host 127.0.0.1 -tidb-port 4000 -tidb-user root \
  -d ./bookshop_export -sorted-kv-dir /tmp/sorted-kv-dir
Starting component tidb-lightning: ...
Verbose debug logs will be written to /tmp/lightning.log.2024-12-17T00.24.04Z
```

#	CHECK ITEM	TYPE	PASSED
1	Source data files size is proper	performance	true
2	the checkpoints are valid	critical	true
3	table schemas are valid	critical	true
...			
...	(省略)		
...			
10	Cluster doesn't have too many empty regions	performance	true
11	Cluster region distribution is balanced	performance	true
12	no CDC or PiTR task found	critical	true

```
tidb lightning exit successfully
```

「tidb lightning exit successfully」が表示されれば正常に終了しています。

特定のテーブルのみを対象にする場合は `-f` オプションを使用します。たとえば `bookshop.books` テー

ブルのみを物理インポートする場合は次のように指定します。

特定のテーブルのみを物理インポート

```
$ tiup tidb-lightning -backend local \  
-tidb-host 127.0.0.1 -tidb-port 4000 -tidb-user root \  
-d ./bookshop_export -sorted-kv-dir /tmp/sorted-kv-dir \  
-f bookshop.books
```

フィルターの詳細な書き方はドキュメントの Table Filter^{注10}を参照してください。

▶ 論理インポートモード

TiDB Lightningで論理インポートモードを使用する場合は-backend オプションでtidbを指定します。

論理インポートモードの実行

```
$ tiup tidb-lightning -backend tidb \  
-tidb-host 127.0.0.1 -tidb-user root -tidb-port 4000 -d ./bookshop_export  
Starting component tidb-lightning: ...  
Verbose debug logs will be written to /tmp/lightning.log.2024-12-17T00.40.58Z
```

#	CHECK ITEM	TYPE	PASSED
1	Source data files size is proper	performance	true
2	the checkpoints are valid	critical	true
3	Cluster version check passed	critical	true
4	Lightning has the correct storage permission	critical	true

```
tidb lightning exit successfully
```

物理インポートモードの時と同様に、-f オプションを使用して特定のテーブルのみをインポート対象にできます。たとえばbookshop.booksテーブルのみを論理インポートする場合は次のように指定します。

特定のテーブルのみを論理インポート

```
$ tiup tidb-lightning -backend tidb \  
-tidb-host 127.0.0.1 -tidb-user root -tidb-port 4000 -d ./bookshop_export \  
-f bookshop.books
```

▶ 設定ファイルを使用した実行

TiDB Lightningはオプションで指定する項目を設定ファイルに記述し、-config オプションで指定で

注10 <https://docs.pingcap.com/tidb/stable/table-filter>

きます。設定ファイルはTOML形式で記述します。設定ファイルの例を以下に示します。

物理インポートモードの設定ファイル

```
[lightning]
level = "info"

[mydumper]
data-source-dir = "./bookshop_export"

[tikv-importer]
backend = "local"
sorted-kv-dir = "/tmp/sorted-kv-dir"

[tidb]
host = "127.0.0.1"
port = 4000
user = "root"
password = ""
```

設定ファイルを使用して TiDB Lightning を実行する場合は次のように指定します。

設定ファイルを使用してTiDB Lightningを実行

```
$ tiup tidb-lightning -config ./config.toml
```

設定ファイルの完全な書式はドキュメントの TiDB Lightning Configuration^{注11}を参照してください。

▶ Webインターフェース経由での実行

TiDB Lightning はサーバーモードで実行することで、Web インターフェースを使用して実行できます。サーバーモードで実行するには `-server-mode` オプションを指定します。待ち受けるアドレスと Port 番号は `-status-addr` オプションで指定します。

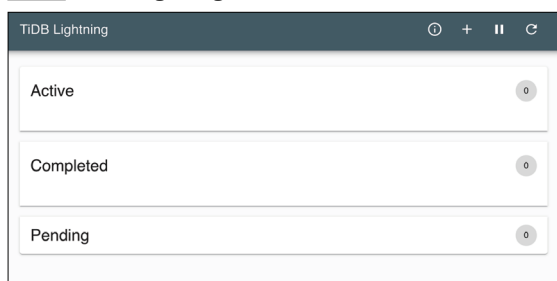
サーバーモードでTiDB Lightningを実行

```
$ tiup tidb-lightning -server-mode -status-addr :8289
```

ブラウザでアクセスすると TiDB Lightning の Web インターフェースが表示されます (図 6.2)。

画面上部の「+」ボタンをクリックすると TiDB Lightning を実行するための設定画面が

図6.2 TiDB Lightningトップ画面



注11 <https://docs.pingcap.com/tidb/stable/tidb-lightning-configuration>

図6.3 TiDB Lightningの設定画面

Submit task

UPLOAD

Task configuration

```
[lightning]
level = "info"

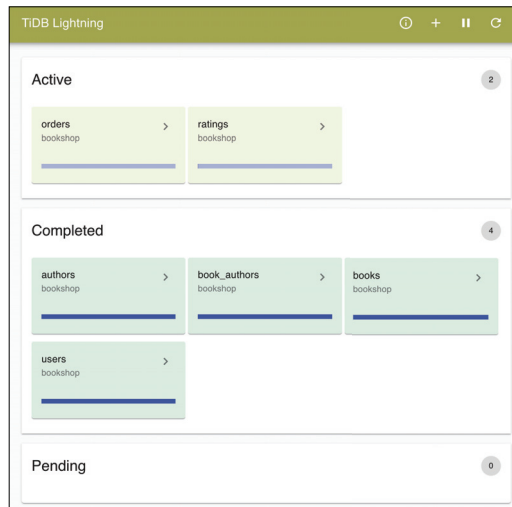
[mydumper]
data-source-dir = "/home/tidb/bookshop_export"

[tikv-importer]
backend = "local"
sorted-kv-dir = "/tmp/sorted-kv-dir"

[tidb]
host = "127.0.0.1"
port = 4000
user = "root"
password = ""
```

CANCEL SUBMIT

図6.4 TiDB Lightningの実行中画面



表示されます(図6.3)。設定ファイルをアップロードするか、直接書き込みます。

設定が完了したら「SUBMIT」をクリックするとTiDB Lightningが実行されます。実行中(Activeなテーブルがある状態の時)は画面上部のバーが黄色く表示されます(図6.4)。

テーブルごとに進捗状況が表示されます。カード上の矢印(>)をクリックすると処理の詳細を確認できます(図6.5)。

すべてのテーブルがCompletedで表示されていれば実行が終了しています。正常に終了していれば画面上部のバーが緑色で表示されます(図6.6)。

図6.5 TiDB Lightningの実行タスク詳細画面

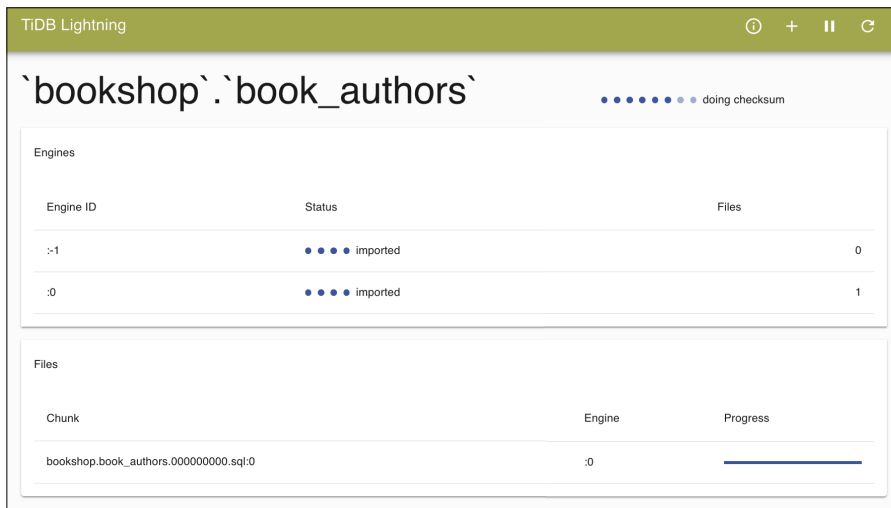
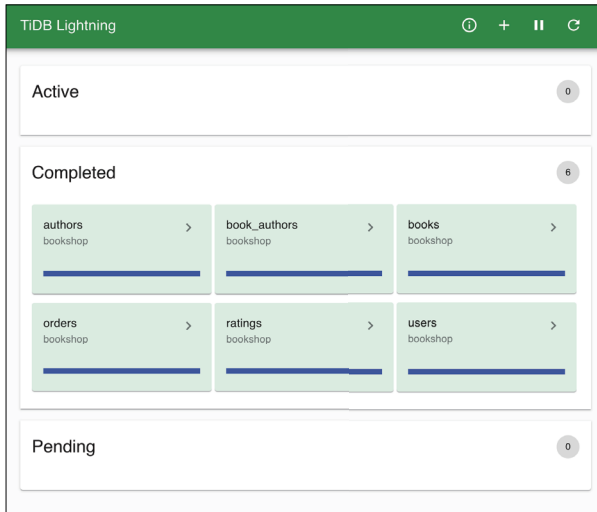


図6.6 TiDB Lightningの実行終了画面



6-2

TiCDCによるTiDB外へのデータ同期

TiCDCはTiDBの更新差分を外部システムに連携するツールです。CDC(Change Data Capture)^{注12}とは、データベースの変更をリアルタイムに取得し、他のシステムに連携する技術です。TiCDCはTiDBの更新情報を取得し、MySQLはもちろんApache Kafka^{注13}やパブリッククラウドのオブジェクトストレージなどの外部システムにデータを同期できます。

TiCDCはチェンジフィードと呼ばれる単位でデータ同期を定義します。チェンジフィードに関する操作はTiCDCサーバーのAPIを通じて行い、TiCDCのクライアントを使用して操作します。

TiCDCを稼働させるサーバーは、本番環境では最低2台起動させることが推奨になっています。その他の推奨要件はドキュメントのSoftware and Hardware Requirements^{注14}を参照してください。

TiCDCクライアントをインストールする

TiCDCの各種操作はcdcコマンドを使用します。TiUPを使用してインストールします。

TiUPによるTiCDCクライアントインストール

```
$ tiup install cdc
```

注12 https://en.wikipedia.org/wiki/Change_data_capture

注13 <https://kafka.apache.org/>

注14 <https://docs.pingcap.com/tidb/stable/hardware-and-software-requirements/#production-environment>

TiCDC クライアントを使用する場合は `tiup cdc` コマンドを実行します。

TiUP から TiCDC クライアントを使用する

```
$ tiup cdc version
Starting component cdc: ...
Release Version: v8.5.0
Git Commit Hash: 3703b18c6da785d57236a1452eed43759e51b411
Git Branch: HEAD
UTC Build Time: 2024-12-17 04:34:05
Go Version: go1.23.3
Failpoint Build: false
```

インストール可能な TiCDC クライアントのバージョンは `tiup list cdc` コマンドで確認できます。

TiCDC のバージョンリスト確認

```
$ tiup list cdc
Available versions for cdc:
Version                Installed  Release      ...
-----
nightly -> v9.0.0-alpha-nightly      2024-12-27T07:56:04Z ...
...
(省略)
...
v8.5.0                  YES       2024-11-11T06:29:50Z ...
v9.0.0-alpha-nightly    2024-12-27T07:56:04Z ...
```

TiDB Playground で TiCDC を起動する

TiDB Playground で TiCDC を起動する場合は `--ticdc` オプションで起動する台数を指定します。デフォルトは 0 になっていますので起動しません。

TiDB Playground で TiCDC を起動

```
$ tiup playground --ticdc 1
```

起動確認のため、TiCDC クライアントを実行します。TiDB Playground 環境で起動した TiCDC は `127.0.0.1:8300` で接続を待ち受けます。TiCDC クライアントでは `--server` オプションで接続先を指定しますので、`127.0.0.1:8300` を指定して実行します。現在は何も設定していませんので `[]` が表示されます。

TiCDC クライアントの実行

```
$ tiup cdc cli changefeed list --server 127.0.0.1:8300
Starting component cdc: ...
[]
```

動作確認のため、MySQLを起動してTiDB PlaygroundからMySQLへデータ同期を行います。本項ではDockerを使用してMySQLを起動します。検証目的ですので環境変数MYSQL_ALLOW_EMPTY_PASSWORDを設定しています。

MySQLの起動

```
$ docker container run --name mysql --publish 3306:3306 \
  --env MYSQL_ALLOW_EMPTY_PASSWORD=true --detach mysql:8.0
```

MySQLへ接続できることを確認します。

MySQLへの接続

```
$ mysql --host 127.0.0.1 --user root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.40 MySQL Community Server - GPL

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SELECT VERSION();
+-----+
| VERSION() |
+-----+
| 8.0.40    |
+-----+
1 row in set (0.00 sec)
```

TiDB PlaygroundからMySQLへのレプリケーションタスクを作成します。--sink-uriオプションでMySQLへの接続情報を指定します。タスクのIDは--changefeed-idオプションで指定します。指定しない場合は自動で生成されたUUIDが使用されます。

TiCDCでMySQLへのレプリケーションタスク作成

```
$ tiup cdc cli changefeed create --server 127.0.0.1:8300 \
  --sink-uri "mysql://root@127.0.0.1:3306/" \
  --changefeed-id "sample-replication-task"
Create changefeed successfully!
ID: sample-replication-task
Info: {"upstream_id":7449781001574338173,"namespace":"default",...
```

「Create changefeed successfully!」と表示されれば成功です。TiCDCクライアントでタスクの一覧を表示します。

TiCDCでレプリケーションタスクの一覧表示

```
$ tiup cdc cli changefeed list --server 127.0.0.1:8300
Starting component cdc: ...
[
  {
    "id": "sample-replication-task",
    "namespace": "default",
    "summary": {
      "state": "normal",
      "tso": 454699009078722565,
      "checkpoint": "2024-12-18 16:24:26.615",
      "error": null
    }
  }
]
```

TiUPを使用してデモ用データをTiDB Playgroundに投入します。

デモ用データの投入

```
$ tiup demo bookshop prepare --host 127.0.0.1 --port 4000 --user root
INFO[0000] Creating the tables if not existed...      dataset=bookshop
INFO[0000] Creating table books.                      dataset=bookshop
INFO[0000] Creating table users.                     dataset=bookshop
INFO[0000] Creating table authors.                   dataset=bookshop
INFO[0000] Creating table book_authors.              dataset=bookshop
INFO[0000] Creating table orders.                    dataset=bookshop
INFO[0000] Creating table ratings.                   dataset=bookshop
INFO[0000] Finished creating tables!                 dataset=bookshop
INFO[0000] Clearing the old data...                  dataset=bookshop
INFO[0001] Loading users data...                     dataset=bookshop
INFO[0002] Loading books data...                     dataset=bookshop
INFO[0003] Loading authors data...                   dataset=bookshop
INFO[0005] Loading book authors data...              dataset=bookshop
INFO[0006] Loading book orders data...               dataset=bookshop
INFO[0050] Loading book ratings data...              dataset=bookshop
INFO[0094] Finished!                                dataset=bookshop
```

MySQLに接続してbookshopデータベースと各種テーブルが作成されていることを確認します。

MySQLでデータベースとテーブルの確認

```
$ mysql --host 127.0.0.1 --port 3306 --user root
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| bookshop |
| information_schema |
```

```

mysql
performance_schema
sys
+-----+
5 rows in set (0.00 sec)

mysql> USE bookshop;
mysql> SHOW TABLES;
+-----+
| Tables_in_bookshop |
+-----+
| authors             |
| book_authors         |
| books               |
| orders              |
| ratings             |
| users               |
+-----+
6 rows in set (0.00 sec)

mysql> SELECT COUNT(*) FROM books;
+-----+
| COUNT(*) |
+-----+
|    20000 |
+-----+
1 row in set (0.01 sec)

```

データが TiDB Playground から MySQL へ同期されたことが確認できます。

続いて TiDB Playground で bookshop データベースを削除します。

デモ用データの削除

```

$ tiup client

my:root@127.0.0.1:4000=> DROP DATABASE bookshop;
DROP DATABASE

```

MySQL で bookshop データベースが削除されていることを確認します。

MySQL でデータベースの確認

```

$ mysql --host 127.0.0.1 --port 3306 --user root bookshop
ERROR 1049 (42000): Unknown database 'bookshop'

```

このように TiCDC を使用することで TiDB の更新情報を外部システムに連携できます。

TiUP を使用して TiCDC を構築する

TiUP を使用して TiCDC サーバーを構築する場合は、3-3 節で作成方法を説明したトポロジーファイル内に `cdc_servers` を記述します。

TiCDC を含めたトポロジーファイル

```
...
(省略)
...
cdc_servers:
  - host: 10.0.0.51
```

TiCDC サーバーの設定を変更する場合は `cdc_servers` セクションに追加します。たとえば TiCDC のクラスター ID の設定をデフォルトの `default` から `new_cluster` に変更する場合は次のように記述します。

TiCDC の設定変更

```
...
(省略)
...
cdc_servers:
  - host: 10.0.0.51
    ticdc_cluster_id: "new_cluster"
```

その他の設定項目はドキュメントの `TiCDC Server Configurations`^{注15} を参照してください。

`tiup cluster check` コマンドを実行して事前確認後、`tiup cluster deploy` コマンドで構築してください。

すでに構築済みの TiDB に対して TiCDC を追加する場合はスケールアウト用のトポロジーファイルを用意します。次の例は新しい TiCDC サーバーとして 10.0.0.52 を追加する場合のトポロジーファイルです。

scale-out-ticdc.yaml

```
cdc_servers:
  - host: 10.0.0.52
```

`tiup cluster scale-out` コマンドで TiCDC を追加してください。

TiCDC の追加

```
$ tiup cluster scale-out <cluster-name> scale-out-ticdc.yaml
```

TiCDC サーバーを削除またはスケールインする場合は `tiup cluster scale-in` コマンドを使用してください。--node オプションでは削除対象の TiCDC サーバーの IP アドレスとポート番号 8300 を指定します。

注15 <https://docs.pingcap.com/tidb/stable/ticdc-server-config>

Apache Kafka、Apache Pulsarへデータを同期する

TiCDCの同期先はApache KafkaやApache Pulsar^{注19}にも対応しています。Apache Kafkaへの同期を行う場合は`--sink-uri`オプションで`kafka://...`のように指定します。詳細な設定方法はドキュメントのReplicate Data to Kafka^{注20}を参照してください。Apache Pulsarへの同期を行う場合は`--sink-uri`オプションで`pulsar://...`のように指定します。詳細な設定方法はドキュメントのReplicate Data to Pulsar^{注21}を参照してください。

設定ファイルを使用してチェンジフィードを作成する

TiCDCクライアントは、設定ファイルを使用することでより詳細なチェンジフィード設定が可能です。対象のデータベースやテーブルをフィルターする設定や、CSV形式の区切り文字の変更などTOML形式の設定ファイルで定義できます。

設定ファイルは`--config`オプションで指定します。設定ファイルに記述できる内容はドキュメントのCLI and Configuration Parameters of TiCDC Changefeeds^{注22}を参照してください。

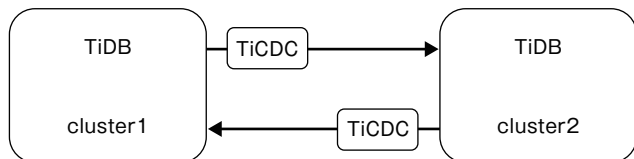
設定ファイルを使用したチェンジフィードの作成

```
$ tiup cdc cli changefeed create --changefeed-id "sample-replication-task-s3" \
--sink-uri "s3://tidb-book-sample/.." --config config.toml
```

双方向レプリケーションを有効にして複数台アクティブなTiDBを構築する

TiCDCは双方向レプリケーションをサポートしています(図6.7)。双方向レプリケーションを使用することで、複数台のTiDBをアクティブな環境として稼働させることができます。双方向レプリケーションを行うためには、チェンジフィードをBDRモード(Bi-Directional Replication mode)で作成する必要があります。

図6.7 TiCDCによる双方向レプリケーション



注19 <https://pulsar.apache.org/>

注20 <https://docs.pingcap.com/tidb/stable/ticdc-sink-to-kafka>

注21 <https://docs.pingcap.com/tidb/stable/ticdc-sink-to-pulsar>

注22 <https://docs.pingcap.com/tidb/stable/ticdc-changefeed-config#changefeed-configuration-parameters>

▶ BDRモードを有効にしたチェンジフィードの作成

BDRモードを有効にしてチェンジフィードを作成するためには、設定でbdr-modeをtrueにする必要があります。デフォルトはfalseです。

動作を確認するために、TiDB Playgroundで2つのTiDBを起動させます。それぞれタグをcluster1、cluster2としておきます。cluster2のTiDBは別端末から起動させてください。また、cluster2ではTiDBとTiCDCが使用するポート番号を変更しておきます。

cluster1の起動

```
$ tiup playground --tag cluster1 --ticdc 1
```

cluster2の起動

```
$ tiup playground --tag cluster2 --ticdc 1 --db.port 4100 --ticdc.port 8400
```

cluster1のチェンジフィードを作成します。bdr-modeをtrueにするために設定ファイルを使用します。

bdr_config.toml

```
bdr-mode = true
```

cluster1でのチェンジフィードの作成

```
$ tiup cdc cli changefeed create --server 127.0.0.1:8300 \  
--changefeed-id "from-cluster1-to-cluster2" \  
--sink-uri "tidb://root@127.0.0.1:4100/" --config bdr_config.toml
```

bdr-modeがtrueになっていることを確認します。

cluster1でのチェンジフィードの確認

```
$ tiup cdc cli changefeed query --server 127.0.0.1:8300 \  
--changefeed-id from-cluster1-to-cluster2 | grep bdr_mode  
"bdr_mode": true,
```

続いてcluster2にチェンジフィードを作成します。

cluster2でのチェンジフィードの作成

```
$ tiup cdc cli changefeed create --server 127.0.0.1:8400 \  
--changefeed-id "from-cluster2-to-cluster1" \  
--sink-uri "tidb://root@127.0.0.1:4000/" --config bdr_config.toml
```

cluster2においてもチェンジフィードの設定のbdr-modeがtrueになっていることを確認します。

cluster2でのチェンジフィードの確認

```
$ tiup cdc cli changefeed query --server 127.0.0.1:8400 \  
--changefeed-id from-cluster2-to-cluster1 | grep bdr_mode  
"bdr_mode": true,
```

これで双方向のデータ同期が可能になりました。

cluster1およびcluster2に接続して動作を確認します。どちらに接続しているか確認しやすいため

にプロンプトを変更して接続します。

cluster1への接続

```
$ mysql --host 127.0.0.1 --port 4000 -u root --prompt "cluster1> "
```

cluster1への接続

```
$ mysql --host 127.0.0.1 --port 4100 -u root --prompt "cluster2> "
```

念のためserver_idを確認して違うTiDBに接続していることを確認します。

cluster1のserver_id

```
cluster1> SHOW STATUS like 'server_id';
+-----+-----+
| Variable_name | Value                                |
+-----+-----+
| server_id     | 61ee4619-52af-420d-b357-3d156ecfe4d6 |
+-----+-----+
1 row in set (0.03 sec)
```

cluster2のserver_id

```
cluster2> SHOW STATUS like 'server_id';
+-----+-----+
| Variable_name | Value                                |
+-----+-----+
| server_id     | 88884be2-6213-4aa6-b8b8-fec3a35fa008 |
+-----+-----+
1 row in set (0.03 sec)
```

▶ BDRロールの操作

双方向レプリケーションが有効時には、特定のDDL(コラム「DDLとDML」参照)の実行を同期させるため、TiDBに対してBDRロールを付与する必要があります。BDRロールはprimaryとsecondaryの2つがあります。primaryロールを持っている場合、次のDDLはチェンジフィードによってsecondaryに同期されます。secondaryロールを持っているTiDBに対しては、次のDDLは直接実行できずにエラーとなります。

- CREATE DATABASE
- CREATE TABLE
- ADD COLUMN(NULLが許容されているか、デフォルト値が設定されているカラム)
- ADD NON-UNIQUE INDEX
- DROP INDEX
- MODIFY COLUMN(デフォルト値かコメントの編集のみ)
- ALTER COLUMN DEFAULT VALUE

- MODIFY TABLE COMMENT
- RENAME INDEX
- ADD TABLE PARTITION
- DROP PRIMARY KEY
- ALTER TABLE INDEX VISIBILITY
- ALTER TABLE TTL
- ALTER TABLE REMOVE TTL
- CREATE VIEW
- DROP VIEW

secondaryロールのTiDBでのDDL実行時のエラー

```
ERROR 8263 (HY000): The operation is not allowed while the bdr role of this cluster is set to secondary.
```

次のDDLはTiDB間で不整合を引き起こす原因となるため、primaryロールを持っていても実行できません。実行するためにはBDRロールをお互いのサーバーから剥奪し、お互いのサーバー上で同じDDLを実行後に再度BDRロールを付与する必要があります。

- DROP DATABASE
- DROP TABLE
- ADD COLUMN(デフォルト値を持っておらず、NULLが許容されないカラム)
- DROP COLUMN
- ADD UNIQUE INDEX
- TRUNCATE TABLE
- MODIFY COLUMN(デフォルト値かコメント以外の編集)
- RENAME TABLE
- DROP PARTITION
- TRUNCATE PARTITION
- ALTER TABLE CHARACTER SET
- ALTER DATABASE CHARACTER SET
- RECOVER TABLE
- ADD PRIMARY KEY
- REBASE AUTO ID
- EXCHANGE PARTITION
- REORGANIZE PARTITION

現在のBDRロールを確認する場合はADMIN SHOW BDR ROLEステートメントを実行します。初期状態は空です。

BDRロールの確認

```
cluster1> ADMIN SHOW BDR ROLE;
+-----+
| BDR_ROLE |
+-----+
|          |
+-----+
1 row in set (0.00 sec)
```

BDRロールを付与する場合はADMIN SET BDR ROLEステートメントを実行します。

BDRロールの付与 primary

```
cluster1> ADMIN SET BDR ROLE PRIMARY;
Query OK, 0 rows affected (0.03 sec)

cluster1> ADMIN SHOW BDR ROLE;
+-----+
| BDR_ROLE |
+-----+
| primary  |
+-----+
1 row in set (0.00 sec)
```

BDRロールの付与 secondary

```
cluster2> ADMIN SET BDR ROLE SECONDARY;
Query OK, 0 rows affected (0.01 sec)

cluster2> ADMIN SHOW BDR ROLE;
+-----+
| BDR_ROLE |
+-----+
| secondary |
+-----+
1 row in set (0.00 sec)
```

BDRロールを剥奪する場合はADMIN UNSET BDR ROLEステートメントを実行します。

BDRロールの剥奪

```
cluster1> ADMIN UNSET BDR ROLE;
Query OK, 0 rows affected (0.01 sec)

cluster2> ADMIN SHOW BDR ROLE;
+-----+
| BDR_ROLE |
+-----+
|          |
+-----+
1 row in set (0.00 sec)
```

DDLとDML

column

DDLとはデータ定義言語(Data Definition Language)の略称で、データベースの構造を定義するための言語です。たとえば、テーブルの作成や削除、カラムの追加や削除などが該当します。

一方でデータベースのデータを操作するための言語はDMLと呼ばれます。DMLはデータ操作言語(Data Manipulation Language)の略称で、データの挿入、更新、削除などがあります。

 データ同期の確認

primary ロールを付与されている cluster1 でデータベースおよびテーブルを作成します。

cluster1でのデータベースおよびテーブルの作成

```
cluster1> CREATE DATABASE bdr_test;
Query OK, 1 row affected (0.14 sec)

cluster1> USE bdr_test;
Database changed

cluster1> CREATE TABLE sample (id INT PRIMARY KEY, text VARCHAR(255));
Query OK, 0 rows affected (0.12 sec)
```

secondary ロールを付与されている cluster2 でデータベースおよびテーブルが同期されていることを確認します。

cluster2でのデータベースおよびテーブルの確認

```
cluster2> SHOW DATABASES;
+-----+
| Database |
+-----+
| INFORMATION_SCHEMA |
| METRICS_SCHEMA |
| PERFORMANCE_SCHEMA |
| bdr_test |
| mysql |
| sys |
| test |
+-----+
7 rows in set (0.00 sec)

cluster2> USE bdr_test;
Database changed

cluster2> SHOW TABLES;
+-----+
| Tables_in_bdr_test |
+-----+
```

```
+-----+
| sample |
+-----+
1 row in set (0.00 sec)
```

cluster1でデータを挿入します。

```
cluster1でのデータの挿入
cluster1> INSERT INTO bdr_test.sample VALUES (1, "I am cluster1");
Query OK, 1 row affected (0.01 sec)
```

cluster2でデータが同期されていることを確認します。

```
cluster2でのデータの確認
cluster2> SELECT * FROM bdr_test.sample;
+----+-----+
| id | text      |
+----+-----+
| 1  | I am cluster1 |
+----+-----+
1 row in set (0.01 sec)
```

cluster2でもデータを挿入します。

```
cluster2でのデータの挿入
cluster2> INSERT INTO bdr_test.sample VALUES (2, "I am cluster2");
Query OK, 1 row affected (0.01 sec)
```

cluster1でデータが同期されていることを確認します。

```
cluster1でのデータの確認
cluster1> SELECT * FROM bdr_test.sample;
+----+-----+
| id | text      |
+----+-----+
| 1  | I am cluster1 |
| 2  | I am cluster2 |
+----+-----+
2 rows in set (0.00 sec)
```

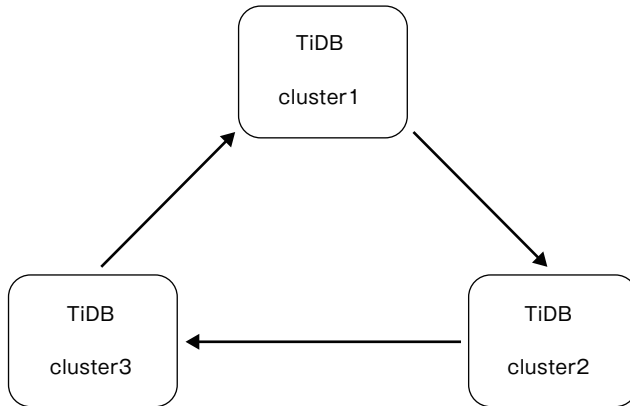
このようにBDRモードを有効にしたチェンジフィードを使用することで、TiDBを2台ともアクティブな環境で稼働させることができます。

▶ 3台環境でのBDRモード

3台構成の環境でBDRモードを使用する場合、図6.8のように循環構造を作成することはサポートされていませんので注意してください(TiCDCは図から省略しています)。この構成では1台に障害が発生

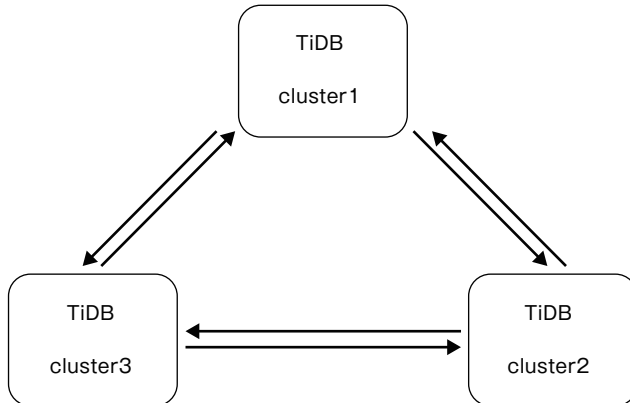
した場合にすべての同期が停止してしまいます。

図6.8 サポートされないBDRモード



3台構成の環境でBDRモードを使用する場合は、**図6.9**のようにお互い同期し合う構成にしてください(TiCDCは図から省略しています)。

図6.9 3台環境のBDRモード



TiDB Cloudでチェンジフィードを作成する

TiDB Cloudにおいても、チェンジフィードを作成して外部にデータを同期できます。同期先は執筆時点ではTiDB Cloud、MySQL、Apache Kafka、パブリッククラウドのオブジェクトストレージをサポートしています。また、TiDB Cloud Serverlessはチェンジフィードを作成できません。TiDB Cloud Dedicatedのみチェンジフィードを作成できます。

TiDB Cloudでのチェンジフィードの使用はReplication Capacity Units(RCU)単位で課金されます。RCUが増えれば増えるほどデータ同期処理のスループットが向上します。料金の詳細はドキュメントの

図6.10 チェンジフィードの設定確認

Create Changefeed

Configure Destination ☒ Configure Replication ☒ Configure Specification ☒ **4 Review**

Replication Destination

Destination: Amazon S3

S3 URI: s3://tidb-book-sample/tidb-from-cloud/

Replication Settings

Table Filter: *

Start Position: FROM NOW

Data Format: CSV

Date Separator: None

Binary Encoding Method: Base64

Delimiter: ,

Quote: "

Null: \N

Include Commit Ts: False

TiDB Extension: Disabled

Specification and Name

Specification: 2 RCUs

Name: changefeed-1

\$0.3338/hour

Cancel Previous **Submit**

Changefeed Cost^{注23}を参照してください。

チェンジフィードの作成はクラスター概要画面の「Changefeed」から行います。画面右上にある「Create Changefeed」をクリックすると作成画面が表示されます。データの同期先の設定や対象のテーブルを指定をした後、設定内容の確認画面が表示されますので問題無ければ「Submit」を選択します(図6.10)。

作成が完了するとチェンジフィード一覧画面上でStatusがRunningになります。作成したチェンジフィードの名前をクリックすることで詳細画面が表示されます。詳細画面ではチェンジフィードの設定内容やメトリクスを確認できます。

作成したチェンジフィードはRCUのスケールアップおよびスケールダウンが可能です。チェンジフィード一覧画面よりActionの3点リーダーをクリック後「Scale Up/Down」をクリックして変更します(図6.11)。同様に停止、再開、削除の操作もActionから可能です。

図6.11 チェンジフィードのAction操作

Name	Destination	Status	Checkpoint Time	Create Time	Action
changefeed-1	Amazon S3	Running	2024-12-22 05:43:32	2024-12-22 05:25:59	...

Changefeed Actions:

- View
- Scale Up/Down
- Pause
- Resume
- Edit
- Delete

注23 <https://www.pingcap.com/tidb-dedicated-pricing-details/#changefeed-cost>

Kubernetes環境へTiCDCをデプロイする

Kubernetes環境へTiCDCをデプロイする場合は、TiDB Operatorを使用します。TidbClusterリソースの.spec.ticdcにTiCDCの設定を記述します。

TiDB OperatorによるTiCDCのデプロイ

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: tidb-sample
spec:
  ...
  (省略)
  ...
  ticdc:
    replicas: 1
```

TiCDCは、たとえばデータ領域とログ領域で別のディスクを使用するといったような場合に、複数のディスクを使用するように定義できます。その場合は次のように.spec.ticdc.storageVolumesにディスクの設定を記述します。

TiCDCで複数のディスクを使用

```
...
(省略)
...
spec:
  ticdc:
    config:
      dataDir: /ticdc/data
      logFile: /ticdc/log/cdc.log
    storageVolumes:
      - name: data
        storageSize: "10Gi"
        storageClassName: local-storage
        mountPath: "/ticdc/data"
      - name: log
        storageSize: "10Gi"
        storageClassName: local-storage
        mountPath: "/ticdc/log"
```

TidbClusterリソースを適用することでTiCDCのStatefulSetリソースが作成されます。

```
$ kubectl get statefulset -l app.kubernetes.io/component=ticdc
NAME                READY   AGE
tidb-sample-ticdc   1/1     14m
```

TiCDCのエンドポイントにアクセスするためのServiceリソースはTiDB Operatorによって作成され

ません。そのため、TiCDC クライアントの操作はTiCDC のPod 内から実行するか、手動でService リソースを作成してください。もしくは、TiCDC のPod がhostNetwork^{注24}を使用するように定義できますので、必要に応じて設定してください。

TiCDCでhostNetworkを使用

```
...
(省略)
...
spec:
  ticdc:
    hostNetwork: true
```

TiCDC のPod 内でTiCDC クライアントを実行するには次のようにします。Pod 内のシェルを起動して実行するか、直接cdc コマンドを実行します。TiDB Operator によって作成されたTiCDC のPod は、エンドポイントとして8301 番ポートを使用していますのでcdc コマンド実行の際は--server オプションを使用してhttp://127.0.0.1:8301を指定してください。

TiCDCのPod内でシェルを起動

```
$ kubectl exec -ti tidb-sample-ticdc-0 -- /bin/sh
sh-5.1#

sh-5.1# /cdc cli capture list --server http://127.0.0.1:8301
[
  {
    "id": "5e5b48cd-d554-4812-9269-288e30e1fc32",
    "is-owner": true,
    "address": "tidb-sample-ticdc-0.tidb-sample-ticdc-peer.default.svc:8301",
    "cluster-id": "default"
  }
]
```

TiCDCのPod内でcdcコマンドを実行

```
$ kubectl exec -ti tidb-sample-ticdc-0 -- /cdc cli capture list --server http://127.0.0.1:8301
```

チェンジフィードの作成は次のように実行します。Amazon S3 への同期の例を示します。

TiCDCのPod内でチェンジフィードの作成

```
$ kubectl exec -ti tidb-sample-ticdc-0 -- \
/cdc cli changefeed create --changefeed-id "sample-replication-task-s3" \
--server http://127.0.0.1:8301 \
--sink-uri "s3://tidb-book-sample/ticdc?protocol=csv&access-key=xxx&secret-access-key=xxx"
Create changefeed successfully!
ID: sample-replication-task-from-k8s
Info: {"upstream_id":7450960009952210070,"namespace":"default", ...}
```

注24 <https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/#pod-s-dns-policy>

```
$ kubectl exec -ti tidb-sample-tidc-0 \
-- /cdc cli changefeed list --server 127.0.0.1:8301
[
  {
    "id": "sample-replication-task-from-k8s",
    "namespace": "default",
    "summary": {
      "state": "normal",
      "tso": 454771872600424451,
      "checkpoint": "2024-12-21 21:36:58.892",
      "error": null
    }
  }
]
```

6

6-3

DMによるデータ移行

TiDBはDMと呼ばれるツールによるデータ移行をサポートしています。MySQL、Percona MySQL、MariaDB、Amazon RDS for MySQL、Amazon AuroraなどのMySQL互換データベースからTiDBへの完全および増分データ移行をサポートします。

DMクラスターはDMマスター、DMワーカーで構成されるクラスターです。DMマスターは移行タスクのメタデータを管理し、DMワーカーにタスクをスケジュールします。DMワーカーは上流および下流の移行タスクを実行します。DMワーカーはステートレスで、最大1,000ノードまで構成できます。

DMクラスターはdmctlコマンドを使用して操作します。データ移行タスクの作成、更新、または削除、状態の確認などの操作が可能です。

本節ではMySQLからTiDB Playgroundへデータを移行し、DMの基本的な使い方を説明します。

dmctlをインストールする

TiUPを使用してdmctlコマンドをインストールします。

TiUPによるdmctlのインストール

```
$ tiup install dmctl
```

dmctlの実行はtiup dmctlコマンドで行います。

TiUPからdmctlコマンドを使用する

```
$ tiup dmctl --version
Starting component dmctl: ...
Release Version: v8.5.0
Git Commit Hash: 3703b18c6da785d57236a1452eed43759e51b411
Git Branch: HEAD
UTC Build Time: 2024-12-17 04:34:05
Go Version: go1.23.3
Failpoint Build: false
```

インストール可能なdmctlコマンドのバージョンはtiup list dmctlコマンドで確認できます。

dmctlのバージョンリスト確認

```
$ tiup list dmctl
Available versions for dmctl:
Version                Installed  Release      ...
-----                -
nightly -> v8.5.0-alpha-nightly      2024-12-21T05:08:00Z ...
...
(省略)
...
v8.5.0-alpha-nightly      2024-12-21T05:08:00Z ...
v8.5.0                    YES       2024-12-19T06:28:26Z ...
```

MySQLを起動して移行用データを用意する

Dockerを使用してMySQLを起動し、TiDBへ移行するデータを用意します。移行用のデータはTiUPクライアントを使用して投入します。検証目的のため、起動するMySQLには環境変数MYSQL_ALLOW_EMPTY_PASSWORDを設定しています。

MySQLの起動

```
$ docker container run --name mysql --publish 3306:3306 \
--env MYSQL_ALLOW_EMPTY_PASSWORD=true --detach mysql:8.0
```

MySQLへ接続できることを確認します。

MySQLへの接続

```
$ mysql --host 127.0.0.1 --user root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.40 MySQL Community Server - GPL

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
```

```

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SELECT VERSION();
+-----+
| VERSION() |
+-----+
| 8.0.40    |
+-----+
1 row in set (0.00 sec)

```

移行用データをTiUPを使用してMySQLに投入します。

移行用データの投入

```

$ tiup demo bookshop prepare --host 127.0.0.1 --port 3306 --user root
Starting component demo: ...
INFO[0000] Creating the tables if not existed...      dataset=bookshop
INFO[0000] Creating table books.                     dataset=bookshop
INFO[0000] Creating table users.                     dataset=bookshop
INFO[0000] Creating table authors.                   dataset=bookshop
INFO[0000] Creating table book_authors.               dataset=bookshop
INFO[0000] Creating table orders.                     dataset=bookshop
INFO[0000] Creating table ratings.                   dataset=bookshop
INFO[0000] Finished creating tables!                  dataset=bookshop
INFO[0000] Clearing the old data...                   dataset=bookshop
INFO[0000] Loading users data...                     dataset=bookshop
INFO[0000] Loading books data...                     dataset=bookshop
INFO[0001] Loading authors data...                   dataset=bookshop
INFO[0001] Loading book authors data...               dataset=bookshop
INFO[0001] Loading book orders data...               dataset=bookshop
INFO[0010] Loading book ratings data...              dataset=bookshop
INFO[0019] Finished!                                dataset=bookshop

```

TiDB PlaygroundでDMクラスターを作成する

TiUPを使用してTiDB PlaygroundにDMクラスターを作成します。DMマスターおよびDMワーカーの台数をそれぞれ起動時に指定します。本節では動作確認のため1台ずつ起動します。DMマスターの台数は--dm-masterオプション、DMワーカーの台数は--dm-workerオプションで指定します。デフォルトはそれぞれ0になっています。

DMクラスターの作成

```

$ tiup playground --dm-master 1 --dm-worker 1
🔥 TiDB Playground Cluster is started, enjoy!

```

```
Connect TiDB:  mysql --comments --host 127.0.0.1 --port 4000 -u root
Connect DM:     tiup dmctl --master-addr 127.0.0.1:8261
TiDB Dashboard: http://127.0.0.1:2379/dashboard
Grafana:        http://127.0.0.1:3000
```

TiDB PlaygroundでDM環境を構築すると「Connect DM:」が表示され、DMクラスターに接続するためのdmctlコマンドが表示されます。tiup dmctl list-memberコマンドでDMクラスターの構成を確認します。

DMクラスターの構成確認

```
$ tiup dmctl --master-addr 127.0.0.1:8261 list-member
Starting component dmctl: ...
{
  "result": true,
  "msg": "",
  "members": [
    {
      "leader": {
        "msg": "",
        "name": "dm-master-0",
        "addr": "127.0.0.1:8261"
      }
    },
    {
      "master": {
        "msg": "",
        "masters": [
          {
            "name": "dm-master-0",
            "memberID": "11007177379717700053",
            "alive": true,
            "peerURLs": [
              "http://127.0.0.1:8291"
            ],
            "clientURLs": [
              "http://127.0.0.1:8261"
            ]
          }
        ]
      }
    },
    {
      "worker": {
        "msg": "",
        "workers": [
```

```

    {
      "name": "dm-worker-0",
      "addr": "127.0.0.1:8262",
      "stage": "free",
      "source": ""
    }
  ]
}
]
}

```

DM マスターとして dm-master-0、DM ワーカーとして dm-worker-0 が起動していて、dm-master-0 がリーダーとして選出されていることが確認できます。なお、リーダーは DM マスターから選出されます。リーダーのみ表示する場合は `--leader` オプション、DM マスターのみ表示する場合は `--master` オプション、DM ワーカーのみ表示する場合は `--worker` オプションが使用でき、組み合わせて表示させることも可能になっています。

DM クラスターの個別構成確認

```

$ tiup dmctl --master-addr 127.0.0.1:8261 list-member --leader
$ tiup dmctl --master-addr 127.0.0.1:8261 list-member --master
$ tiup dmctl --master-addr 127.0.0.1:8261 list-member --worker
$ tiup dmctl --master-addr 127.0.0.1:8261 list-member --master --worker

```

DM クラスターにデータソースを追加する

移行元のデータベースをデータソースとして追加します。データソースの追加は `tiup dmctl operate-source create` コマンドを使用します。データソースの設定は YAML ファイルで指定します。source-id はデータソースの識別子です。from には移行元のデータベースの接続情報を指定します。from.password を記述する際は、`tiup dmctl encrypt` コマンドを使用して暗号化した値を指定することが推奨されています。

data_source_mysql.yaml

```

source-id: "docker-mysql"

from:
  host: "127.0.0.1"
  port: 3306
  user: "root"
  password: ""

```

データソースの設定を読み込ませて DM クラスターにデータソースを追加します。

データソースの追加

```
$ tiup dmctl --master-addr 127.0.0.1:8261 operate-source create ./data_source_mysql.yaml
Starting component dmctl: ...
{
  "result": true,
  "msg": "",
  "sources": [
    {
      "result": true,
      "msg": "",
      "source": "docker-mysql",
      "worker": "dm-worker-0"
    }
  ]
}
```

なお、`--print-sample-config` オプションを使用して、データソースの設定のテンプレート表示もできます。テンプレートを元にデータソースの設定を作成できますので参考にしてください。

データソースの設定テンプレートの表示

```
$ tiup dmctl --master-addr 127.0.0.1:8261 operate-source --print-sample-config
Starting component dmctl: ...

#server id of slave for binlog replication
#each instance (master and slave) in replication group should have different server id
server-id: 101

#represents a MySQL/MariaDB instance or a replication group
source-id: mysql-replica-01

#flavor: mysql/mariadb
flavor: mysql

#enable relay log
enable-relay: false
# relay-binlog-name: ''
# relay-binlog-gtid: ''
# relay-dir: ./relay_log

#enable gtid in relay log unit
enable-gtid: false

# when this field is true, all tasks on the current source will be started, and vice versa
enable: true

#charset of DSN of source mysql/mariadb instance
# charset: ''
```



```

from:
  host: 127.0.0.1
  user: root
  password: '123456'
  port: 3306

#relay log purge strategy
#purge:
# interval: 3600
# expires: 24
# remain-space: 15

#task status checker
#checker:
# check-enable: true
# backoff-rollback: 5m
# backoff-max: 5m

```

▶ TiDB PlaygroundのDMクラスターでパスワードの暗号化を使用する場合

tiup dmctl encrypt コマンドを使用してパスワードを暗号化するためには、DM マスターの設定で secret-key-path を設定しなければなりません。TiDB Playground では secret-key-path はデフォルトで設定されていないので、別途設定する必要があります。secret-key-path の設定は次の手順で行います。

まず始めに secret-key-path に設定するシークレットファイルを作成します。任意の文字列を sha256sum コマンドによってハッシュ化してファイルに保存します。

シークレットの作成

```
$ head -n 256 /dev/urandom | sha256sum | awk '{print $1}' > /path/to/key
```

次に DM マスターノード用の設定ファイルを用意し、secret-key-path を設定します。

dm_master.config

```
secret-key-path = "/path/to/key"
```

最後に設定ファイルを読み込ませて TiDB Playground を起動します。

dm_master.config を指定して TiDB Playground を起動

```
$ tiup playground --dm-master 1 --dm-master-config /path/to/dm_master.config \
--dm-worker 1
```

起動したら暗号化が使用できるようになります。tiup dmctl encrypt コマンドによって表示された文字列をデータソース設定ファイルの .from.password に記述してください。

パスワードの暗号化

```
$ tiup dmctl encrypt 'mypassword' --master-addr 127.0.0.1:8261
Starting component dmctl: ...
+f0V/hRWyqipIUhxBdVQdEAc6Ydk6J7sJJ8G
```

DMクラスターに移行タスクを作成する

DMクラスターに移行タスクを作成します、タスク構成ファイルを準備し、`tiup dmctl start-task` コマンドを使用してタスクを作成します。

sample_dm_task.yaml

```
name: sample
task-mode: all

target-database:
  host: "127.0.0.1"
  port: 4000
  user: "root"
  password: ""

mysql-instances:
- source-id: "docker-mysql"
  block-allow-list: "ba-rule1"

block-allow-list:
  ba-rule1:
    do-dbs: ["bookshop"]
```

`mysql-instances` の設定で先ほど作成したデータソースの `docker-mysql` を指定しています。また、MySQL の `bookshop` データベースのみを移行対象にするために `block-allow-list` を使用しています。`block-allow-list` は移行対象のデータベースやテーブルを指定するための設定です。`block-allow-list.<ルール名>.do-dbs` に対象データベースを指定します。テーブルを指定する場合は `do-tables` を使用します。対象から除外したい場合は `ignore-dbs` および `ignore-tables` を使用します。

移行タスクの作成

```
$ tiup dmctl --master-addr 127.0.0.1:8261 start-task ./sample_dm_task.yaml
Starting component dmctl: ...
{
  "result": true,
  "msg": "",
  "sources": [
    {
      "result": true,
      "msg": "",
```

```

        "source": "docker-mysql",
        "worker": "dm-worker-0"
    }
],
"checkResult": "pre-check is passed. "
}

```

移行タスクを作成したらステータスを確認します。

移行タスクのステータス確認

```

$ tiup dmctl --master-addr 127.0.0.1:8261 query-status
Starting component dmctl: ...
{
  "result": true,
  "msg": "",
  "tasks": [
    {
      "taskName": "sample",
      "taskStatus": "Running",
      "sources": [
        "docker-mysql"
      ]
    }
  ]
}

```

taskStatusがRunningになっていることを確認します。

データの移行を確認する

移行タスクが正常に動作していることを確認します。TiDB Playgroundに接続してデータが移行されているか確認します。

データの確認

```

$ tiup client
Starting component client: ...
Connected with driver mysql (8.0.11-TiDB-v8.5.0)
Type "help" for help.

my:root@127.0.0.1:4000=> SHOW DATABASES;
      Database
-----
INFORMATION_SCHEMA
METRICS_SCHEMA
PERFORMANCE_SCHEMA
bookshop

```

```
dm_meta
lightning_task_info
mysql
sys
test
(9 rows)
```

移行対象として設定したbookshopデータベースが表示されていることを確認します。dm_metaおよびlightning_task_infoはDMクラスターのメタデータを管理するためのデータベースです。

データベース内のテーブルの件数を確認します。

```
----- テーブルの確認 -----
my:root@127.0.0.1:4000=> USE bookshop;
USE
my:root@127.0.0.1:4000=> SELECT COUNT(*) FROM books;
count(*)
-----
      20000
(1 row)
```

移行元との差分のチェックはsync-diff-inspectorを使用して確認できます。sync-diff-inspectorについては次節で説明します。

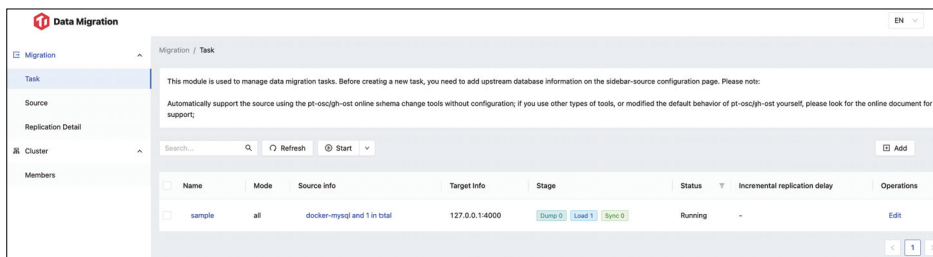
Webコンソールを使用してDMクラスターを操作する

DMクラスターはWebコンソールを使用して操作、確認できます。Webコンソールを有効にするにはDMマスターの設定でopenapiを有効にします。TiDB Playgroundで有効にする場合は設定ファイル内に記述して起動時に読み込ませてください。

```
dm_master.config
openapi = true
```

<http://<DMマスターノード>:8261/dashboard/>にアクセスすることでデータソースの登録や移行タスクの作成が可能です(図6.12)。

図6.12 DMクラスターのWebコンソール



なお、Web コンソールは執筆時点では実験的な機能となっています。

TiDB Operatorを使用したDMクラスターの構築

TiDB Operatorを使用して Kubernetes 上に DM クラスターを構築する場合は DMcluster リソースを使用します(リスト6.7)。

.spec.master.replicas および .spec.worker.replicas で、DM マスターおよび DM ワーカーの台数を指定します(リスト6.7の❶と❷)。`.spec.master.service.type`でDMマスターのServiceリソースのタイプを指定します(リスト6.7の❸)。デフォルトはClusterIPですが、NodePortやLoadBalancerを指定可能です。DMマスターのServiceリソースはdmctlコマンドのエンドポイントに指定して使用します。Kubernetes クラスター外からアクセスする場合はNodePortまたはLoadBalancerを使用してエンドポイントを外部へ公開してください。本項ではNodePortを使用します。

リスト6.7 DMclusterリソースの例

```
dm_cluster_sample.yaml
apiVersion: pingcap.com/v1alpha1
kind: DMcluster
metadata:
  name: dm-sample
spec:
  version: v8.5.0
  master:
    baseImage: pingcap/dm
    replicas: 1 —❶
    storageSize: "10Gi"
    service:
      type: NodePort —❸
  worker:
    baseImage: pingcap/dm
    replicas: 1 —❷
    storageSize: "10Gi"
```

kubectl apply コマンドで DMcluster リソースを適用することで DM クラスターが作成されます。

DMクラスターの作成

```
$ kubectl apply -f dm_cluster_sample.yaml
dmcluster.pingcap.com/dm-sample created
```

kubectl get dmcluster コマンドで DM クラスターのステータスを確認します。しばらくすると READY が True になります。

DMクラスターのステータス確認

```
$ kubectl get dmcluster
NAME      READY  MASTER  ...
dm-sample  True   pingcap/dm:v8.5.0  ...
```

DMcluster リソースが作成されると、TiDB Operator によって DM マスターと DM ワーカーの StatefulSet リソースが作成されます。

```
$ kubectl get statefulset -l app.kubernetes.io/component=dm-master
NAME                READY  AGE
dm-sample-dm-master  1/1    11m

$ kubectl get statefulset -l app.kubernetes.io/component=dm-worker
NAME                READY  AGE
dm-sample-dm-worker  1/1    10m
```

DM マスターのエンドポイントを Kubernetes クラスター外に対して公開していない場合は、Kubernetes 上の Pod 内から `dmctl` コマンドを実行してください。Pod 内からは `<metadata.name>-dm-master` というサービス名を `--master-addr` オプションに指定して実行します。ポート番号は 8261 を指定します。

Pod内からdmctlコマンドを実行

```
$ kubectl exec -ti dm-sample-dm-master-0 -- /dmctl --master-addr dm-sample-dm-master:8261 list-member
{
  "result": true,
  "msg": "",
  "members": [
    {
      "leader": {
        "msg": "",
        "name": "dm-sample-dm-master-0",
        "addr": "dm-sample-dm-master-0.dm-sample-dm-master-peer:8261"
      }
    },
    {
      "master": {
        "msg": "",
        "masters": [
          {
            "name": "dm-sample-dm-master-0",
            "memberID": "4368510456757332941",
            "alive": true,
            "peerURLs": [
              "http://dm-sample-dm-master-0.dm-sample-dm-master-peer:8291"
            ],
            "clientURLs": [
              "http://dm-sample-dm-master-0.dm-sample-dm-master-peer:8261"
            ]
          }
        ]
      }
    }
  ]
}
```

```
]
    }
  ]
}
},
{
  "worker": {
    "msg": "",
    "workers": [
      {
        "name": "dm-sample-dm-worker-0",
        "addr": "dm-sample-dm-worker-0.dm-sample-dm-worker-peer:8262",
        "stage": "free",
        "source": ""
      }
    ]
  }
}
```

DM マスターで Web コンソールを使用する場合は DMCluster リソース内の `.spec.master.config` で `openapi` を有効にします(リスト 6.8 の❶)。`.spec.master.config` を変更しても Pod は再作成されないため、変更する際は一度 DMCluster リソースを削除してから再作成してください。

リスト6.8 DMClusterリソースの例 (Webコンソールの有効化)

```
dm_cluster_sample.yaml
apiVersion: pingcap.com/v1alpha1
kind: DMCluster
metadata:
  name: dm-sample
spec:
  version: v8.5.0
  master:
    baseImage: pingcap/dm
    replicas: 1
    storageSize: "10Gi"
    service:
      type: NodePort
    config: | ——— ❶
      openapi = true
  worker:
```

`.spec.master.service.type`でNodePortやLoadBalancerを指定してKubernetesクラスター外からの

アクセスを公開している場合は、Service リソースを確認して DM マスターノードの Web コンソールにアクセスしてください。

DM マスターの Service リソースの確認

```
$ kubectl get svc \
--selector app.kubernetes.io/component=dm-master,app.kubernetes.io/used-by=end-user
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
dm-sample-dm-master	NodePort	10.96.44.105	<none>	8261:32251/TCP	4m58s

Kubernetes クラスター外へアクセスを公開していない場合は、`kubectl port-forward` コマンドを使用して接続してください。コマンド実行後にブラウザで `http://127.0.0.1:8261/dashboard/` にアクセスしてください。

DM マスターへポートフォワーディング

```
$ kubectl port-forward svc/dm-sample-dm-master 8261:8261
```

6-4

sync-diff-inspectorによるデータ同期の差分確認

`sync-diff-inspector` はデータベースに保存されているデータの差分を確認するためのツールです。MySQL と MySQL、TiDB と TiDB、MySQL と TiDB のデータを比較できます。また、データが不一致であるシナリオの場合は、少量であればデータの修復が可能です。

sync-diff-inspector と sync_diff_inspector

column

本節では用語として `sync-diff-inspector` と `sync_diff_inspector` が記載されていますが、混同しやすいためあらためて用語の説明をします。

ハイフンを使用している `sync-diff-inspector` はツールとしての名称になります。一方アンダースコアを使用している `sync_diff_inspector` はコマンドラインツールのバイナリーの名前になります。文章内では `sync_diff_inspector` とコード用のフォントを使用します。

sync_diff_inspector コマンドをインストールする

`sync-diff-inspector` はコマンドラインツールとして `sync_diff_inspector` が用意されており、TiDB Community Toolkit に含まれています。GitHub のリリースページ^{注25}からバイナリーをダウンロードするか、次のリンクからダウンロードしてください。

注25 <https://github.com/pingcap/tidb-tools/releases>

TiDB Community Toolkitのダウンロードリンク

```
https://download.pingcap.org/tidb-community-toolkit-v{version}-linux-{arch}.tar.gz
```

リンク内の{version}はTiDBのバージョン番号を示し、{arch}はシステムのアーキテクチャーであるamd64またはarm64を示します。TiDBのバージョン8.5.0を指定して、amd64のLinux環境へダウンロードする場合は次のようになります。

sync_diff_inspectorコマンドのダウンロードとインストール

```
$ curl -O https://download.pingcap.org/tidb-community-toolkit-v8.5.0-linux-amd64.tar.gz
$ tar xzf tidb-community-toolkit-v8.5.0-linux-amd64.tar.gz
$ install --target-directory /path/to tidb-community-toolkit-v8.5.0-linux-amd64/sync_diff_inspector
```

Dockerが使用できる環境であれば、Dockerイメージを使用して実行できます。

Dockerイメージを使用して実行

```
$ docker container run --rm -ti pingcap/tidb-tools:master /sync_diff_inspector --version
App Name: sync_diff_inspector
Release Version: v8.5.0-1-g1cb50a4
Git Commit Hash: 1cb50a457a5f5267e9273bd98ddf79607534cf4e
Git Branch: HEAD
UTC Build Time: 2024-12-19 10:58:17
Go Version: go1.23.1
```

pingcap/tidb-tools:latestのイメージは長い間更新されていないので使用しないようにしてください。また、Dockerイメージでは特定のバージョンによるイメージタグが付与されていないので、使用する際はmasterタグもしくはnightlyタグを指定してください。特定のバージョンで実行する必要がある場合はバイナリーをダウンロードすることをおすすめします。

sync_diff_inspectorコマンドを使用してデータの差分を確認する

MySQLとTiDB Playgroundを使用してsync_diff_inspectorコマンドの動作確認を行います。MySQLとTiDB Playground両方にTiUPを使用してデモ用データを投入します。MySQLは環境変数MYSQL_ALLOW_EMPTY_PASSWORDを設定しておきます。

MySQLの起動

```
$ docker container run --name mysql --publish 3306:3306 \
--env MYSQL_ALLOW_EMPTY_PASSWORD=true --detach mysql:8.0
```

TiDB Playgroundの起動

```
$ tiup playground
```

TiUPを使用してデモ用データをMySQLとTiDB Playgroundに投入します。authorsテーブルは一致しており、booksテーブルに差分があるシナリオを想定します。tiup demo bookshop prepare コマンド

を使用しますが、データは毎回ランダムに生成されるためデフォルトのデータ量だと差分が大きくなってしまいますので、books テーブルのデータの件数を少なくしておきます。

MySQLヘデモ用データの投入

```
$ tiup demo bookshop prepare --host 127.0.0.1 --port 3306 --user root --books 10
```

TiDB Playgroundヘデモ用データの投入

```
$ tiup demo bookshop prepare --host 127.0.0.1 --port 4000 --user root --books 10
```

また、authors テーブルに関しては一度TRUNCATE し、MySQLとTiDBで同じデータを挿入しておきます。

authorsテーブルのデータ挿入

```
mysql> USE bookshop;
mysql> TRUNCATE authors;

mysql> INSERT INTO authors VALUES(1133005406, 'Rosanna Dooley', 1, 1980, 1988);
mysql> INSERT INTO authors VALUES(1248809090, 'Cynthia Schimmel', 0, 1950, 1963);
mysql> INSERT INTO authors VALUES(1394068284, 'Dayana Wunsch', 0, 1970, 1986);
mysql> INSERT INTO authors VALUES(2042998664, 'Sandy Gorczany', 1, 1993, 1995);
mysql> INSERT INTO authors VALUES(2408423491, 'Matilda Romaguera', 1, 1968, 1970);
mysql> INSERT INTO authors VALUES(2698209206, 'Leda Marvin', 1, 1982, NULL);
mysql> INSERT INTO authors VALUES(3141157734, 'Chyna Goyette', 0, 1944, 1994);
mysql> INSERT INTO authors VALUES(3620803039, 'Caterina Ondricka', 0, 1936, 1982);
mysql> INSERT INTO authors VALUES(3754010759, 'Ashleigh Eichmann', 0, 1958, 1978);
mysql> INSERT INTO authors VALUES(4288605205, 'Tess Mitchell', 1, 1964, 2007);
```

sync_diff_inspector コマンドの設定ファイルを作成します(リスト6.9)。export-fix-sqlをtrueにすることで、差分を修正するためのSQLファイルを出力します(リスト6.9の❶)。data-sourcesで比較対象となるデータベースの接続情報を指定します(リスト6.9の❷)。routesは比較元と先のテーブルのマッピングを指定します(リスト6.9の❸)。taskではアウトプットディレクトリの指定や対象のデータベースを指定します(リスト6.9の❹)。routesはオプションとなり、省略可能です。設定ファイルの詳細な記述内容はドキュメントのsync-diff-inspector User Guide^{注26}を参照してください。

リスト6.9 sync_diff_inspectorコマンドの設定ファイル

config.toml

```
export-fix-sql = true —❶
```

```
[data-sources] —❷
```

```
[data-sources.mysql]
```

```
host = "127.0.0.1"
```

```
port = 3306
```

注26 <https://docs.pingcap.com/tidb/stable/sync-diff-inspector-overview#configuration-file-description>

```

user = "root"
password = ""
route-rules = ["bookshop-rule"]

[data-sources.tidb]
host = "127.0.0.1"
port = 4000
user = "root"
password = ""

[routes] —— ❸
[routes.bookshop-rule]
schema-pattern = "bookshop"
target-schema = "bookshop"

[task] —— ❹
output-dir = "./output"
source-instances = ["mysql"]
target-instance = "tidb"
target-check-tables = ["bookshop.authors", "bookshop.books"]

```

sync_diff_inspector コマンドを実行します。--config オプションで設定ファイルを指定します。

```

sync_diff_inspectorの実行
$ sync_diff_inspector --config config.toml
A total of 2 tables need to be compared

Comparing the table structure of `bookshop`.`authors` ... equivalent
Comparing the table structure of `bookshop`.`books` ... equivalent
Comparing the table data of `bookshop`.`authors` ... equivalent
Comparing the table data of `bookshop`.`books` ... failure

-----
Progress [=====>] 100% 0/0
The data of `bookshop`.`books` is not equal

The rest of tables are all equal.

A total of 2 tables have been compared, 1 tables finished, 1 tables failed, 0 tables skipped.
The patch file has been generated in
'output/fix-on-tidb/'
You can view the comparison details through './output/sync_diff.log'

```

「The data of `bookshop`.`books` is not equal」と表示され、books テーブルに差分があることが確認できました。

アウトプットディレクトリには下記のファイルが出力されています。

修正するためのSQLファイルの確認

```
$ tree output
output
├── checkpoint
│   └── f5af502a71130b8316a1f6282e0a61b92db8b4babade1c7cc268528f3a1f5f36
├── fix-on-tidb
│   └── bookshop:books:0:0-0:0.sql
├── summary.txt
└── sync_diff.log

2 directories, 4 files
```

fix-on-tidbディレクトリには差分を修正するためのSQLファイルが出力されています。設定ファイル内でexport-fix-sqlをfalseにした場合には生成されません。

bookshop:books:0:0-0:0.sql

```
-- table: bookshop.books
-- range in sequence: Full
DELETE FROM `bookshop`.`books` WHERE `id` = 24597025 AND `title` = ...
REPLACE INTO `bookshop`.`books`(`id`,`title`,`type`,`published_at`, ...
...
(省略)
...
```

summary.txtには比較結果のサマリーが出力されています。「The following tables contains inconsistent data」を確認すると、booksテーブル内で10件のデータに差分があることが示されています。

summary.txt

```
...
(省略)
...

The following tables contains inconsistent data

+-----+-----+ ... +-----+-----+
| TABLE | RESULT | ... | DATA DIFF ROWS | UPCOUNT | DOWNCOUNT |
+-----+-----+ ... +-----+-----+
| `bookshop`.`books` | succeed | ... | +10/-10 | 10 | 10 |
+-----+-----+ ... +-----+-----+

...
(省略)
...
```

このようにsync-diff-inspectorを使用することで、データベース間のデータの差分を簡単に確認できます。

6-5 まとめ

本章ではTiDBを運用していく上で欠かせない各種ツールを紹介しました。TiDBはこのようなツールが充実していることも特徴で、既存のMySQLからの移行や、効率の良いバックアップ、復元が複雑な手順を踏まずに実現できます。TiDB Cloudでは手元にツールを用意しなくても、Webコンソール経由でバックアップやリストアなどの操作ができますので、ツールの使い方を覚えなくても直感的に操作できます。ぜひ活用してください。

第 **7** 章

TiDBを便利に 使うためのテクニック

リソース制御、ベクトル検索、監視、
パフォーマンス測定

本章ではTiDBを使いこなすために役に立つ便利なテクニックを紹介します。TiDBの機能を最大限に活用するためのノウハウを身につけましょう。

7-1

SQLによるバックアップとリストア

TiDBはSQLによるバックアップとリストアをサポートしています。本節ではBACKUP、RESTORE、LOAD DATA、IMPORT INTOのステートメントについて説明します。

なお、BACKUP ステートメントと RESTORE ステートメントは執筆時点では実験的機能であり、本番環境での使用は推奨されません。予告無しに変更または削除される可能性がありますので使用の際は注意してください。また、どちらも TiDB Cloud Serverless では使用できません。

BACKUP

BACKUP ステートメントは内部ではBRと同じ仕組みが使用されます。BACKUP ステートメントを使用するにあたり、BRを別途インストールする必要はありません。BRについては6-1節を参照してください。BACKUP ステートメントを実行するにはBACKUP_ADMINまたはSUPER権限が必要になります。

BACKUP ステートメントはバックアップタスク全体が完了するか、失敗するか、キャンセルされるまでブロックされます。そのため、場合によっては長時間接続したままになります。加えて、BACKUP ステートメントとRESTORE ステートメントは一度に実行できるタスクは同時に1つだけです。すでに実行されているタスクがあった場合、新しいタスクは現在のタスクが完了するまで待機します。RESTORE ステートメントについては次項で説明します。

BACKUP ステートメントを使用してAmazon S3にバックアップを取得する場合は次のように実行します。T0に続いてs3://でバケットを指定します。access-keyおよびsecret-access-keyはAmazon S3の認証情報を指定します。

Amazon S3にbookshopデータベースのバックアップを取得

```
mysql> BACKUP DATABASE bookshop
-> T0 's3://tidb-book-sample/sql/backup/database/bookshop?access-key=xxx&secret-access-key=xxx';
```

バックアップが完了すると次のように出力されます。

バックアップ完了メッセージ

```
+-----+-----+ ...
| Destination                                | Size    | ...
+-----+-----+ ...
| s3://tidb-book-sample/sql/backup/database/bookshop | 27091404 | ...
+-----+-----+ ...
1 row in set (9.44 sec)
```


過去の時点でのスナップショットを取得する場合はSNAPSHOTと共にBACKUPステートメントを実行します。SNAPSHOTには相対時間、タイムスタンプ、TSOの値を指定できます。相対時間は1h(1時間前)、1d(1日前)、1w(1週間前)などが指定できます。タイムスタンプは'2024-12-01 12:00:00'のように指定します。TSOの値は454682624563085316のように指定します。指定できる過去の期間はGC(ガベージコレクション)の設定次第になります。デフォルトのGCの設定は10分間のデータ保持期間となります。GCの設定については次節「ガベージコレクションの設定」を参照してください。

バックアップのスナップショット指定

```
mysql> BACKUP DATABASE test TO 's3://tidb-book-sample...' SNAPSHOT = 36 HOUR AGO;

mysql> BACKUP DATABASE test TO 's3://tidb-book-sample...' SNAPSHOT = '2024-12-01 12:00:00';

mysql> BACKUP DATABASE test TO 's3://tidb-book-sample...' SNAPSHOT = 454682624563085316;
```

外部ストレージはAmazon S3の他にGoogle Cloud Storage、Azure Blob Storageにも対応しています。指定の仕方の詳細はドキュメントのURI Formats of External Storage Services^{注1}を参照してください。

外部ストレージ以外にlocal://を指定して、TiDBサーバーおよびTiKVサーバーのローカルディスクへのバックアップも可能です。すべてのTiDBサーバーおよびTiKVサーバーのローカルディスクが使用されますので、NFS(Network File System)で共有しているディレクトリを使用することをおすすめします。

BACKUPステートメント内で指定できるバックアップ対象はTiDB全体、データベース、テーブル単位です。

TiDB全体のバックアップ

```
mysql> BACKUP DATABASE * TO 's3://...';
```

データベース全体のバックアップ

```
mysql> BACKUP DATABASE bookshop TO 's3://...';
```

テーブルのバックアップ

```
mysql> BACKUP TABLE bookshop.books TO 's3://...';
```

複数テーブルのバックアップ

```
mysql> BACKUP TABLE bookshop.books, bookshop.orders TO 's3://...';
```

バックアップの履歴はSHOW BACKUPSステートメントで確認できます。

バックアップ履歴

```
mysql> SHOW BACKUPS;
+-----+-----+-----+-----+-----+-----+
| Id | Destination | State | Progress | ... | Connection | Message |
+-----+-----+-----+-----+-----+-----+

```

注1 <https://docs.pingcap.com/tidb/stable/external-storage-uri>

```
+-----+-----+-----+-----+ ... +-----+-----+
| 1 | s3://tidb-book-sample/... | Checksum | 100 | ... | 2069889022 | NULL |
| 2 | s3://tidb-book-sample/... | Checksum | 100 | ... | 2069889030 | NULL |
| 3 | s3://tidb-book-sample/... | Backup | 62.4 | ... | 2069889032 | NULL |
+-----+-----+-----+-----+ ... +-----+-----+
3 rows in set (0.00 sec)
```

SHOW BACKUPS ステートメントで表示される State は次の状態があります。

State	説明
Backup	バックアップ中
Checksum	チェックサム処理
Wait	他の処理の終了を待機中

State でのフィルターや Progress の状況でフィルター表示したい場合は次のように実行します。

State でのフィルター表示

```
mysql> SHOW BACKUPS WHERE State = 'Backup';
```

Progress でのフィルター表示

```
mysql> SHOW BACKUPS WHERE Progress < 100;
```

BACKUP ステートメントをキャンセルする場合は KILL TIDB QUERY ステートメントを実行します。SHOW BACKUPS ステートメント実行時に表示される Connection を指定してキャンセルします。

バックアップのキャンセル

```
mysql> KILL TIDB QUERY 2069889032;
```

RESTORE

BACKUP ステートメントを実行してバックアップされたデータをリストアするには RESTORE ステートメントを実行します。RESTORE ステートメントを実行するには RESTORE_ADMIN または SUPER 権限が必要になります。

BACKUP ステートメントと RESTORE ステートメントは、一度に実行できるタスクは同時に1つだけです。すでに実行されているタスクがあった場合、新しいタスクは現在のタスクが完了するまで待機します。BACKUP ステートメントについては前項を参照してください。

RESTORE ステートメントを実行する際は、リストア対象のテーブルまたはデータベースが TiDB に存在していない必要があります。

RESTORE ステートメントを使用して Amazon S3 からリストアする場合は次のように実行します。FROM に続いて s3:// でバケットを指定します。access-key および secret-access-key は Amazon S3 の認証情報を指定します。

Amazon S3からbookshopデータベースのリストア

```
mysql> RESTORE DATABASE bookshop
-> FROM 's3://tidb-book-sample/sql/backup/database/bookshop?access-key=xxxxxxxxxx&secret-access
```

リストアが完了すると次のように出力されます。

リストア完了時のメッセージ

```
+-----+-----+ ...
| Destination                | Size    | ...
+-----+-----+ ...
| s3://tidb-book-sample/sql/backup/database/bookshop | 13876148 | ...
+-----+-----+ ...
```

FROMに指定可能なURIはBACKUPステートメントと同様にAmazon S3の他にGoogle Cloud Storage、Azure Blob Storage、ローカルディスクが指定できます。

RESTOREステートメント内で指定できるリストア対象はTiDB全体、データベース、テーブルです。

TiDB全体のリストア

```
mysql> RESTORE DATABASE * TO 's3://...';
```

データベース全体のリストア

```
mysql> RESTORE DATABASE bookshop FROM 's3://...';
```

テーブルのリストア

```
mysql> RESTORE TABLE bookshop.books FROM 's3://...';
```

複数テーブルのリストア

```
mysql> RESTORE TABLE bookshop.books, bookshop.orders FROM 's3://...';
```

リストアの履歴はSHOW RESTORESステートメントで確認できます。SHOW BACKUPSステートメントの実行時と同様にWHERE句でのフィルター表示が可能です。

LOAD DATA

LOAD DATAステートメントはローカルファイルまたは外部ストレージからデータをインポートするためのステートメントです。MySQLも同様のLOAD DATAステートメントを実行してローカルファイルからデータをインポート可能ですが、TiDBのLOAD DATAステートメントは外部ストレージからのインポートにも対応しています。

TiDB Cloudに接続してLOAD DATAを行う場合、ローカルファイルを読み込ませるためにmysqlコマンドでの接続時に--local-infileオプションを使用する必要があります。

次の例はローカルディレクトリにエクスポートしたbookshop.booksテーブル(CSV形式)をインポートする例です。FIELDS TERMINATED BYで区切り文字を指定します。ENCLOSED BYで囲み文字を指定しま

す。IGNORE 1 LINESでヘッダー行を無視します。

```
ローカルファイルからのインポート
mysql> LOAD DATA LOCAL INFILE 'bookshop.books.000000000.csv'
-> INTO TABLE bookshop.books
-> FIELDS TERMINATED BY ','
-> ENCLOSED BY '\"'
-> IGNORE 1 LINES;
Query OK, 20000 rows affected (0.44 sec)
Records: 20000 Deleted: 0 Skipped: 0 Warnings: 0
```

Amazon S3からLOAD DATAを行う場合は次のように指定します。

```
外部ストレージからのインポート
mysql> LOAD DATA INFILE
-> 's3://.../bookshop.books.000000000.csv?access-key=xxx&secret-access-key=xxx'
-> INTO TABLE bookshop.books
-> FIELDS TERMINATED BY ','
-> ENCLOSED BY '\"'
-> IGNORE 1 LINES;
Query OK, 20000 rows affected (0.44 sec)
Records: 20000 Deleted: 0 Skipped: 0 Warnings: 0
```

外部ストレージはAmazon S3の他にGoogle Cloud Storage、Azure Blob Storageにも対応しています。

IMPORT INTO

IMPORT INTOステートメントはTiDB Lightningの物理インポートモードと同じ仕組みでデータをインポートします。IMPORT INTOステートメントを実行するにはインポート対象テーブルに対するSELECT、UPDATE、INSERT、DELETE、およびALTER権限が必要です。ローカルファイルをインポートする場合は別途FILE権限も必要になります。

IMPORT INTOステートメントは1つのテーブルに対してのみ実行できます。対象のテーブルが複数の場合はそれぞれIMPORT INTOステートメントを実行する必要があります。また、対象のテーブルは空である必要があります。

IMPORT INTOはローカルファイル、または外部ストレージからデータをインポートできますが、SELECTステートメントの結果のインポートも可能です。

IMPORT INTOステートメントを使用してAmazon S3からインポートする場合は次のように指定します。SQLが書かれているデータの場合はFORMATをsqlに指定します。

Amazon S3からのSQLファイルのインポート

```
mysql> IMPORT INTO bookshop.books
  -> FROM 's3://.../bookshop.books.0000000000.sql?access-key=xxx&secret-access-key=xxx'
  -> FORMAT 'sql';
```

Job_ID	Data_Source	Target_Table	...	Status	...
4	s3://tidb-book-sample/...	`bookshop`.`books`	...	finished	...

```
1 row in set (5.97 sec)
```

CSVファイルをインポートする場合はFORMATをcsvに指定します。1行目をスキップする場合はWITH句でskip_rows=1を指定します。

Amazon S3からのCSVファイルのインポート

```
mysql> IMPORT INTO bookshop.books
  -> FROM 's3://.../bookshop.books.0000000000.csv?access-key=xxx&secret-access-key=xxx'
  -> FORMAT 'csv'
  -> WITH skip_rows=1;
```

SELECTステートメントの結果をインポートする場合は次のように指定します。

SELECTステートメントの結果をインポート

```
mysql> IMPORT INTO new_bookshop.books
  -> FROM SELECT * FROM bookshop.books;
```

Query OK, 20000 rows affected (2.64 sec)
Records: 20000, ID: 7aca2547-1171-4ab1-a2db-3af01bf0d133

また、AS OF TIMESTAMPと組み合わせることで特定の日時のデータをインポートできます。

```
mysql> IMPORT INTO new_bookshop_20241215.books
  -> FROM SELECT * FROM bookshop.books AS OF TIMESTAMP '2024-12-15 00:00:00'
```

Query OK, 20000 rows affected (0.58 sec)
Records: 20000, ID: 892d946e-9ce7-4acd-bf60-045fe1120e6b

IMPORT INTOステートメントの実行は時間がかかる場合があるため、バックグラウンドのインポートジョブとして実行させることができます。WITH句でDETACHEDを指定することでIMPORT INTOステートメントはすぐにインポートジョブ情報を返します。

バックグラウンドでインポートジョブを実行

```
mysql> IMPORT INTO bookshop.books
  -> FROM 's3://.../bookshop.books.0000000000.sql?access-key=xxx&secret-access-key=xxx'
  -> FORMAT 'sql'
  -> WITH DETACHED;
```

Job_ID	Data_Source	Target_Table	...	Status	...

```
+-----+-----+-----+ ... +-----+ ...  
|      5 | s3://tidb-book-sample/... | `bookshop`.`books` | ... | pending | ...  
+-----+-----+-----+ ... +-----+ ...  
1 row in set (0.89 sec)
```

実行したインポートジョブのStatusは初期状態のpendingで表示されます。インポートジョブの状態を確認する場合はSHOW IMPORT JOB ステートメントを、表示されたJob_IDを指定して実行します。

インポートジョブの状態確認

```
mysql> SHOW IMPORT JOB 5;  
+-----+-----+-----+ ... +-----+ ...  
| Job_ID | Data_Source          | Target_Table      | ... | Status  | ...  
+-----+-----+-----+ ... +-----+ ...  
|      5 | s3://tidb-book-sample/... | `bookshop`.`books` | ... | finished | ...  
+-----+-----+-----+ ... +-----+ ...  
1 row in set (0.00 sec)
```

Statusがfinishedであればインポートジョブが完了しています。

すべてのインポートジョブを表示する場合はSHOW IMPORT JOBS ステートメントを実行します。

すべてのインポートジョブの状態確認

```
mysql> SHOW IMPORT JOBS;  
+-----+-----+-----+ ... +-----+ ...  
| Job_ID | Data_Source          | Target_Table      | ... | Status  | ...  
+-----+-----+-----+ ... +-----+ ...  
|      1 | s3://tidb-book-sample/... | `bookshop`.`orders` | ... | finished | ...  
|      2 | s3://tidb-book-sample/... | `bookshop`.`authors` | ... | finished | ...  
|      3 | s3://tidb-book-sample/... | `bookshop`.`ratings` | ... | finished | ...  
|      4 | s3://tidb-book-sample/... | `bookshop`.`users` | ... | finished | ...  
|      5 | s3://tidb-book-sample/... | `bookshop`.`books` | ... | finished | ...  
+-----+-----+-----+ ... +-----+ ...  
5 row in set (0.01 sec)
```

7-2

ガベージコレクションの設定

TiDBのストレージクラスターを担当しているTiKVでは、データを保存する際にMVCC(Multi-Version Concurrency Control)を使用してタイムスタンプと共にバージョン管理してデータを保存しています。そのため、古いデータ(更新前のデータまたは削除されたデータ)はすぐにTiKV内から削除されずに一定期間残っています。

TiKVでは一定期間過ぎた古いデータを削除するGC(ガベージコレクション)が定期的に行われます。

す。デフォルトでは古いデータは10分間保持されています。

保持期間の変更

GCによる保持期間を変更する場合はtidb_gc_life_timeのシステム変数を変更します。

GCの保持期間の確認

```
mysql> select @@tidb_gc_life_time;
+-----+
| @@tidb_gc_life_time |
+-----+
| 10m0s                |
+-----+
1 row in set (0.00 sec)
```

1h(1時間)に変更する場合は次のように実行します。

GCの保持期間の変更

```
mysql> SET GLOBAL tidb_gc_life_time = 1h;
Query OK, 0 rows affected (0.02 sec)

mysql> select @@tidb_gc_life_time;
+-----+
| @@tidb_gc_life_time |
+-----+
| 1h0m0s              |
+-----+
1 row in set (0.00 sec)
```

保持期間に設定できる範囲はTiDB Self-ManagedおよびTiDB Cloud Dedicatedの場合は10分から8,760時間(1年)までの範囲になります。TiDB Cloud Serverlessの場合は10分から168時間(1週間)までの範囲になります。

保持期間を大きくする場合には、その分データ容量が増えるためディスク容量に注意して設定してください。また、保持しているデータのバージョンが多くなるとSELECTステートメントの実行速度が遅くなる可能性があります。スローログに表示されるTotal_keysおよびProcess_keysを確認し、数字に大きな乖離^{かいり}がある場合は保持期間を短くすることを検討してください。

リスト7.1のSQLは調査する際に役に立つSQLの例です。WHERE句の条件は適宜変更してください。Is_internal = 0はTiDB内部ではなく、ユーザーが実行したSQLであることを判別しています(リスト7.1の❶)。Cop_time > 1はTiKVからのデータ取得(コプロセッサ)にかかった時間が1秒より大きいことを条件にしています(リスト7.1の❷)。Total_keysはTiKVが処理したすべてのキーの数になります。Process_keysはTiKVが処理したすべてのキーからMVCCの古いバージョンを除いたキー数です。Process_keys > 0はProcess_keysが1件以上であることを条件にしています(リスト7.1の❸)。Total_

keys / Process_keys >= 10はTotal_keysがProcess_keysと比べて10倍以上であることを条件としています(リスト7.1の❶)。Time >= NOW() - INTERVAL 10 MINUTEは直近10分間のスローログを取得する条件です(リスト7.1の❷)。

なお、INFORMATION_SCHEMA内のSLOW_QUERYテーブルからは接続しているTiDBサーバーのスローログのみ取得できます。TiDBクラスター全体のスローログから調査したい場合はCLUSTER_SLOW_QUERYテーブルから取得するように変更してください(リスト7.1の❸)。

リスト7.1 Total_keysとProcess_keysの乖離の確認例

```
SELECT
  Time,
  DB,
  Index_names,
  Process_keys,
  Total_keys,
  CONCAT(
    LEFT(REGEXP_REPLACE(Query, '\\s+', ' '), 20),
    '...',
    RIGHT(REGEXP_REPLACE(Query, '\\s+', ' '), 10)
  ) as Query,
  Query_time,
  Cop_time,
  Process_time
FROM
  INFORMATION_SCHEMA.SLOW_QUERY —❸
WHERE
  Is_internal = 0 —❶
  AND Cop_time > 1 —❷
  AND Process_keys > 0 —❹
  AND Total_keys / Process_keys >= 10 —❺
  AND Time >= NOW() - INTERVAL 10 MINUTE —❻
ORDER BY Total_keys DESC
LIMIT 5;
```

▶ TiKV MVCC In-Memory Engine

TiDBのバージョン8.5.0より、多数のバージョンを保持しているデータをスキャンする処理を高速化するために、TiKV MVCC In-Memory Engine(IME)が導入されました。この機能を有効にすると最新のバージョンをTiKVのメモリ上にキャッシュできるようになり、スキャン処理が高速化されます。この機能は頻繁に更新または削除されるデータを参照する必要がある場合や、GCの保持期間(tidb_gc_life_time)を長く設定している場合に有効です。

IMEはデフォルトで無効になっています。有効にする場合はTiKVの設定内のin-memory-engineセクションの項目を設定してください(リスト7.2)。enableがtrueの場合はIMEが有効になります(リスト7.2の❶)。デフォルトはfalseでIMEは無効です。capacityはIMEで使用するTiKVのメモリ容量を指

定します(リスト 7.2 の❷)。デフォルトはシステムメモリ容量の10%になりますが、最大5GiBまでです。gc-run-intervalはIMEのキャッシュデータのGCが実行される間隔を指定します(リスト 7.2 の❸)。デフォルトは3mで3分間隔です。なお、このGC処理はMVCCのデータのGCとは異なり、IMEのキャッシュデータのGCとなります。mvcc-amplification-thresholdはIMEのキャッシュとしてロードされるデータのMVCCのバージョン数のしきい値を設定します(リスト 7.2 の❹)。デフォルトは10です。

リスト7.2 IMEの設定項目

```
[in-memory-engine]
enable = true      —❶
capacity = "5GiB" —❷
gc-run-interval = "3m" —❸
mvcc-amplification-threshold = 10 —❹
```

enableの設定は動的に反映できません。IMEの有効化および無効化はTiKVを再起動する必要がありますので注意してください。enable以外の設定は動的に反映できます。

ガベージコレクションの実行間隔の設定

GCが実行される間隔はtidb_gc_run_intervalのシステム変数で設定します。デフォルトでは10分間隔でGCが実行されます。

```
GCの実行間隔の確認
mysql> select @@tidb_gc_run_interval;
+-----+
| @@tidb_gc_run_interval |
+-----+
| 10m0s                  |
+-----+
1 row in set (0.01 sec)
```

1h(1時間)に変更する場合は次のように実行します。

```
GCの実行間隔の変更
mysql> SET GLOBAL tidb_gc_run_interval = 1h;
Query OK, 0 rows affected (0.02 sec)

mysql> select @@tidb_gc_run_interval;
+-----+
| @@tidb_gc_run_interval |
+-----+
| 1h0m0s                 |
+-----+
1 row in set (0.00 sec)
```

ガベージコレクションの停止

GCによるデータの削除を停止する場合はtidb_gc_enableのシステム変数をOFF(0)に設定します。デフォルトはON(1)です。GCの停止はおすすめしませんので、検証時のみ停止するようにしてください。

GCの停止

```
mysql> SET GLOBAL tidb_gc_enable = OFF;  
1 row in set (0.00 sec)
```

```
mysql> select @@tidb_gc_enable;
```

```
+-----+
```

```
| @@tidb_gc_enable |
```

```
+-----+
```

```
|                0 |
```

```
+-----+
```

```
1 row in set (0.00 sec)
```

7-3

リソース制御による マルチテナント環境の構築

TiDBはユーザーやSQLに対してリソース制御をする仕組みが備わっており、この制御は特にマルチテナント環境でTiDBを利用する場合には重要です。たとえばマルチテナント環境でTiDBを運用したい場合は、ノイズネイバー問題をなるべく起こさないようにユーザーに対して適切なリソース制御が求められます。

TiDBはリソースをリクエストユニット(RU)という仮想的な単位で管理します。リクエストユニットはCPU消費量やストレージI/O帯域などのリソース消費量を元に計算されます(表7.1)。

表7.1 リクエストユニットの消費量

リソースタイプ	リクエストユニット消費量
データ読み取り	2つのストレージ読み取りバッチごとに1消費 8つのストレージ読み取り要求ごとに1消費 64KiBのデータ読み取りごとに1消費
データ書き込み	1回のストレージ書き込みバッチごとに1消費 1回のストレージ書き込み要求ごとに1消費 1KiBの書き込みごとに1消費
CPU消費量	3ミリ秒ごとに1消費

リクエストユニットを適用する際はリソースグループを作成する必要があり、作成時にリクエストユニット量を指定します。その後リソースグループをユーザーやSQLに対して割り当てます。この操作はバインドと呼ばれます。リソースグループのバインドはユーザーの他、セッションレベルでのバ

インド、特定のSQLに対してのバインドにも対応しています。

リクエストを処理する際にリクエストユニットが不足した場合、そのリクエストはリクエストユニットが確保できるまで待機しますが、BURSTABLEフラグを有効にしている場合はリクエストユニット数の制限を超えて処理させることが可能です。リクエストユニットが確保できない状態が続いた場合、該当処理は最終的にタイムアウトすることになります。また、リソースグループを定義する際にPRIORITYによる優先度の設定ができます。PRIORITYはHIGH、MEDIUM、LOWの値を設定可能で、ストレージ層における処理の優先度を制御できます。

リソースグループの確認

定義されているリソースグループはINFORMATION_SCHEMA内のRESOURCE_GROUPSテーブルで確認できます。初期状態ではdefaultリソースグループが作成されています。

リソースグループの確認

```
mysql> SELECT * FROM INFORMATION_SCHEMA.RESOURCE_GROUPS;
```

NAME	RU_PER_SEC	PRIORITY	BURSTABLE	QUERY_LIMIT	BACKGROUND
default	UNLIMITED	MEDIUM	YES	NULL	NULL

1 row in set (0.01 sec)

defaultリソースグループは秒間リクエストユニット数の制限(RU_PER_SEC)がUNLIMITED(内部的にはINTタイプの最大値である2,147,483,647に相当)に設定されており、かつBURSTABLEフラグが有効になっています。PRIORITYはMEDIUMです。どのリソースグループにもバインドされていないすべてのリクエストはdefaultリソースグループに自動的にバインドされます。

リソースグループによるリクエストユニットの消費量の履歴はmysqlスキーマ内のrequest_unit_by_groupテーブルに格納されます。

リクエストユニットの消費量の履歴の確認

```
mysql> SELECT * FROM mysql.request_unit_by_group;
```

start_time	end_time	resource_group	total_ru
2024-12-01 00:00:00.000000	2024-12-02 00:00:00.000000	default	185374
2024-12-02 00:00:00.000000	2024-12-03 00:00:00.000000	default	966290
2024-12-03 00:00:00.000000	2024-12-04 00:00:00.000000	default	271367
2024-12-04 00:00:00.000000	2024-12-05 00:00:00.000000	default	990936

4 rows in set (0.02 sec)

request_unit_by_groupテーブルはTiDB内のスケジュールタスクによって毎日の終わりに自動的に

データが挿入されます。リクエストユニット消費量が0の場合はレコードは生成されません。デフォルトではデータは92日間保存されます。

リソースグループの管理

リソースグループの作成はCREATE RESOURCE GROUP ステートメントを実行します。リソース名と秒間あたりのリクエストユニット消費量を定義します。リソースグループを作成、変更、または削除するにはSUPERまたはRESOURCE_GROUP_ADMIN権限が必要になります。

リソースグループの作成

```
mysql> CREATE RESOURCE GROUP my_rg RU_PER_SEC = 500;  
Query OK, 0 rows affected (0.52 sec)
```

作成したリソースグループを確認します。

リソースグループの確認

```
mysql> SELECT * FROM INFORMATION_SCHEMA.RESOURCE_GROUPS;  
+-----+-----+-----+-----+-----+-----+  
| NAME   | RU_PER_SEC | PRIORITY | BURSTABLE | QUERY_LIMIT | BACKGROUND |  
+-----+-----+-----+-----+-----+-----+  
| default | UNLIMITED | MEDIUM  | YES       | NULL        | NULL       |  
| my_rg   | 500        | MEDIUM  | NO        | NULL        | NULL       |  
+-----+-----+-----+-----+-----+-----+  
2 rows in set (0.00 sec)
```

リソースグループ作成時にBURSTABLEフラグやPRIORITYは省略可能です。省略した場合はBURSTABLEはfalse、PRIORITYはMEDIUMに設定されます。リソースグループ作成時に設定する場合は次のように実行します。

リソースグループの作成2

```
mysql> CREATE RESOURCE GROUP my_burst_rg RU_PER_SEC = 100 BURSTABLE = true;  
Query OK, 0 rows affected (0.54 sec)  
  
mysql> CREATE RESOURCE GROUP my_high_rg RU_PER_SEC = 1000 PRIORITY = HIGH;  
Query OK, 0 rows affected (0.52 sec)
```

定義済みのリソースグループの内容を変更する場合はALTER RESOURCE GROUP ステートメントを実行します。先ほど作成したmy_rgに対してBURSTABLEフラグを有効にする場合は次のように実行します。

リソースグループの設定変更

```
mysql> ALTER RESOURCE GROUP my_rg BURSTABLE = true;  
Query OK, 0 rows affected (0.52 sec)
```

リソースグループを削除する場合はDROP RESOURCE GROUP ステートメントを実行します。

リソースグループの削除

```
mysql> DROP RESOURCE GROUP my_rg;
Query OK, 0 rows affected (0.53 sec)
```

▶ランナウェイクエリの制御

ランナウェイクエリは、予想以上にリソースを使用したり、実行時間がかかったりしてしまうようなクエリのことを指します。TiDBではランナウェイクエリを識別した際に、リソースを使い果たしたり他のクエリに影響を与えたりしないように、ランナウェイクエリに対して自動的にアクションを実行できます。

ランナウェイクエリに関する設定はリソースグループの定義時に `QUERY_LIMIT` を設定することで行います。`QUERY_LIMIT` は次のパラメーターを指定できます。`EXEC_ELAPSED`、`PROCESSED_KEYS`、`RU` はランナウェイクエリとして識別されるクエリの条件設定になります。`ACTION` はランナウェイクエリに対して実行されるアクションを指定します。`WATCH` はリソースを使い果たす大量の同時実行型ランナウェイクエリを回避するために、ランナウェイクエリとして識別されたクエリと似たようなクエリを、条件にマッチする前にすぐにランナウェイクエリとして認識できるようにするための設定です。各パラメーターの詳細を表7.2に示します。

表7.2 QUERY_LIMITの設定パラメーター

パラメーター	設定	説明
<code>EXEC_ELAPSED</code>	数値	クエリ実行時間がこの値を超えるとランナウェイクエリとして識別される
<code>PROCESSED_KEYS</code>	数値	クエリが処理したキーの数がこの値を超えるとランナウェイクエリとして識別される
<code>RU</code>	数値	クエリが消費したリクエストユニット数がこの値を超えるとランナウェイクエリとして識別される
<code>ACTION</code>	<code>DRYRUN</code>	アクションは実行されない
	<code>COOLDOWN</code>	クエリの実行優先度が最低まで下がる
	<code>KILL</code>	クエリを終了させる
	<code>SWITCH_GROUP</code>	指定された別のリソースグループに切り替わる
<code>WATCH</code>	<code>EXACT</code>	同じSQLテキストを持つクエリを識別する
	<code>SIMILAR</code>	同じSQLテキストを持つクエリを識別する。リテラル値の違いは無視
	<code>PLAN</code>	同じ実行計画を持つクエリを識別する
	<code>DURATION</code>	識別の有効期間を指定する。デフォルトは0（無制限）

たとえば、ランナウェイクエリとして1,000件以上キーを処理したクエリを識別し、自動で終了させる設定は次のように定義します。

ランナウェイクエリの定義

```
mysql> CREATE RESOURCE GROUP my_rw_rg RU_PER_SEC = 1000 QUERY_LIMIT = (PROCESSED_KEYS=1000, ACTION=KILL);
Query OK, 0 rows affected (0.10 sec)
```

リソースグループ `my_rw_rg` をバインドさせてクエリを実行してみます。エラーと共に「identified as runaway query」のメッセージが表示されます。

ランナウェイクエリの確認

```
mysql> SELECT /*+ RESOURCE_GROUP(my_rw_rg) */ * FROM bookshop.books WHERE id > 10000000;  
ERROR 8253 (HY000): Query execution was interrupted, identified as runaway query [ProcessedKeys =  
1696(1000)]
```

ランナウェイクエリと識別されたクエリはmysqlスキーマ内のtidb_runaway_queriesテーブルに記録されます。

```
mysql> SELECT * FROM mysql.tidb_runaway_queries ORDER BY start_time DESC LIMIT 1 \G  
***** 1. row *****  
resource_group_name: my_rw_rg  
start_time: 2024-12-01 11:58:45  
repeats: 1  
match_type: identify  
action: kill  
sample_sql: SELECT /*+ RESOURCE_GROUP(my_rw_rg) */ * FROM bookshop.books ...  
sql_digest: 57a6037de727ccb4bfa1f5b2d2dc836806d51447dc564b4e70e1cb636eb4d1d4  
plan_digest: c7a1b8c44291aa428bbf468071214b77eaf494243d3cb0173e7604771ff7da18  
tidb_server: 127.0.0.1:4000  
rule: ProcessedKeys = 1696(1000)  
1 row in set (0.01 sec)
```

WATCHの設定をした場合はINFORMATION_SCHEMA内のRUNAWAY_WATCHESテーブルに状態が記録されます。次の例ではWATCH=SIMILAR DURATION='1h'を設定した場合の出力結果です。

```
mysql> CREATE RESOURCE GROUP my_rw_rg  
-> QUERY_LIMIT = (PROCESSED_KEYS=1000, ACTION=KILL, WATCH=SIMILAR DURATION='1h');  
  
mysql> SELECT /*+ RESOURCE_GROUP(my_rw_rg) */ * FROM bookshop.books WHERE id > 10000000;  
  
mysql> SELECT * FROM INFORMATION_SCHEMA.RUNAWAY_WATCHES \G  
***** 1. row *****  
ID: 1  
RESOURCE_GROUP_NAME: my_rw_rg  
START_TIME: 2024-12-01 11:58:45  
END_TIME: 2024-12-01 12:58:45  
WATCH: Similar  
WATCH_TEXT: 57a6037de727ccb4bfa1f5b2d2dc836806d51447dc564b4e70e1cb636eb4d1d4  
SOURCE: 127.0.0.1:4000  
ACTION: Kill  
RULE: ProcessedKeys = 1696(1000)  
1 row in set (0.01 sec)
```

QUERY_LIMITの設定を削除する場合はALTER RESOURCE GROUPステートメントを実行してNULLを指定します。

ランナウェイクエリの設定削除

```
mysql> ALTER RESOURCE GROUP my_rw_rg QUERY_LIMIT = NULL;
Query OK, 0 rows affected (0.08 sec)
```

▶ バックグラウンドタスクの制御

データのバックアップや自動統計収集などのバックグラウンドタスクは、時に多くのリソースを消費することがあります。これらのタスクは通常は定期的または不定期にトリガーされ、他の高優先度タスクのパフォーマンスに影響する場合があります。TiDB ではこの問題を軽減する方法として、バックグラウンドタスクのリソースを制御するためにリソースグループを割り当てる機能が提供されています。

TiDB ではバックグラウンドタスクとして次の種類のタスクのリソース制御をサポートしています(表 7.3)。

表7.3 バックグラウンドタスクのリソース制御

タスク種別	説明
lightning	TiDB Lightningのデータインポートタスク。物理および論理モードの両方をサポート
br	BRを使用したバックアップおよびリストアタスク。ポイントインタイムリストアは未サポート
ddl	reorg DDLによる書き戻し処理のタスク
stats	手動で実行されるか、TiDBによって自動的にトリガーされる統計を収集するタスク
background	システム変数を使用して現在のセッションのタスクタイプを指定するための予約済みのタスクタイプ

reorg DDLとは

column

DDL ステートメントは、オフラインDDL ステートメントとオンラインDDL ステートメントに分類されます。これはDDL ステートメント実行時にユーザーのアプリケーションをブロックするかどうかで分かれています。オフラインDDL ステートメントは対象のデータベースオブジェクトをロックします。一方、オンラインDDL ステートメントはブロックすること無くDDLを実行します。TiDBのDDLステートメントはオンラインDDLステートメントとして実行されます。

オンラインDDLステートメントはさらに論理DDLステートメントと物理DDLステートメントに分類され、論理DDLステートメントは通常テーブル名の変更や列名の変更など、オブジェクトに格納されているデータを処理せずにメタデータのみを変更します。論理DDLステートメントは完了するまでに数十ミリ秒または数秒しかかからないことが多く、システムリソースをあまり消費しません。物理DDLステートメントはreorg DDLとも呼ばれ、メタデータを変更するだけでなく、格納されているデータも変更します。たとえば、TiDBがテーブルのインデックスを作成する場合、テーブル全体のスキャンを実行して新しく追加されたインデックスを構築する場合は該当します。

バックグラウンドタスクを制御するためにはリソースグループを定義する際にBACKGROUNDを設定します。BACKGROUNDは次のパラメーターを指定できます。TASK_TYPEはリソース制御を適用するバックグラウンドタスクの種類を指定します。UTILIZATION_LIMITは各TiKVでバックグラウンドタスクが消費できるリソースの最大パーセンテージ(0~100)を設定します。

バックグラウンドタスクは執筆時点ではすべて default リソースグループにバインドされます。そのため、バックグラウンドタスクのリソースを制御する場合は default リソースグループに対して設定をする必要があります。

TiDB Lightning および BR のタスクのリソース使用量を 20% に制限する場合は次のように設定します。

バックグラウンドタスクのリソース制御

```
mysql> ALTER RESOURCE GROUP default BACKGROUND=(TASK_TYPES='lightning,br', UTILIZATION_LIMIT=20);
Query OK, 0 rows affected (0.08 sec)
```

現在のセッションのタスクをバックグラウンドタイプとして実行する場合は `tidb_request_source_type` を使用してタスクタイプを明示的に background に指定します。

バックグラウンドタスクとして実行

```
mysql> ALTER RESOURCE GROUP default BACKGROUND=(TASK_TYPES="background", UTILIZATION_LIMIT=30);
Query OK, 0 rows affected (0.07 sec)

mysql> SET @@tidb_request_source_type="background";
Query OK, 0 rows affected (0.00 sec)

mysql> BACKUP DATABASE bookshop TO ...
```

なお、バックグラウンドタスクのリソース制御機能は執筆時点では実験的な機能です。本番環境での使用は推奨されません。この機能は予告なしに変更または削除される可能性がありますので注意してください。

リソースグループのバインド

リソースグループをバインドする方法は3つのレベルで提供されています。ユーザーレベル、セッションレベル、クエリレベルの3つです。システム変数 `tidb_resource_control_strict_mode` が ON の場合、セッションレベルおよびクエリレベルでリソースグループをバインドするためには SUPER、RESOURCE_GROUP_ADMIN または RESOURCE_GROUP_USER 権限が必要になります。デフォルトでは `tidb_resource_control_strict_mode` は ON になっています。

▶ ユーザーレベルで行う場合

ユーザーレベルでリソースグループをバインドする場合はユーザー作成時にリソースグループを指定するか、既存のユーザーに対しては ALTER USER ステートメントを実行します。

ユーザー作成時にバインド

```
mysql> CREATE USER 'my_user'@'%' IDENTIFIED BY 'my_password' RESOURCE GROUP my_rg;
Query OK, 0 rows affected (0.06 sec)
```


既存のユーザーにバインド

```
mysql> ALTER USER 'my_user'@'%' RESOURCE GROUP my_rg;
Query OK, 0 rows affected (0.03 sec)
```

ユーザーにバインドされているリソースグループを確認する場合はmysqlスキーマ内のuserテーブルを参照します。User_attributes カラムにリソースグループが設定されています。

リソースグループの確認

```
mysql> SELECT USER, JSON_EXTRACT(User_attributes, "$.resource_group") AS resource_group
-> FROM mysql.user WHERE user = "my_user";
+-----+-----+
| USER | resource_group |
+-----+-----+
| my_user | "my_rg" |
+-----+-----+
1 row in set (0.00 sec)
```

なお、リソースグループのバインドを解除する場合はALTER USER ステートメントでRESOURCE GROUP をdefaultに設定します。

リソースグループの解除

```
mysql> ALTER USER 'my_user'@'%' RESOURCE GROUP 'default';
Query OK, 0 rows affected (0.04 sec)
```

▶ セッションレベルで行う場合

次にセッションレベルでのリソースグループのバインド方法を示します。セッションレベルでリソースグループをバインドする場合はSET RESOURCE GROUP ステートメントを実行します。

セッションレベルでのリソースグループのバインド

```
mysql> SET RESOURCE GROUP my_rg;
Query OK, 0 rows affected (0.00 sec)
```

現在のセッション内でバインドされているリソースグループを確認する場合はSELECT CURRENT_RESOURCE_GROUP() ステートメントを実行します。

現在のセッションのリソースグループの確認

```
mysql> SELECT CURRENT_RESOURCE_GROUP();
+-----+
| CURRENT_RESOURCE_GROUP() |
+-----+
| my_rg |
+-----+
1 row in set (0.00 sec)
```

クエリレベルで行う場合

最後にクエリレベルでのリソースグループのバインド方法を示します。クエリレベルでリソースグループをバインドする場合はクエリ内のコメント構文でRESOURCE_GROUPを使用します。このコメント構文はSELECT、INSERT、UPDATEおよびDELETEステートメントをサポートしています。

クエリレベルでのリソースグループのバインド

```
mysql> SELECT /*+ RESOURCE_GROUP(my_rg) */ * FROM bookshop.books;
```

リクエストユニットの計測

リクエストユニットは直感的にどれくらい消費するのか把握しにくい概念です。そこで役に立つのがEXPLAIN ANALYZEステートメントによるリクエストユニットの計測です。EXPLAIN ANALYZEステートメントはクエリの実行計画を表示するSQLで、リクエストユニットの消費量も表示されます。EXPLAIN ANALYZEステートメントを使用することでクエリの実行計画を確認しながらリクエストユニットの消費量を把握できます。なお、EXPLAIN ANALYZEによって表示されるリクエストユニット数は、実際に消費されるリクエストユニット数とは異なる場合があります。

20,000件あるデモ用データのbooksテーブルを全件取得するクエリを実行した場合のリクエストユニットの計測結果を確認します。リクエストユニットの消費量はexecution info内のRUに表示されます。

EXPLAIN ANALYZEによるリクエストユニットの計測1

```
mysql> EXPLAIN ANALYZE SELECT * FROM bookshop.books;
+-----+-----+ ... +-----+ ...
| id          | estRows | ... | execution info          | ...
+-----+-----+ ... +-----+ ...
| TableReader_5 | 20000.00 | ... | time:98.6ms, loops:21, RU:52.313960, ...
|   └─TableFullScan_4 | 20000.00 | ... | tikv_task:{proc max:20ms, min:1ms, ...
+-----+-----+ ... +-----+ ...
2 rows in set (0.11 sec)
```

次にLIMIT 100を指定した場合のリクエストユニットの計測結果を確認します。

EXPLAIN ANALYZEによるリクエストユニットの計測2

```
mysql> EXPLAIN ANALYZE SELECT * FROM bookshop.books LIMIT 100;
+-----+-----+ ... +-----+ ...
| id          | estRows | ... | execution info          | ...
+-----+-----+ ... +-----+ ...
| Limit_7      | 100.00  | ... | time:1.44ms, loops:2, RU:0.709051 ...
|   └─TableReader_11 | 100.00  | ... | time:1.43ms, loops:1, cop_task: ...
|     └─Limit_10    | 100.00  | ... | tikv_task:{time:1ms, loops:3}, ...
|       └─TableFullScan_9 | 100.00  | ... | tikv_task:{time:1ms, loops:3} ...
+-----+-----+ ... +-----+ ...
4 rows in set (0.00 sec)
```

図7.1 TiDB Cloudでのリクエストユニットの確認

Diagnosis

Slow QuerySQL Statement

Statements may take 1-3 minutes to appear here. View [Statements docs](#) to learn more.

30mPast 30 minutesAll KindsFilter Keyword

Statement Template	Total Latency ↓	Mean Latency	Executions Count	Plans Count	Total RU	Mean RU
INSERT INTO `ratings` (`book_id`, ...	10.8 s	37.0 ms	292	1	123.4 K	423
INSERT INTO `orders` (`id`, `bo...	10.0 s	34.3 ms	292	1	108.8 K	373
INSERT INTO `books` (`id`, `tit...	1.2 s	62.2 ms	19	1	6.0 K	314
DROP TABLE IF EXISTS `orders`;	828.6 ms	828.6 ms	1	1	0	0
DROP TABLE IF EXISTS `ratings`;	7419 ms	7419 ms	1	1	0	0

リクエストユニット消費量がRU:52.313960からRU:0.709051に減少していることが確認できます。

TiDB Cloudを利用している場合は、クラスター概要画面のナビゲーションペインの「Diagnosis」を選択し、「SQL Statement」タブを選択することで実行されたクエリをリクエストユニットと共に確認できます(図7.1)。リクエストユニットの列でソート表示もできますので、リクエストユニットの消費量が多いクエリを特定する場合に便利です。

リソースグループ制御の無効化

TiDB 全体でリソースグループ制御を無効にする場合はシステム変数のtidb_enable_resource_controlをOFFに設定します。デフォルトではONになっています。

```
mysql> SET GLOBAL tidb_enable_resource_control = 'OFF';
```

7-4

メタデータロック

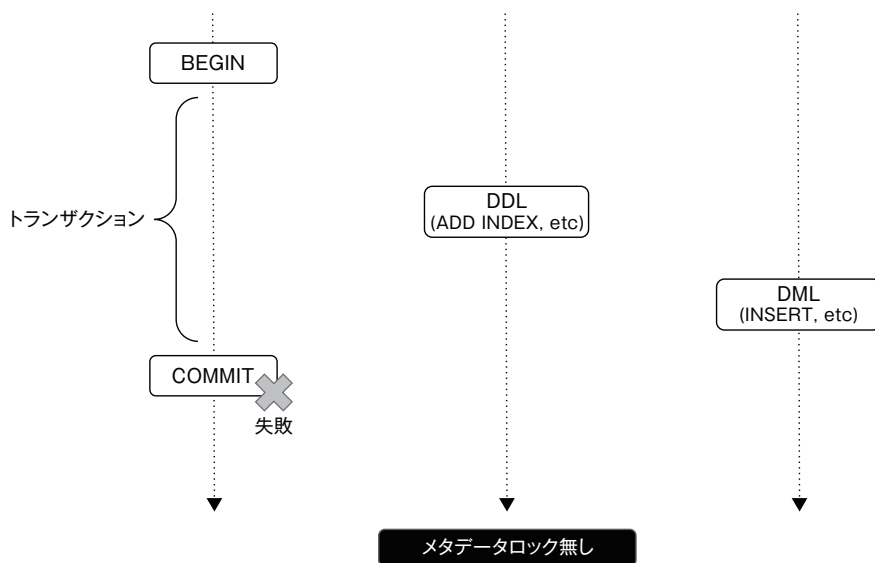
TiDBではDDLステートメントの実行をオンラインで行います。そのため、ワークロードが発生しているテーブルに対してDDLステートメントを実行できます。TiDBの6.3.0より前のバージョンでは、トランザクション処理が行われている間にDDLステートメントが実行されてスキーマが変更されると、そのトランザクションは次のメッセージを出力して失敗していました。

```
ERROR 8028 (HY000): Information schema is changed during the execution of the statement(for example, table definition may be updated by other DDL ran in parallel). If you see this error often, try increasing `tidb_max_delta_schema_count`. [try again later]
```

この問題を解決するために、TiDBではメタデータロックの仕組みが導入されました。メタデータロックはTiDBのバージョン6.3.0で実験的に組み込まれ、6.5.0でGAになりました。TiDBではトランザクションの開始時にメタデータのスナップショットが取得されます。メタデータロックを有効にすると、トランザクション内のDMLステートメントは実行時にメタデータに対してロックを取得するようになります。その結果、DDLステートメントはトランザクションで行われるDMLステートメントの実行を待ってから実行されるようになります。

メタデータロックを使用しない場合の処理の流れを図7.2に示します。上から下に向かっている矢印は時間の経過を表しています。

図7.2 メタデータロック無しの場合



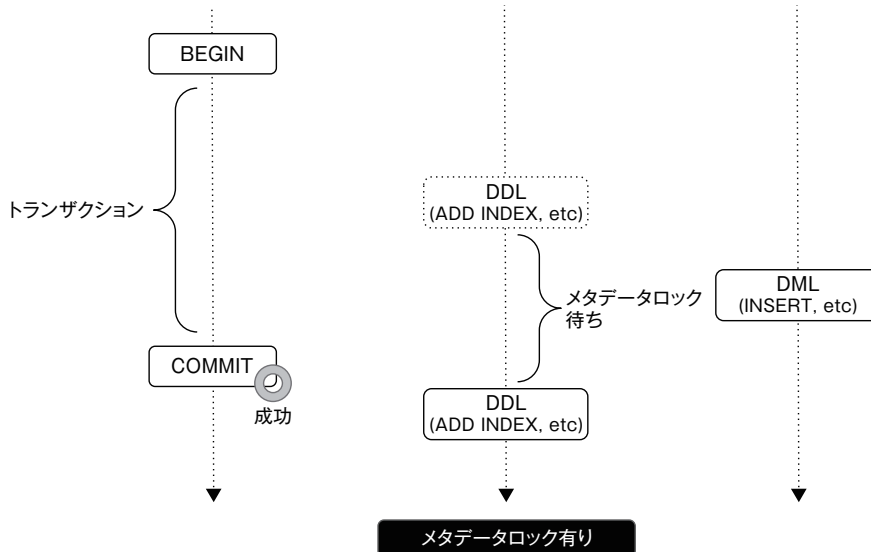
トランザクションが発生している最中に、他のセッションで実行されたDDLステートメントやその後続くDMLステートメントは、トランザクションの終了を待機せずに実行が完了します。しかし、トランザクションは失敗することになります。

次にメタデータロックを使用した場合の処理の流れを図7.3に示します。

トランザクション中のDDLステートメントは、メタデータロックによりトランザクションが完了するまで待機します。トランザクションが完了するとDDLステートメントが実行されます。その間のDMLステートメントはブロックされることなく実行が完了します。このようにメタデータロックを使用することで、トランザクションがエラーになることなくDDLステートメントを実行できます。

メタデータロックにより待機しているDDLステートメントの詳細はmysqlスキーマ内のtidb_mdls_viewテーブルで確認できます。次の例はトランザクション処理中にALTER TABLEステートメントを実行した際の表示です。

図7.3 メタデータロック有りの場合



メタデータロックの確認

```
mysql> SELECT * FROM mysql.tidb_md1_view \G
***** 1. row *****
  job_id: 116
  db_name: test
  table_name: t
    query: ALTER TABLE t ADD INDEX idx(a)
  session_id: 752877578
  start_time: 2025-02-06 03:24:56.153000
  SQL_DIGESTS: ["begin","insert into `t` values ( ? )"]
1 row in set (0.01 sec)
```

queryの内容が待機中のDDLステートメントです。session_idはトランザクションを実行しているセッションIDで、SQL_DIGESTSはトランザクション中に実行されたDMLステートメントの内容です。

待機中のDDLステートメントを即時実行する必要がある場合は、どのトランザクションによって待機しているのかをtidb_md1_viewから確認し、トランザクションをコミットまたはロールバックしてください。トランザクションのコミットはCOMMITステートメント、ロールバックはROLLBACKステートメントを実行します。それでも解決できない場合はトランザクションをKILLで強制終了できます。

トランザクションの強制終了

```
mysql> KILL TIDB QUERY 752877578;
Query OK, 0 rows affected (0.01 sec)
```

メタデータロックはデフォルトで有効になっています。メタデータロックを無効にする場合はシステム変数のtidb_enable_metadata_lockをOFFに設定します。

メタデータロックの無効化

```
mysql> SET GLOBAL tidb_enable_metadata_lock = OFF;  
Query OK, 0 rows affected (0.01 sec)
```

tidb_enable_metadata_lockの設定を変更する際には、TiDB内でDDLステートメントが実行されていないことを確認してください。DDLステートメント実行中に設定を変更すると、データの不整合が発生する可能性があります。

メタデータロックの挙動について、実際のTiDBとMySQLに対してデモを行うTiDB Quick Demo - Reduce Index Locking During Online DDL^{注2}の動画が公開されていますので、よりメタデータロックについて理解を深めたい場合はぜひ参照してください。

7-5

TTLを使用したデータの自動削除

TiDBにはデータのライフサイクルを効率的に管理するための自動データ削除機能が備わっています。この機能は特定のデータを格納した後に一定期間が経過すると、データを自動的に削除する仕組みであり、データの保持期間はTTL(Time to Live)の設定で行います。この機能を利用することで不要になった古いデータを効率的にクリーンアップでき、ストレージの有効活用につながります。たとえば、一時的なデータを格納するテーブルや一定期間を過ぎたログデータを自動的に削除したい場合に、TTLによる自動削除は特に役に立つでしょう。

TTLの設定

TTLはテーブルを作成する際に設定するか、TTLが設定されていない既存のテーブルに対して後から設定できます。また、すでにTTLが設定されている場合は後から削除できます。

TTLの設定はCREATE TABLE ステートメントまたはALTER TABLE ステートメントを実行して行います。次の例はcreated_at より3日経つと削除されるTTLの設定方法です。

TTLの設定

```
mysql> CREATE TABLE ttl_table (  
-> id int PRIMARY KEY,  
-> created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
-> ) TTL = created_at + INTERVAL 3 DAY;  
Query OK, 0 rows affected (0.08 sec)
```

注2 <https://www.youtube.com/watch?v=i49pXPGWbwU>

TTLを変更する場合はALTER TABLE ステートメントを実行します。保存期間を先ほど設定した3日から7日に変更する場合は次のように実行します。

TTLの変更

```
mysql> ALTER TABLE ttl_table TTL = created_at + INTERVAL 7 DAY;  
Query OK, 0 rows affected (0.08 sec)
```

TTLの設定を削除する場合はALTER TABLE ステートメント実行時にREMOVE TTLを使用します。

TTLの削除

```
mysql> ALTER TABLE ttl_table REMOVE TTL;  
Query OK, 0 rows affected (0.07 sec)
```

TTLのバックグラウンドジョブの設定

TiDBの内部ではTTLによる期限切れデータの削除をバックグラウンドジョブとして実行します。この削除ジョブはデフォルトでは1時間ごとに起動されるため、TTLの期限を過ぎたデータが即座に削除されるわけではありません。この点は運用時に注意が必要です。削除ジョブの実行間隔はTTL_JOB_INTERVALの設定を変更することでカスタマイズ可能です。たとえば、削除ジョブの頻度を高めてデータの即時性を重視する場合や、負荷を軽減するために実行間隔を長くしたい場合にこの設定が役立ちます。なお、TiDB Cloud ServerlessではTTL_JOB_INTERVALの変更はできません。

TTLのバックグラウンドジョブの設定

```
mysql> ALTER TABLE ttl_table TTL_JOB_INTERVAL = '24h';  
Query OK, 0 rows affected (0.07 sec)
```

さらに、削除ジョブの実行時間を特定のスケジュールウィンドウに限定できます。たとえば、システムの利用が少ない深夜(午前1時から3時など)の時間帯に削除ジョブを実行させることで、通常のクエリ処理に影響を与えずにデータを削除できます。

TTLのバックグラウンドジョブのスケジュールウィンドウ設定

```
mysql> SET @@global.tidb_ttl_job_schedule_window_start_time = '01:00 +0900';  
Query OK, 0 rows affected (0.04 sec)  
  
mysql> SET @@global.tidb_ttl_job_schedule_window_end_time = '03:00 +0900';  
Query OK, 0 rows affected (0.05 sec)
```

削除ジョブの実行を無効にするには、テーブルに対してTTL_ENABLEをOFFに設定します。

TTLの削除ジョブの無効化

```
mysql> ALTER TABLE ttl_table TTL_ENABLE = 'OFF';
```

また、TiDB全体で削除ジョブを停止する場合はシステム変数tidb_ttl_job_enableをOFFにしてください。

TiDB全体でTTLの削除ジョブの無効化

```
mysql> SET @@global.tidb_ttl_job_enable = OFF;
```

TiDB全体の削除ジョブの実行数はシステム変数tidb_ttl_running_tasksで設定できます。デフォルトは-1になります。これはTiKVのノード数になることを意味しています。

TiDB全体でTTLの削除ジョブの実行数の設定

```
mysql> SET @@global.tidb_ttl_running_tasks = 10;
```

削除ジョブの履歴はmysqlスキーマ内のtidb_ttl_job_historyテーブルに記録されます。実際に処理された件数はsummary_textカラム内に記録されます。

TTLの削除ジョブの履歴の確認

```
mysql> SELECT * FROM mysql.tidb_ttl_job_history limit 1 \G
***** 1. row *****

      job_id: 7b6f5348c9e142ef847dcfbfb27c043
     table_id: 111
parent_table_id: 111
   table_schema: test
      table_name: ttl_table
partition_name: NULL
   create_time: 2024-12-01 23:32:54
   finish_time: 2024-12-01 23:33:03
      ttl_expire: 2024-12-01 23:32:54
summary_text: {"total_rows":10,"success_rows":10,"error_rows":0, ...}
expired_rows: 10
deleted_rows: 10
error_delete_rows: 0
           status: finished
1 row in set (0.01 sec)
```

7-6

ホットスポットの回避

TiDBにおいて、データはリージョンと呼ばれる単位で分割されてストレージクラスターに分散されて配置されます。そのため、通常であればデータの読み取りや書き込みは分散され、負荷が分散されるアーキテクチャーになります。ただし特定のシナリオ、たとえばデータ処理が特定のリージョンに偏ってしまうといった場合に、ホットスポットと呼ばれる負荷が集中する点が形成される可能性があります。

主キーが単調増加するテーブルがあったとしましょう。このテーブルに対してINSERTステートメン

トを繰り返した場合どうなるでしょうか。TiDBのリージョンはキーの範囲で分割されるため、単調増加するキーの場合は特定のリージョンに対してのみ書き込みが行われることになります。

この問題は主キーとしてAUTO_INCREMENTを使用している場合によく発生します。そのため、TiDBではAUTO_INCREMENTの挙動がMySQLとは少し異なっており、なるべくホットスポットを回避するように設計されています(後述する方法でMySQLと同じ挙動にできます)。また、TiDBではAUTO_RANDOMという機能が実装されており、ランダムかつ一意な値を割り当てることが可能です。TiDBでAUTO_INCREMENTを使用する場合は、代わりにAUTO_RANDOMを使用することを検討してください。

TiDBのAUTO_INCREMENT

TiDBのAUTO_INCREMENTは、各TiDBサーバーで別々の範囲の値をキャッシュします。TiDBサーバーAでは1から30000まで、TiDBサーバーBでは30001から60000までといったように、各TiDBサーバーごとに単調増加を保証します。

実際にAUTO_INCREMENTの挙動を確認するために、TiDB PlaygroundでTiDBサーバー2台を起動します。4000番ポートと4001番ポートのTiDBサーバーが起動します。

TiDBサーバー2台の起動

```
$ tiup playground --db 2
...
(省略)
...
Connect TiDB:  mysql --comments --host 127.0.0.1 --port 4000 -u root
Connect TiDB:  mysql --comments --host 127.0.0.1 --port 4001 -u root
...
```

動作確認のためにAUTO_INCREMENTを使用したテーブルを作成します。4000番ポートのTiDBサーバーに接続します。

テスト用データベースとテーブルの作成

```
$ mysql --comments --host 127.0.0.1 --port 4000 -u root

mysql> USE test;
Database changed

mysql> CREATE TABLE auto_incr_test (
  -> id INT PRIMARY KEY AUTO_INCREMENT,
  -> c INT);
Query OK, 0 rows affected (0.52 sec)
```

データを挿入します。

データの挿入

```
mysql> INSERT INTO auto_incr_test (c) VALUES (1), (2), (3);
Query OK, 3 rows affected (0.01 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

現時点でのデータは次のような状態になります。

データの確認

```
mysql> SELECT * FROM auto_incr_test;
+----+-----+
| id | c |
+----+-----+
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
+----+-----+
3 rows in set (0.00 sec)
```

次に4001番ポートのTiDBサーバーに接続して同じテーブルにデータを挿入します。

データの挿入

```
$ mysql --comments --host 127.0.0.1 --port 4001 -u root

mysql> USE test;
Database changed

mysql> INSERT INTO auto_incr_test (c) VALUES (4), (5), (6);
Query OK, 3 rows affected (0.01 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

データを確認します。

データの確認

```
mysql> SELECT * FROM auto_incr_test ORDER BY id;
+-----+-----+
| id   | c |
+-----+-----+
| 1    | 1 |
| 2    | 2 |
| 3    | 3 |
| 30001 | 4 |
| 30002 | 5 |
| 30003 | 6 |
+-----+-----+
6 rows in set (0.00 sec)
```

このようにそれぞれのTiDBサーバーで別々の範囲の値が割り当てられていることが確認できます。

▶ 次に採番される値の確認

AUTO_INCREMENTで次に採番される値を確認するにはINFORMATION_SCHEMA内のTABLESを参照します。

```

次に採番される値の確認
mysql> SELECT AUTO_INCREMENT FROM INFORMATION_SCHEMA.TABLES
       -> WHERE TABLE_SCHEMA = 'test' AND TABLE_NAME = 'auto_incr_test';
+-----+
| AUTO_INCREMENT |
+-----+
|                4 |
+-----+
1 row in set (0.00 sec)

```

この値は接続しているTiDBサーバーでの値になります。上記の結果は4000番ポートのTiDBの出力です。4001番ポートのTiDBサーバーに接続して同じSQLを実行した場合は次のように表示されます。

```

次に採番される値の確認
mysql> SELECT AUTO_INCREMENT FROM INFORMATION_SCHEMA.TABLES
       -> WHERE TABLE_SCHEMA = 'test' AND TABLE_NAME = 'auto_incr_test';
+-----+
| AUTO_INCREMENT |
+-----+
|           30004 |
+-----+
1 row in set (0.00 sec)

```

▶ 再起動すると払い出される範囲が変更される

TiDBサーバーが再起動した場合、以前担当していた値の範囲は使用されなくなり、新しい値の範囲が払い出されます。そのため、TiDBサーバーが頻繁に再起動するような状況では払い出される値が枯渇する可能性があります。もしそのような状況が想定される場合には、対策としてBIGINTを利用することを検討してください。

4000番ポートで起動しているTiDBサーバーをスケールインで停止し、再度スケールアウトで起動します。

```

TiDBサーバーのスケールイン
$ tiup playground display
...
1078658  tidb      26m12.98819134s
1078684  tidb      26m12.784207867s
...

$ tiup playground scale-in --pid 1078658
scale in tidb success

$ tiup playground scale-out --db 1

```

再度 4000 番ポートの TiDB サーバーに接続し、データを挿入して確認します。

データの挿入

```
$ mysql --comments --host 127.0.0.1 --port 4000 -u root

mysql> USE test;
Database changed

mysql> INSERT INTO auto_incr_test (c) VALUES (7);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM auto_incr_test ORDER BY id;
+-----+-----+
| id    | c    |
+-----+-----+
| 1     | 1    |
| 2     | 2    |
| 3     | 3    |
| 30001 | 4    |
| 30002 | 5    |
| 30003 | 6    |
| 60001 | 7    |
+-----+-----+
7 rows in set (0.00 sec)
```

新しく 60001 から採番されることが確認できます。

MySQLと同じ挙動のAUTO_INCREMENT

MySQL における AUTO_INCREMENT の挙動を TiDB でも再現できます。つまり、TiDB サーバーが複数あっても全体で単調増加する AUTO_INCREMENT を実現できます。

全体で単調増加する AUTO_INCREMENT を実現するためには、AUTO_ID_CACHE 1 を設定します。

全体で単調増加するAUTO_INCREMENTの設定

```
mysql> USE test;
Database changed

mysql> CREATE TABLE auto_incr_test2 (
  -> id INT PRIMARY KEY AUTO_INCREMENT,
  -> c INT) AUTO_ID_CACHE 1;
```

それぞれの TiDB サーバーからデータを挿入して確認します。

データの挿入

```
$ mysql --comments --host 127.0.0.1 --port 4000 -u root
```

```
mysql> USE test;
Database changed

mysql> INSERT INTO auto_incr_test2 (c) VALUES (1), (2), (3);
```

```
$ mysql --comments --host 127.0.0.1 --port 4001 -u root

mysql> USE test;
Database changed

mysql> INSERT INTO auto_incr_test2 (c) VALUES (4), (5), (6);
```

データを確認します。

```
データの確認
mysql> SELECT * FROM auto_incr_test2 ORDER BY id;
+----+-----+
| id | c   |
+----+-----+
| 1  | 1   |
| 2  | 2   |
| 3  | 3   |
| 4  | 4   |
| 5  | 5   |
| 6  | 6   |
+----+-----+
6 rows in set (0.00 sec)
```

全体で単調増加する AUTO_INCREMENT が実現されていることが確認できます。

AUTO_ID_CACHE 1 が指定された場合、単一の TiDB サーバーが中央集権的な役割を持って値を採番するようになります。他の TiDB サーバーは中央集権的な役割を持つ TiDB サーバーから値を取得するようになります。

AUTO_RANDOM の使用

AUTO_RANDOM はランダムかつ一意な値を割り当てる機能です。AUTO_RANDOM を使用することで、ホットスポットを回避しつつ、ランダムな値を採番できます。AUTO_RANDOM は 64 ビットの値を払い出すため、BIGINT 型を使用する必要があります。また、クラスター化テーブルである必要があります。クラスター化テーブルについては第 2 章内の「TiDB クラスター」を参照してください。

```
AUTO_RANDOM の使用
mysql> USE test;
Database changed
```

```
mysql> CREATE TABLE auto_rand_test (
  -> id BIGINT PRIMARY KEY AUTO_RANDOM, c int );
Query OK, 0 rows affected, 1 warning (0.09 sec)
```

AUTO_INCREMENT と同様に 2 台の TiDB サーバーからデータを挿入します。4000 番ポートの TiDB サーバーは 1 から 3 までの値を、4001 番ポートの TiDB サーバーは 4 から 6 までの値を挿入します。

データの挿入

```
mysql> USE test;
Database changed

mysql> INSERT INTO auto_rand_test (c) VALUES (1);
mysql> INSERT INTO auto_rand_test (c) VALUES (2);
mysql> INSERT INTO auto_rand_test (c) VALUES (3);

mysql> INSERT INTO auto_rand_test (c) VALUES (4);
mysql> INSERT INTO auto_rand_test (c) VALUES (5);
mysql> INSERT INTO auto_rand_test (c) VALUES (6);
```

データを確認します。

データの確認

```
mysql> SELECT * FROM auto_rand_test ORDER BY c;
+-----+-----+
| id          | c    |
+-----+-----+
| 5476377146882523137 | 1    |
| 3170534137668829186 | 2    |
| 7493989779944505347 | 3    |
| 4035225266123994417 | 4    |
| 4323455642275706162 | 5    |
| 8646911284551382323 | 6    |
+-----+-----+
6 rows in set (0.01 sec)
```

このようにランダムな値が採番されていることが確認できます。

AUTO_RANDOM は使用するにあたり、いくつかの制限があります。制限についての詳細はドキュメントの [AUTO_RANDOM^{注3}](https://docs.pingcap.com/tidb/stable/auto-random#restrictions) を参照してください。

▶ AUTO_RANDOM の値について

AUTO_RANDOM は 64 ビットの値が払い出されます。その構造は表 7.4 のようになります。

注3 <https://docs.pingcap.com/tidb/stable/auto-random#restrictions>

表7.4 AUTO_RANDOMの値の構造

名前	ビットの長さ	説明
符号ビット	0 or 1	UNSIGNED属性がある場合は長さが0
予約ビット	64 - 予約ビット	値は固定で0
シャードビット	1から15	トランザクションの開始時刻から計算されるハッシュ値
自動増分ビット	予約ビット - 符号ビット - シャードビット	自動増分の値

予約ビットは32から64の範囲で指定し、デフォルトは64です。シャードビットはデフォルトで5になります。それぞれ変更する場合はAUTO_RANDOM設定時に次のように指定します。

```
... AUTO_RANDOM(シャードビット, 予約ビット)
```

デフォルトは次の指定と同じになります。

```
... AUTO_RANDOM(5, 64)
```

具体的な値を表示して確認します。先ほどのauto_rand_testテーブルのデータを参照します。

AUTO_RANDOMの値の確認

```
mysql> SELECT LPAD(BIN(id),64,0) as RAND_BIT
-> FROM auto_rand_test ORDER BY c;

+-----+
| RAND_BIT |
+-----+
| 0100110000000000000000000000000000000000000000000000000000000001 |
| 0010110000000000000000000000000000000000000000000000000000000010 |
| 0110100000000000000000000000000000000000000000000000000000000011 |
| 0011100000000000000000000000000000000000000000000000000000000001 |
| 0011110000000000000000000000000000000000000000000000000000000001 |
| 0111100000000000000000000000000000000000000000000000000000000001 |
+-----+
6 rows in set (0.00 sec)
```

先頭の1ビットは符号ビットで、次の5ビットがシャードビット、残りの58ビットが自動増分ビットになります。

自動増分ビットに注目すると、AUTO_INCREMENTと同様に単調増加していることが確認できます。

自動増分ビットの確認

```
mysql> SELECT c, CONV(RIGHT(LPAD(BIN(id),64,0), 58), 2, 10) as RAND_BIT
-> FROM auto_rand_test ORDER BY c;

+-----+-----+
| c | RAND_BIT |
+-----+-----+
| 1 | 1 |
| 2 | 2 |
```

```
| 3 | 3 |
| 4 | 30001 |
| 5 | 30002 |
| 6 | 30003 |
+-----+
6 rows in set (0.01 sec)
```

シャードビットはTiKVのノード数を元に算出することがベストプラクティスです。算出方法は $\log(2, \text{TiKVのノード数})$ です。たとえばTiKVのノード数が16である場合はシャードビットとしては4が適切となります。シャードビットを増やした場合はランダム性は高まりますが、その分自動増分ビットが減少するため、採番できる値の範囲が狭くなりますのでバランスを考慮して設定してください。

■ 非クラスター化テーブルにおけるSHARD_ROW_ID_BITSの使用

非クラスター化テーブル(主キーがCLUSTEREDではない、または主キーのないテーブル)では、TiDBは各行に対してRowIDを暗黙的に付与します。RowIDは単調増加するIDになるため、INSERTステートメントが大量に実行された場合に単一のリージョンに対して書き込みが集中し、ホットスポットが発生する可能性があります。

このような課題を解決するために、RowIDの値を単調増加ではなくランダムに払い出す仕組みとしてSHARD_ROW_ID_BITSを使用できます。SHARD_ROW_ID_BITSを使用するとRowIDの先頭ビットがシャードビットになり、ランダムな値が採番されるようになります。

SHARD_ROW_ID_BITSはテーブル作成時に指定します。またはALTER TABLEステートメントを使用して後から設定できます。SHARD_ROW_ID_BITSの値は0から15までの範囲で指定できます。デフォルトは0でシャードしません。

SHARD_ROW_ID_BITSの使用

```
mysql> USE test;
Database changed

mysql> CREATE TABLE shard_row_id_test (c int) SHARD_ROW_ID_BITS = 4;
Query OK, 0 rows affected (0.09 sec)
```

データを挿入し、RowIDの値を確認します。RowIDは`_tidb_rowid`カラムに格納されています。

データの挿入

```
mysql> INSERT INTO shard_row_id_test (c) VALUES (1);
Query OK, 1 row affected (0.02 sec)

mysql> INSERT INTO shard_row_id_test (c) VALUES (2);
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO shard_row_id_test (c) VALUES (3);
```


7-7

ベクトル検索機能

TiDBはTiDB Cloud ServerlessおよびTiDB Self-Managed、TiDB Communityのバージョン8.4.0からベクトル検索を実験的にサポートしています。画像や音声、テキストなどのデータをベクトル化してベクトル型のカラムに保存して検索可能です。本節ではベクトル検索の簡単な使い方を説明します。

ベクトル検索が注目されている背景

AIが業界を変革し続ける中で、特に複雑で高次元なデータを効率的に扱えるデータベースの必要性が急速に高まっています。RDBMSは従来からテキストや数値のような構造化データを得意としてきましたが、AIアプリケーションで頻繁に用いられる画像、音声、さらにはこれらを埋め込みベクトルに変換したデータの管理や検索には適していません。さらに、AIが生成するデータを解釈し活用するためには、データの意味を深く理解するセマンティックな検索機能が不可欠です。

このような背景から、AIモデルによって生成されたベクトル埋め込みデータをデータベース内で直接管理し、構造化データと統合して扱えるようにするためにデータベースにベクトル検索機能を追加することが近年注目されています。

TiDBにおけるベクトル検索

TiDBではベクトルデータはVECTOR型または次元数を指定したVECTOR(N)型で定義されたカラムに格納します。次元数は最大16,383次元まで対応しています。VECTOR型のカラムは異なる次元のベクトルデータを格納できますが、HNSW^{注4}インデックスが使用できません。HNSWインデックスを使用する場合はVECTOR(N)型を使用してください。また、HNSWインデックスを使用する場合はTiFlashへのデータコピーが必要になります。

実際にベクトル検索を確認するためにテーブルを作成します。ここでは3次元のベクトルデータを格納するテーブルを作成します。

ベクトルデータを格納するテーブルの作成

```
mysql> use test;
Database changed

mysql> CREATE TABLE embedded_documents (
->   id      INT      PRIMARY KEY,
```

注4 https://en.wikipedia.org/wiki/Hierarchical_navigable_small_world

```
-> document TEXT,
-> embedding VECTOR(3)
-> );
Query OK, 0 rows affected (0.12 sec)
```

サンプルデータとして3件のベクトルデータを挿入します。

ベクトルデータを格納するテーブルにデータを挿入

```
mysql> INSERT INTO embedded_documents
-> VALUES
-> (1, 'dog', '[1,2,1]'),
-> (2, 'fish', '[1,2,4]'),
-> (3, 'tree', '[1,0,0]');
Query OK, 3 rows affected (0.02 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM embedded_documents;
+----+-----+-----+
| id | document | embedding |
+----+-----+-----+
| 1 | dog      | [1,2,1]   |
| 2 | fish     | [1,2,4]   |
| 3 | tree     | [1,0,0]   |
+----+-----+-----+
3 rows in set (0.02 sec)
```

TiDBではベクトル距離を求める関数がいくつか提供されています(表7.5)。

表7.5 ベクトル距離関数

ベクトル距離関数名	説明
VEC_L1_DISTANCE	2つのベクトル間のL1距離(マンハッタン距離)を計算する
VEC_L2_DISTANCE	2つのベクトル間のL2距離(ユークリッド距離)を計算する
VEC_COSINE_DISTANCE	2つのベクトル間のコサイン距離を計算する
VEC_NEGATIVE_INNER_PRODUCT	2つのベクトルの内積の負数を計算する

それぞれのベクトル距離関数を使用した例は次の通りです。

VEC_L1_DISTANCE

```
mysql> SELECT document,
-> VEC_L1_DISTANCE(embedding, '[1,2,3]') AS VEC_L1_DISTANCE
-> FROM embedded_documents;
+-----+-----+
| document | VEC_L1_DISTANCE |
+-----+-----+
| dog      | 2               |
| fish     | 1               |
| tree     | 5               |
+-----+-----+
```

```
+-----+-----+
3 rows in set (0.01 sec)
```

```
VEC_L2_DISTANCE
mysql> SELECT document,
  -> VEC_L2_DISTANCE(embedding, '[1,2,3]') AS VEC_L2_DISTANCE
  -> FROM embedded_documents;
```

```
+-----+-----+
| document | VEC_L2_DISTANCE |
+-----+-----+
| dog      |                2 |
| fish     |                1 |
| tree     | 3.605551275463989 |
+-----+-----+
3 rows in set (0.01 sec)
```

```
VEC_COSINE_DISTANCE
mysql> SELECT document,
  -> VEC_COSINE_DISTANCE(embedding, '[1,2,3]') AS VEC_COSINE_DISTANCE
  -> FROM embedded_documents;
```

```
+-----+-----+
| document | VEC_COSINE_DISTANCE |
+-----+-----+
| dog      | 0.12712843905603044 |
| fish     | 0.00853986601633272 |
| tree     | 0.7327387580875756 |
+-----+-----+
3 rows in set (0.02 sec)
```

```
VEC_NEGATIVE_INNER_PRODUCT
mysql> SELECT document,
  -> VEC_NEGATIVE_INNER_PRODUCT(embedding, '[1,2,3]') AS VEC_NEGATIVE_INNER_PRODUCT
  -> FROM embedded_documents;
```

```
+-----+-----+
| document | VEC_NEGATIVE_INNER_PRODUCT |
+-----+-----+
| dog      | -8 |
| fish     | -17 |
| tree     | -1 |
+-----+-----+
3 rows in set (0.02 sec)
```

このようにベクトルデータを格納し、ベクトル距離関数を使用することでベクトル検索できます。
LlamaIndexとの組み合わせ^{注5}やLangChainとの組み合わせ^{注6}のチュートリアルも用意されていますの

注5 <https://docs.pingcap.com/tidbcloud/vector-search-integrate-with-llamaindex>

注6 <https://docs.pingcap.com/tidbcloud/vector-search-integrate-with-langchain>

で、興味のある方はぜひチュートリアルを実行してみてください。

ベクトル距離関数以外にもさまざまなベクトル関数が用意されています。詳細はドキュメントの [Vector Functions and Operators](#)^{注7} を参照してください。

7-8 TerraformによるTiDB Cloudの管理

Infrastructure as Code (IaC) を使用して構成を管理しておくことは、環境の再現性や変更管理を行う上で有効です。Terraform^{注8} は IaC のツールの一つで、コードでインフラストラクチャを管理できます。

近年、Terraform のライセンスは Mozilla Public License v2.0 (MPL-2.0) から Business Source License v1.1 (BUSL-1.1) に変更されました。そのため、ライセンスが変更されたことを受けフォークしてオープンソース化された OpenTofu^{注9} も公開されています。

TiDB Cloud は Terraform 用のプロバイダーが公開されており、TiDB Cloud Serverless や TiDB Cloud Dedicated のクラスター作成やバックアップ設定などを管理できます。

本節では Terraform を使用して TiDB Cloud のクラスターを作成する手順を説明します。Terraform のインストールはドキュメントの [Install Terraform](#)^{注10} を参照してください。本節の動作確認は Terraform のバージョン v1.10.3 および OpenTofu のバージョン v1.8.7 を使用しています。OpenTofu を使用する場合はコマンドラインの terraform を tofu に置き換えてください。OpenTofu のインストール方法はドキュメントの [Installing OpenTofu](#)^{注11} を参照してください。

TiDB CloudのAPIキーの取得

Terraform を実行してリソースを操作するために、TiDB Cloud の API キーを取得します。API キーは TiDB Cloud のコンソール画面から「Organization Settings」を選択し、API Keys 画面から行います。API キーの作成は Organization Owner のみが行えます。API キーを作成すると Public Key と Private Key が表示されますので安全に保存してください。

注7 <https://docs.pingcap.com/tidbcloud/vector-search-functions-and-operators>

注8 <https://www.terraform.io/>

注9 <https://opentofu.org/>

注10 <https://learn.hashicorp.com/tutorials/terraform/install-cli>

注11 <https://opentofu.org/docs/intro/install/>

Terraformの定義

取得したAPIキーを使用してプロバイダーの定義をします。プロバイダーの定義をするためにprovider.tfファイルを作成します(リスト7.3)。

リスト7.3 プロバイダー定義

```
provider.tf
provider "tidbcloud" {
  public_key = "xxxxxxxxx"
  private_key = "xxxxxxxxx"
  sync       = true ❶
}
```

public_keyとprivate_keyは直接書き込むこともできますが、環境変数でも設定可能です。その場合は次のように環境変数を定義してください。

環境変数の設定

```
export TIDBCLOUD_PUBLIC_KEY="xxxxxxxxx"
export TIDBCLOUD_PRIVATE_KEY="xxxxxxxxx"
```

syncはクラスター操作を同期的に行うか非同期的に行うかを指定します(リスト7.3の❶)。trueに指定するとTerraformはクラスター操作の完了を待ちます。falseを指定するとTerraformはクラスター操作の完了を待たずに終了します。デフォルトはfalseです。

次にクラスターの定義をします。main.tfファイルを作成し、tidbcloud_clusterリソースを定義します(リスト7.4)。リソース定義内でプロジェクトIDが必要になりますが、tidbcloud_projectsを使用して自動で取得しています(リスト7.4の❶)。TiDB Cloud Serverlessのクラスターを作成する場合はcluster_typeをDEVELOPERに指定します(リスト7.4の❷)。TiDB Cloud Dedicatedのクラスターを作成する場合はcluster_typeをDEDICATEDに指定してください。

リスト7.4 Terraformのメイン定義

```
main.tf
terraform {
  required_providers {
    tidbcloud = {
      source = "tidbcloud/tidbcloud"
      version = "~> 0.3.0"
    }
  }
}

data "tidbcloud_projects" "my_projects" {
  page_size = 10
}
```

```

}

locals {
  project_name = "default_project"
  project_id = { —❶
    value = element([for s in data.tidbcloud_projects.my_projects.items : s.id if s.name == local.❷
project_name], 0)
  }
}

resource "tidbcloud_cluster" "my_serverless_cluster" {
  project_id    = local.project_id.value
  name          = "cluster-from-terraform"
  cluster_type  = "DEVELOPER" —❷
  cloud_provider = "AWS"
  region        = "ap-northeast-1"
  config = {
    root_password = "my_secret_password"
  }
}

output "connection_strings" {
  value = lookup(tidbcloud_cluster.my_serverless_cluster.status, "connection_strings")
}

```

なお、本手順ではTerraformの`terraform.tfstate`ファイルの保存先の定義はしませんが、実際の運用では`backend`ブロックを使用して保存先を指定することをおすすめします。詳細はドキュメントのBackend block configuration overview^{注12}を参照してください。

`provider.tf`と`main.tf`を作成したらTerraformを初期化します。

Terraformの初期化

```
$ terraform init
```

```
...
```

```
(省略)
```

```
...
```

```
Terraform has been successfully initialized!
```

You may now begin working with Terraform. Try running "terraform plan" to see any changes that are required for your infrastructure. All Terraform commands should now work.

If you ever set or change modules or backend configuration for Terraform, rerun this command to reinitialize your working directory. If you forget, other commands will detect it and remind you to do so if necessary.

注12 <https://developer.hashicorp.com/terraform/language/backend>

続いてterraform planコマンドを実行して作成されるリソースを確認します。

```
作成されるリソースの確認
$ terraform plan
Terraform used the selected providers to generate the following execution plan. ...
+ create

Terraform will perform the following actions:

# tidbcloud_cluster.serverless_tier_cluster will be created
+ resource "tidbcloud_cluster" "serverless_tier_cluster" {
  + cloud_provider = "AWS"
  + cluster_type   = "DEVELOPER"
  + config         = {
    + components = (known after apply)
    + port       = (known after apply)
    + root_password = "my_secret_password"
  }
  + create_timestamp = (known after apply)
  + id               = (known after apply)
  + name             = "cluster-from-terraform"
  + project_id       = "xxxxxxxxxxxxxxxxxxxxxxxxxxxx"
  + region           = "ap-northeast-1"
  + status           = (known after apply)
}

Plan: 1 to add, 0 to change, 0 to destroy.

Changes to Outputs:
+ connection_strings = (known after apply)
```

表示内容に問題がなければterraform applyコマンドを実行してリソースを作成します。確認が求められますのでyesを入力します。確認をスキップしたい場合は-auto-approve オプションを指定してください。

```
リソースの作成
$ terraform apply
...
(省略)
...
tidbcloud_cluster.serverless_tier_cluster: Creating...
tidbcloud_cluster.serverless_tier_cluster: Still creating... [10s elapsed]
tidbcloud_cluster.serverless_tier_cluster: Creation complete after 14s ...

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

Outputs:
```



```

connection_strings = {
  "default_user" = "xxxxxxxxxx.root"
  "standard" = {
    "host" = "gateway01.ap-northeast-1.prod.aws.tidbcloud.com"
    "port" = 4000
  }
  "vpc_peering" = {
    "host" = "gateway01-privatelink.ap-northeast-1.prod.aws.tidbcloud.com"
    "port" = 4000
  }
}

```

リソースの作成が終わったら connection_strings が出力されますので、設定したパスワードと共に TiDB クラスターに接続してください。

connection_stringsを元に接続

```

$ mysql --host gateway01.ap-northeast-1.prod.aws.tidbcloud.com \
  --user xxxxxxxxxxxx.root --port 4000 --password

```

作成したリソースを削除する場合は terraform destroy コマンドを実行します。

リソースの削除

```

$ terraform destroy

```

クラスター操作以外にバックアップやインポート、リストアの定義も可能です。TiDB Cloud プロバイダーの詳細な使い方は tidbcloud Provider^{注13}のドキュメントを参照してください。

7-9

Advanced StatefulSetを使用したTiDBの構築

KubernetesではStatefulSetリソースを使用してPodを作成すると、Pod名に連続したインデックス番号を割り当てます。たとえばレプリカが3つある場合、Podの名前はpod-0、pod-1、pod-2になります。スケールアウトする場合はインデックス番号を順番に増やしてPodが追加されます。スケールインする場合はインデックス番号が大きい順からPodが削除されます。たとえば、レプリカを4つにスケールアウトするとpod-3が追加されます。レプリカを2つにスケールインするとpod-2が削除されます。

Advanced StatefulSetコントローラーは、このようなStatefulSetリソースの制約を解決してインデックス番号を自由に制御できるようにするためのカスタムコントローラーです。たとえば、レプリカの

注13 <https://registry.terraform.io/providers/tidbcloud/tidbcloud/latest/docs>

数が3つある場合、pod-0、pod-2、pod-3のようにpod-1をスキップしてレプリカの数を持続できます。ローカルストレージを使用していた場合、Podはノードのストレージリソースに関連付けられ自由にスケジュールできません。pod-1が稼働しているノードを維持したまま、別のノードにPodを移行したいといった場合に有効です。

Advanced StatefulSetの有効化

Advanced StatefulSetを有効にするには、カスタムリソース定義(CRD)を作成し、TiDB Operatorの設定を変更する必要があります。まず、Advanced StatefulSetを有効にするためのCRDを作成します。使用するTiDB Operatorのバージョンに応じてURLを変更してください。

Advanced StatefulSetのCRDの作成

```
$ kubectl apply -f \
  https://raw.githubusercontent.com/pingcap/tidb-operator/v1.6.1/manifests/advanced-statefulset-crd.v1.yaml
customresourcedefinition.k8s.io/statefulsets.apps.pingcap.com created
```

次にTiDB Operatorの設定を変更します。すでにTiDB Operatorがインストールされている場合はhelm upgrade コマンドで設定を変更できます。新規にTiDB Operatorをインストールする場合はhelm install コマンドを使用してください。

Advanced StatefulSetの有効化

```
$ helm upgrade tidb-operator pingcap/tidb-operator --version v1.6.1 \
  --set 'features={AdvancedStatefulSet=true}' \
  --set 'advancedStatefulset.create=true'
```

Advanced StatefulSetを有効にすると、advanced-statefulset-controllerが別途作成されます。

```
$ kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
advanced-statefulset-controller-68f464945f-wnfjp	1/1	Running	0	34s
tidb-controller-manager-7497c8ccd6-c9wr4	1/1	Running	0	34s

Advanced StatefulSetの使用

Advanced StatefulSetを有効化した後は通常通りTidbClusterリソースを作成します。TiDB Operatorは自動でAdvanced StatefulSetを使用します。そのため、StatefulSetリソースが作成されていないことを確認します。

StatefulSetの確認

```
$ kubectl get tidbcluster
```

NAME	READY	PD	STORAGE	READY	DESIRE	...
tidb-sample	True	pingcap/pd:v8.5.0	1Gi	3	3	...

```
$ kubectl get statefulset
No resources found in default namespace.
```

Advanced StatefulSetを確認します。リソース名はstatefulsets.apps.pingcap.comです。また、省略形でastsとしても使用できます。

Advanced StatefulSetの確認

```
$ kubectl get statefulsets.apps.pingcap.com
```

NAME	DESIRED	CURRENT	AGE
tidb-sample-pd	3	3	16m
tidb-sample-tidb	3	3	15m
tidb-sample-tikv	3	3	15m

インデックスの操作

インデックスを操作する場合はTiDBClusterリソースのアノテーションに特定の値を設定します。サポートされているアノテーションのキーは次の通りです。

- **pd.tidb.pingcap.com/delete-slots**
PDコンポーネント内で削除するインデックス番号を指定します
- **tidb.tidb.pingcap.com/delete-slots**
TiDBコンポーネント内で削除するインデックス番号を指定します
- **tikv.tidb.pingcap.com/delete-slots**
TiKVコンポーネント内で削除するインデックス番号を指定します

アノテーションの値には[0]や[1,3]のように欠番にしたい番号を記載します。

現在3つのPodで稼働しているTiDBコンポーネントのうち、tidb-sample-tidb-1を削除したい場合は次のようにアノテーションを設定します。

インデックスの操作

```
$ kubectl annotate tidbcluster tidb-sample 'tidb.tidb.pingcap.com/delete-slots=[1]'
tidbcluster.pingcap.com/tidb-sample annotated
```

実際にtidb-sample-tidb-1が削除されていることを確認します。レプリカの数を変更していませんので、tidb-sample-tidb-1の代わりにtidb-sample-tidb-3が新しく起動されていることが確認できます。

```
$ kubectl get pod --selector app.kubernetes.io/component=tidb
```

NAME	READY	STATUS	RESTARTS	AGE
tidb-sample-tidb-0	2/2	Running	0	23m
tidb-sample-tidb-2	2/2	Running	0	23m
tidb-sample-tidb-3	2/2	Running	0	41s

元に戻す場合は値を [] にするか、アノテーションを削除します。値を [] に上書きする場合は `--overwrite` オプションが必要になります。アノテーションを削除する場合は `tidb.tidb.pingcap.com/delete-slots-` を指定してください。

インデックスの操作の解除1

```
$ kubectl annotate tidbcluster tidb-sample \  
  'tidb.tidb.pingcap.com/delete-slots=[]' --overwrite  
tidbcluster.pingcap.com/tidb-sample annotated
```

インデックスの操作の解除2

```
$ kubectl annotate tidbcluster tidb-sample tidb.tidb.pingcap.com/delete-slots-  
tidbcluster.pingcap.com/tidb-sample annotated
```

`tidb-sample-tidb-3` が削除され、`tidb-sample-tidb-1` が起動されていることを確認します。

元に戻っていることを確認

```
$ kubectl get pod --selector app.kubernetes.io/component=tidb
```

NAME	READY	STATUS	RESTARTS	AGE
tidb-sample-tidb-0	2/2	Running	0	27m
tidb-sample-tidb-1	2/2	Running	0	114s
tidb-sample-tidb-2	2/2	Running	0	17m

7-10

PrometheusやGrafanaのカスタマイズ

TiUP や TiDB Operator を使用することで、Prometheus や Grafana を使用したモニタリング環境の構築や設定は簡単に行えます。通常であれば、特に設定を変更しなくても十分ですが、場合によっては Prometheus や Grafana の設定をカスタマイズして構築したい場合もあるかもしれません。その場合は必要に応じて設定をカスタマイズしてください。ただし、設定変更は注意深く行ってください。

本節では Prometheus と Grafana のカスタマイズ方法を説明します。なお、TiDB Playground は執筆時点では設定変更に対応していませんので注意してください。

Prometheusのカスタマイズ

Prometheus の Recording rules^{注14} や Alerting rules^{注15} を設定したい場合は設定ファイルを用意し、TiUP のトポロジーファイルで指定します。

注14 https://prometheus.io/docs/prometheus/latest/configuration/recording_rules/

注15 https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/

例としてTiDBサーバーへの接続が100を超えた場合にアラートを出す設定を反映します。まずTiUPを実行するサーバーでルールを記述したファイルを用意します(tidb_sample.rules.yml)。ファイルの名前は*.rules.ymlである必要がありますので注意してください。ファイルを配置するディレクトリは任意の場所で問題ありませんが、TiUP実行ユーザーがアクセスできる必要があります。ここでは/home/tidb/prometheus_rule/ディレクトリを使用します。

```
tidb_sample.rules.yml
groups:
- name: tidb_alert_sample
  rules:
  - alert: TooManyTiDBServerConnection
    expr: max_over_time(tidb_server_connections[5m]) > 100
    for: 5m
    keep_firing_for: 5m
    labels:
      severity: page
    annotations:
      summary: Too many TiDB server connections to { $labels.instance }
```

次に、トポロジーファイル内のmonitoring_serversでルールファイルが保存されているディレクトリをrule_dirで定義します。

```
Prometheusのカスタムルールの適用
monitoring_servers:
- host: 10.0.0.41
  rule_dir: /home/tidb/prometheus_rule
```

TiUPでPrometheusコンポーネントを作成すると、/home/tidb/prometheus_ruleディレクトリ以下にある*.rules.ymlが定義されたサーバー(例では10.0.0.41)上の/tidb-deploy/prometheus-9090/confディレクトリに転送されます。ディレクトリ内のprometheus.ymlを確認すると下記の設定が記述されていることが確認できます。

```
prometheus.ymlの内容を確認
$ grep --after 1 rule_files /tidb-deploy/prometheus-9090/conf/prometheus.yml
rule_files:
- 'tidb_sample.rules.yml'
```

Prometheusのスクレイプ設定はTiUPにより自動で設定されます。スクレイプ設定をカスタマイズする場合はトポロジーファイル内でadditional_scrape_confを設定してください。次の例はtikv_thread_nonvoluntary_context_switchesおよびtikv_thread_voluntary_context_switchesのメトリクスを削除する設定です。

スクレイプ設定のカスタマイズ

```
monitoring_servers:
- host: 10.0.0.41
  additional_scrape_conf:
    metric_relabel_configs:
      - source_labels: [__name__]
        regex: tikv_thread_nonvoluntary_context_switches|tikv_thread_voluntary_context_switches
        action: drop
```

データの保存期間は `storage_retention` で設定します。デフォルトは30dとなり、メトリクスは30日間保存されます。次の例はメトリクスの保存期間を15日間に変更する設定です。

メトリクス保存期間のカスタマイズ

```
monitoring_servers:
- host: 10.0.0.41
  storage_retention: 15d
```

その他にも `additional_args` を設定して Prometheus の起動時のオプションを変更したり、`external_alertmanagers` を設定して外部の Alertmanager を設定したりなどカスタマイズが可能です。次の例は Prometheus の起動時の引数に `--web.enable-lifecycle` を指定する設定です。

Prometheusの起動時の引数の指定

```
monitoring_servers:
- host: 10.0.0.41
  additional_args:
    - --web.enable-lifecycle
```

その他の設定項目を確認するにはドキュメントの [Topology Configuration File for TiDB Deployment Using TiUP^{注16}](#) を参照してください。

Grafanaのカスタマイズ

Grafana のカスタマイズは Prometheus と同様にトポロジーファイル内で設定します。

Grafana ダッシュボードのログインに必要なユーザー名およびパスワードは、デフォルトで両方とも `admin` になっています。変更する場合には TiUP のトポロジーファイルを次のように設定してください。

Grafanaログインユーザーの変更

```
grafana_servers:
- host: 10.0.0.41
  username: new_admin
  password: new_password
```

注16 https://docs.pingcap.com/tidb/stable/tiup-cluster-topology-reference#monitoring_servers

ログイン認証を無効にする場合はanonymous_enableをtrueにします。

Grafanaのログイン認証を無効にする

```
grafana_servers:
  - host: 10.0.0.41
    anonymous_enable: true
```

その他のGrafanaの設定^{注17}はconfigで設定可能です。次の例はGrafanaのログレベルをwarningに変更し、出力をJSON形式に変更します。

Grafana設定のカスタマイズ

```
grafana_servers:
  - host: 10.0.0.41
    config:
      log.file.level: warning
      log.file.format: json
```

TiUPは運用に役立つGrafanaダッシュボードを自動で多数セットアップします。それだけでも十分なのですが、ダッシュボードを追加したい場合は次の手順で可能です。

追加したいGrafanaダッシュボードのJSONファイルを用意します。grafana.comにて公開されているダッシュボード^{注18}から取得するか、独自に作成してください。JSONファイルは任意のディレクトリ内に配置してください。ただし、TiUP実行ユーザーがアクセスできる必要があります。また、ファイルの名前は*.jsonにする必要があります。配置したらトポロジーファイル内のdashboard_dirにディレクトリ名を設定してください。

追加ダッシュボードの設定

```
grafana_servers:
  - host: 10.0.0.41
    dashboard_dir: /home/tidb/grafana_dashboard
```

TiUPでGrafanaコンポーネントを作成すると、定義されたサーバー上の/tidb-deploy/grafana-3000/provisioning/dashboardsディレクトリ以下にダッシュボードのJSONファイルが転送されます。その後Grafanaの起動時に読み込まれます。ただし、TiUPによってセットアップするはずだった多数のGrafanaダッシュボードがセットアップされなくなりますので注意してください。dashboard_dirを設定する場合は、まずdashboard_dirを設定しない状態でGrafanaコンポーネントを起動させ、Grafanaサーバーにある/tidb-deploy/grafana-3000/provisioning/dashboards内に配置される多数のダッシュボード定義ファイルを/home/tidb/grafana_dashboard/に保存しておくことをおすすめします。または、GitHubからダッシュボード定義ファイルを入手してください。

注17 <https://grafana.com/docs/grafana/v7.5/administration/configuration/>

注18 <https://grafana.com/grafana/dashboards/>

- PD用のダッシュボード定義ファイル^{注19}
- TiDB用のダッシュボード定義ファイル^{注20}
- TiKV用のダッシュボード定義ファイル^{注21}
- TiFlash用のダッシュボード定義ファイル^{注22}
- TiProxy用のダッシュボード定義ファイル^{注23}

その他カスタマイズ項目を確認するにはドキュメント^{注24}を参照してください。

Alertmanagerのカスタマイズ

Alertmanagerの設定をカスタマイズする場合は設定ファイルを作成し、TiUPのトポロジーファイルの `config_file` で指定します。設定ファイル内の項目はAlertmanagerのドキュメントの Configuration^{注25}を参照してください。次の例はAlertmanagerでSlack通知をするための設定です。

```
alertmanager_slack.yaml
global:
  resolve_timeout: 5m
  slack_api_url: '<slack_webhook_url>'

route:
  receiver: 'slack-notifications'
  group_by: [alertname, datacenter, app]

receivers:
- name: 'slack-notifications'
  slack_configs:
  - channel: '#alerts'
    text: 'Triggered: {{ .GroupLabels.alertname }} | App: {{ .GroupLabels.app }}'
    send_resolved: true
```

作成した `alertmanager_slack.yaml` を TiUP のトポロジーファイル内の `config_file` で指定します。配置場所は任意の場所で問題ありません。ただし、TiUP 実行ユーザーがアクセスできる必要があります。

```
Alertmanagerのカスタム設定
alertmanager_servers:
- host: 10.2.100.18
  config_file: /home/tidb/alertmanager_config/alertmanager_slack.yaml
```

注19 <https://github.com/tikv/pd/blob/master/metrics/grafana>

注20 <https://github.com/pingcap/tidb/tree/master/pkg/metrics/grafana>

注21 <https://github.com/tikv/tikv/tree/master/metrics/grafana>

注22 <https://github.com/pingcap/tiflash/tree/master/metrics/grafana>

注23 <https://github.com/pingcap/tiproxy/tree/main/pkg/metrics/grafana>

注24 https://docs.pingcap.com/tidb/stable/tiup-cluster-topology-reference#grafana_servers

注25 <https://prometheus.io/docs/alerting/0.27/configuration/>

TiUPでAlertmanagerコンポーネントが作成されると、alertmanager_serversに設定されているサーバー上の/tidb-deploy/alertmanager-9093/conf/alertmanager.ymlの内容が書き換わります。



TiDB Dashboardの活用



TiDB DashboardはTiDBを安定して運用するために欠かせないツールと言えるでしょう。本節ではTiDB Dashboardの詳細について説明します。TiDB DashboardはTiDB Self-Managed、TiDB CommunityおよびTiDB Playgroundで使用できます。TiDB CloudではWebコンソールが提供されていますので、そちらを参照してください。

TiDB Dashboardへのログインを制御する

TiDB Dashboardにアクセスすると、まず最初にログイン画面が表示されます。TiDB Dashboardでは、ログインに使用するユーザーとパスワードはデータベースのユーザーおよびパスワードを使用します。

TiDB Dashboardの表示に最低限必要な権限は次になります。

- PROCESS
- SHOW DATABASES
- CONFIG
- DASHBOARD_CLIENT

TiDB サーバーがSEM(Security Enhanced Mode)で稼働している場合は追加で次の権限が必要になります。

- RESTRICTED_TABLES_ADMIN
- RESTRICTED_STATUS_ADMIN
- RESTRICTED_VARIABLES_ADMIN

TiDB Dashboardではシステム変数を操作する場合があります。その場合は次の権限が必要になります。

- SYSTEM_VARIABLES_ADMIN

また、高速プランバインディング機能を使用する場合は下記の権限が必要になります。

- SYSTEM_VARIABLES_ADMIN
- SUPER

TiDB Dashboardの表示に最低限必要な権限を持つdashboardユーザーを作成する例を次に記述します。

TiDB Dashboard参照権限を持つユーザーを作成

```
mysql> CREATE USER 'dashboard'@'%' IDENTIFIED BY '<password>';
Query OK, 0 rows affected (0.07 sec)

mysql> GRANT PROCESS, CONFIG, SHOW DATABASES, DASHBOARD_CLIENT ON *.* TO 'dashboard'@'%';
Query OK, 0 rows affected (0.08 sec)
```

TiDBはRBAC(Role-Based Access Control)に対応しているため、TiDB Dashboard参照用のロールを作成してユーザーに割り当てることで制御が可能です。dashboard_viewerのロールを作成し、dashboardユーザーに割り当てる例を次に記述します。

RBACによる権限管理

```
mysql> CREATE USER 'dashboard'@'%' IDENTIFIED BY '<password>';
Query OK, 0 rows affected (0.07 sec)

mysql> CREATE ROLE 'dashboard_viewer';
Query OK, 0 rows affected (0.04 sec)

mysql> GRANT PROCESS, CONFIG, SHOW DATABASES, DASHBOARD_CLIENT ON *.* TO 'dashboard_viewer'@'%';

mysql> GRANT 'dashboard_viewer' TO 'dashboard'@'%';
Query OK, 0 rows affected (0.04 sec)

mysql> SET DEFAULT ROLE 'dashboard_viewer' to 'dashboard'@'%';
Query OK, 0 rows affected (0.06 sec)
```

SSOを使用したログイン

TiDB DashboardはSSO(Single Sign-On)を使用したログインも可能です。SSOを有効にするにはTiDB Dashboardにログインし、画面左下に表示されているユーザー名をクリック後に「Enable to use SSO when sign into TiDB Dashboard」を有効にします(図7.4)。

有効にするとSSOログインに必要な情報を入力する画面が表示されます。

本項ではGoogle認証を使用してTiDB Dashboardにログインする手順を説明します。まずOIDC Client IDとOIDC Client Secretを取得します。Google CloudのヘルプページのSetting up OAuth 2.0^{注26}の手順に従って取得してください。設定時にredirect URIsの項目がありますので、たとえばhttp://127.0.0.1:12333/dashboard/?sso_callback=1のように設定します。IPアドレスやポート番号は環境に合わせて変

注26 <https://support.google.com/cloud/answer/6158849?hl=en>

図7.4 SSOの有効化

TiDB Dashboard v8.5.0

Overview

Cluster Info

Top SQL

SQL Statements

Slow Queries

Key Visualizer

Cluster Diagnostics

Session

Share Current Session Revoke Authorization Codes Sign Out

Single Sign-On (SSO)

Enable to use SSO when sign into TiDB Dashboard

☐

OIDC based SSO is supported

Update Cancel

TiDB Dashboard Internal Version v8.5.0

TiDB Dashboard Build Git Hash 358548e82b7b0cf97d6b8c9b929f66397856f0db

TiDB Dashboard Build Time 2024-12-19 05:54:48

TiDB Dashboard Run in Standalone Mode

Yes

PD Version

root

更してください。

OIDC Client IDとOIDC Client Secretを取得したら画面で入力し、OIDC Discovery URLにはhttps://accounts.google.com/を入力します。

次に「Authorize Impersonation」をクリックしてSQL User to ImpersonateおよびSQL User Passwordを設定します(図7.5)。Impersonate SQL UserはSSOログイン時にTiDBに接続するユーザーを指定します。

図7.5 impersonate SQL User

Authorize Impersonation

SQL User to Impersonate

root

SQL User Password

The password of the SQL user will be stored encrypted. The impersonation will fail after SQL user changes the password.

Authorize and Save Cancel

図7.6 SSOの有効化設定の完了

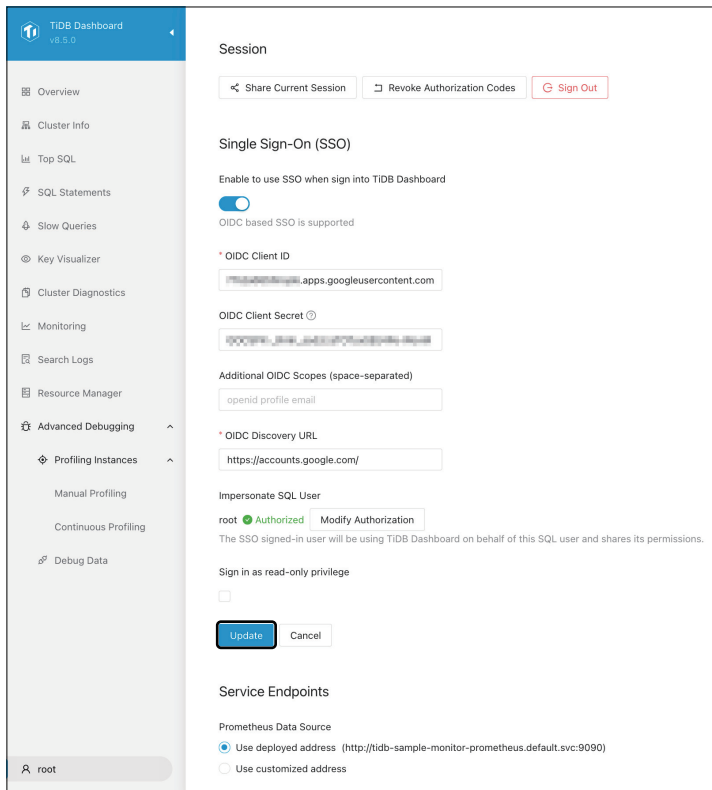


図7.7 SSOログイン画面

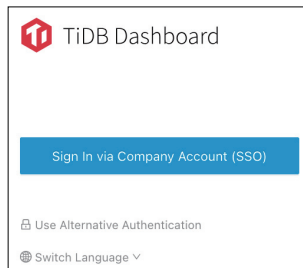
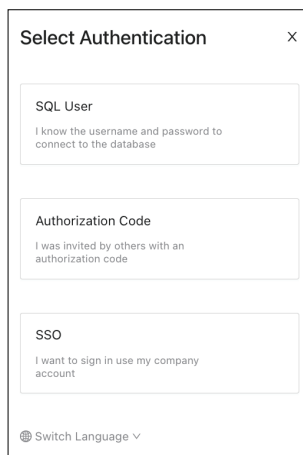


図7.8 ログイン方式の切り替え



設定が完了したら「Authorize and Save」をクリックします。設定した Impersonate SQL User と共に緑の文字で「Authorized」が表示されます(図7.6)。

すべての設定が完了したら「Update」をクリックして設定を保存します。

次のログイン以降、ログイン画面がSSOログイン画面に切り替わります(図7.7)。

なお、SSO ログインせずに従来の方式でログインする場合は「Use Alternative Authentication」をクリックすることでログイン方式を切り替えることが可能です(図7.8)。

TiDB Dashboardのページ詳細

TiDB Dashboardではさまざまな情報が表示されています。本項では日々の運用に役立つTiDB Dashboardの項目を説明していきます。どの項目を見たら良いのか分からない、といった場合はこちらを参考にしてください。また、各ページの使い分けの指標を記述しますのでこちらも参考にしてください。

- **Top SQL**

TiDBサーバーやTiKVサーバーのCPU使用率が高い時間帯がある場合や特定のSQLによる負荷が高い可能性がある場合

- **SQL Statements**

各SQLクエリの実行時間は短い、実行回数が多く累積の実行時間が長い場合

- **Slow Queries**

特定のSQLの処理時間が長い場合

- **Key Visualizer**

I/Oのボトルネックの調査、ホットスポットの疑いがある場合

TiDB Dashboardのページについての詳細を順に説明します。

Overview

TiDB Dashboardにログイン後、まず最初に表示されるTiDB全体の概要ページです。QPSやレイテンシー、クラスターのCPU使用率とメモリ使用量、読み取りと書き込みのI/O量を確認できます。「View More Metrics」をクリックすることで「Monitoring」に遷移して、より詳細な情報を確認できます。ページの右側にはクラスターのサーバー構成が表示されており、「Alive Instances」をクリックすることで「Cluster Info」に遷移します。また、モニタリングコンポーネントを有効にしていればGrafanaダッシュボードへ遷移できます。各種メトリクスはモニタリングコンポーネントが有効になっていない場合には表示されませんので注意してください。

Cluster Info

クラスターのサーバー構成を表示します。表示される項目はタブで分かれており、「Instances」はコンポーネントごとのサーバー一覧が表示されます。「Hosts」はクラスターの物理的な構成リソースを表示します。「Disks」はクラスターのディスク容量のキャパシティが表示されます。「Store Topology」はデータがどのように保存されるかを可視化します。「Statistics」ではコンポーネントごとのリソース量などが確認できます。

「Instances」タブのStatusやUp Timeの情報を確認して、各サーバーが正常に稼働しているかをStatusで確認してください。また、稼働時間(Up Time)を確認して意図しない再起動が発生していた場合は原因を調べてください。「Hosts」タブではCPU UsageやMemory Usageを確認して80%以上使用していた場合は増設を検討してください。「Disks」タブでも同様にDisk Usageを確認して容量不足にならないように増設を検討してください。

Top SQL

Top SQLはSQLごとのCPUオーバーヘッドをリアルタイムに監視できます。表示されたSQLを選択することで実行計画の確認もできますので、パフォーマンス問題の原因の特定に役立ちます。実行中

のものも含め、実行されたすべてのSQLが収集され、収集されたデータは最大30日間保存できます。TiDB サーバーおよびTiKVサーバーごとにオーバーヘッドを確認でき、特定のサーバーの高負荷の原因調査にも役立ちます。

このTop SQL機能はデフォルトで無効になっています。有効にするにはページの右上隅にある歯車ボタンをクリックして、「Enable Feature」を有効にします。またはクラスターのシステム変数tidb_enable_top_sqlをtrueにします。システム変数の変更は次のように実行します。

Top SQL機能を有効化する

```
mysql> SET GLOBAL tidb_enable_top_sql = true;  
Query OK, 0 rows affected (0.01 sec)
```

無効にする時はfalseを設定してください。またはページの右上隅にある歯車ボタンから無効化してください。

Top SQL機能を無効化する

```
mysql> SET GLOBAL tidb_enable_top_sql = false;  
Query OK, 0 rows affected (0.01 sec)
```

Top SQL機能はTiDBのパフォーマンスへの影響が若干ありますので、有効にする際には注意してください。ドキュメントのTiDB Dashboard Top SQL Page^{注27}によればパフォーマンスへの影響は通常3%未満とされています。

Top SQL機能はTiDBのNgMonitoringコンポーネントが稼働している場合に利用可能な機能です。TiUPやTiDB PlaygroundでTiDBを構築した場合はデフォルトでNgMonitoringが有効になっています。TiDB Operatorを使用して構築した場合は別途TidbNGMonitoringリソースを定義する必要があります。定義の仕方は第4章内の「NgMonitoringを有効化する」を参照してください。

SQL Statements

SQLの実行ステータスの統計情報が表示されます。SQLは正規化され、似たようなSQLは1つにまとめられます。このページは合計実行時間または平均実行時間が長いSQLを分析するためによく使用されます。SQLはさまざまな条件でフィルター可能で、データベース別に表示や、実行の種類(SELECTやSETなど)ごとに表示できます。リストに表示されたSQLをクリックすることで詳細ページに移動し、より詳細な情報が表示されます。

実行ステータスの統計データはINFORMATION_SCHEMA内のSTATEMENTS_SUMMARYテーブルおよびSTATEMENTS_SUMMARY_HISTORYテーブルに格納されています。デフォルトではSTATEMENTS_SUMMARYテーブルは30分ごとにクリアされます。STATEMENTS_SUMMARY_HISTORYテーブルはデフォルトで最大3,000行のSQLが12時間保存されます(30分間隔で24回保存されます)。各種設定を変更する場合は次の設定

注27 <https://docs.pingcap.com/tidb/stable/top-sql>

項目を参考にしてください。完全な設定項目のリストはシステム変数一覧^{注28}から確認できます。

- **tidb_enable_stmt_summary**
SQL実行ステータスの統計を有効化(デフォルトはtrue)
- **tidb_stmt_summary_refresh_interval**
統計データの更新間隔秒数(デフォルトは1800)
- **tidb_stmt_summary_history_size**
統計データの履歴保存サイズ(デフォルトは24)
- **tidb_stmt_summary_max_stmt_count**
統計データに保存する最大のSQL数(デフォルトは3000)

たとえば統計データの更新間隔を1時間(3600秒)にするには次のように実行します。

統計データの更新間隔を1時間に変更

```
mysql> SET GLOBAL tidb_stmt_summary_refresh_interval = 3600;  
Query OK, 0 rows affected (0.01 sec)
```

各種設定の変更はページの右上隅にある歯車ボタンからでも可能です。

デフォルトではSTATEMENTS_SUMMARY_HISTORYテーブルはTiDBサーバーのメモリ上に保存されます。そのためTiDBサーバーが再起動されるとデータは失われます。TiDBのバージョン6.6.0よりtidb_stmt_summary_enable_persistentを有効にすることで永続化できるようになりましたが、この機能は執筆時点ではまだ実験的な機能となっていますので、使用する際には注意してください。tidb_stmt_summary_enable_persistentは起動後に変更できない読み込み専用のシステム変数となっています(デフォルトではfalseです)。そのため、有効にするにはクラスター起動時に設定を読み込ませる必要があります。TiUPを使用する時はトポロジーファイル内に次の内容を記述してください。

topology.yaml

```
server_configs:  
  tidb:  
    instance.tidb_stmt_summary_enable_persistent: true
```

TiDB Playgroundでは次のように設定を読み込ませて起動します。

tidb_stmt_enable_persistent.toml

```
[instance]  
tidb_stmt_summary_enable_persistent = true
```

tidb_stmt_summary_enable_persistentを有効にして起動

```
$ tiup playground --db.config tidb_stmt_enable_persistent.toml
```

有効になっているかの確認はシステム変数を参照します。

注28 <https://docs.pingcap.com/ja/tidb/stable/system-variables>

tidb_stmt_summary_enable_persistentの確認

```
mysql> SHOW GLOBAL VARIABLES LIKE 'tidb_stmt_summary_enable_persistent';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| tidb_stmt_summary_enable_persistent | ON |
+-----+-----+
1 row in set (0.01 sec)
```

tidb_stmt_summary_enable_persistentを有効化する際は、必要に応じて次の設定項目も設定してください。

- **tidb_stmt_summary_filename**
保存するファイルの名前(デフォルトはtidb-statements.log)
- **tidb_stmt_summary_file_max_days**
保存する日数(デフォルトは3)
- **tidb_stmt_summary_file_max_size**
保存するファイルの最大MiBサイズ(デフォルトは64)
- **tidb_stmt_summary_file_max_backups**
保存するファイルのバックアップ数(デフォルトは0)

▶ Slow Queries

クラスター内のすべてのスロークエリを表示、検索できます。検索時にはデータベースの指定やキーワードを条件にできます。表示されたスロークエリを選択すると詳細な実行計画を確認できます。「SQL Statements」のページとは違い、SQLは正規化されずに表示されます。

スロークエリのリスト表示時はデフォルトでは「Query」「Finish Time」「Latency」「Max Memory」について表示されます。表示する項目は選択可能で、ページ右側の「Columns」で表示したい項目を選択できます。各項目についてはCLUSTER_SLOW_QUERYテーブルのスキーマ^{注29}を参照してください。また、各項目でソートして表示できますのでスロークエリ検索時に活用してください。

デフォルトでは実行時間が300ミリ秒を超えるSQLクエリはスロークエリとみなされ、スロークエリログに記録されます(デフォルトはtidb-slow.logです)。スロークエリログはSQLから参照でき、INFORMATION_SCHEMA内のSLOW_QUERYテーブルから参照できます(テーブル参照時にファイルからデータを読み込みます)。SLOW_QUERYテーブルは接続したTiDBサーバー内のスロークエリが格納されています。クラスター全体のスロークエリを確認したい場合はCLUSTER_SLOW_QUERYテーブルを参照してください。CLUSTER_SLOW_QUERYテーブルはSLOW_QUERYテーブルとほぼ同じスキーマですが、INSTANCEカラムが追加されています。

スロークエリの有効／無効および実行時間のしきい値は、TiDBサーバーの設定項目のtidb_enable_

注29 <https://docs.pingcap.com/tidb/stable/information-schema-slow-query>

slow_log(デフォルトはtrueです)とtidb_slow_log_threshold(単位はミリ秒でデフォルトは300です)で行います。

スロークエリ機能の無効化

```
mysql> SET GLOBAL tidb_enable_slow_log = false;
Query OK, 0 rows affected (0.01 sec)
```

スロークエリのしきい値を変更

```
mysql> SET GLOBAL tidb_slow_log_threshold = 1000;
Query OK, 0 rows affected (0.01 sec)
```

上記の手順での設定変更は接続しているTiDBサーバーのみ設定が反映されます。複数のTiDBサーバーがある環境では、必要に応じて他のTiDBサーバーの設定も合わせて変更してください。

TiUPにて設定する場合はトポロジーファイルに次の内容を記述してください。

スローログを無効化

```
server_configs:
  tidb:
    instance.tidb_enable_slow_log: false
```

TiDB Playgroundでは次のように設定します。

tidb_disable_slow_log.toml

```
[instance]
tidb_enable_slow_log = false
```

スローログを無効化して起動

```
$ tiup playground --db.config tidb_disable_slow_log.toml
```

▶ Key Visualizer

ストレージ層を担当しているTiKV内でのキーのアクセス状況をヒートマップで可視化します。明るくなっている部分はアクセスが多いことを意味します。ヒートマップの横軸は時間の経過を表し、縦軸はリージョンを表します。

明るくなっているリージョンが表示されている場合は、高トラフィックが狭いキー範囲のデータに集中しているホットスポットが発生していますので、そのリージョンを担当しているTiKVサーバーが高負荷になっている可能性があります。ホットスポットが発生している場合は、書き込みトラフィックが高頻度な場合のベストプラクティス^{注30}が公開されていますので、ドキュメントを参考にホットスポットの解消を試みてください。

注30 <https://docs.pingcap.com/tidb/stable/high-concurrency-best-practices>

▶ Cluster Diagnostic

クラスターに対して診断が可能で、診断結果のレポートを参照できます。診断機能はクラスターに存在する可能性のある問題を発見して、診断結果とクラスター関連の負荷監視情報をレポートにまとめます。診断時間は1分から最大60分まで指定できます。また、「Compare by Baseline」ボタンを有効にすることで、レポートを比較する機能もあります。「Baseline Range Start Time」に比較の元になる時間帯を入力し、「Range Start Time」との比較をレポートで表示できます。比較レポート内では、Baselineで指定したレポート内容がTime Range t1、比較対象のレポート内容がTime Range t2として出力されます。診断レポートの詳細な読み方はドキュメントのTiDB Dashboard Diagnostic Report^{注31}を参照してください。

▶ Monitoring

さまざまなメトリクスをグラフで確認できます。「Cluster Info」ページと同様に、各種メトリクスはモニタリングコンポーネントが有効になっていない場合には表示されませんので注意してください。各種グラフはカテゴリで分類されています。カテゴリの説明は次になります。

- **Database Time**

データベース内のワークロードごとのパフォーマンスが確認できます。パフォーマンスのボトルネックの特定に役立ちます

- **Application Connection**

クラスターに対する接続の状況が確認できます。現在の接続数、エラーになった接続数などが表示されます

- **SQL Count**

1秒間あたりのクエリ処理数や失敗したクエリ数が確認できます

- **Latency Break Down**

さまざまな処理のレイテンシーが確認できます。レイテンシーは平均と99%タイルで表示されます

- **Transaction**

1秒間あたりのトランザクション数とトランザクション処理時間が確認できます

- **Core Path Duration**

TiKVやPDのパフォーマンスが確認できます

- **Server**

コンポーネントごとのCPU使用率やメモリ使用率などが確認できます。表示されるコンポーネントは主要コンポーネントに加え、TiFlashとTiProxyが含まれます

各グラフの詳細はドキュメントのTiDB Dashboard Monitoring Page^{注32}を参照してください。

注31 <https://docs.pingcap.com/tidb/stable/dashboard-diagnostics-report>

注32 <https://docs.pingcap.com/tidb/stable/dashboard-monitoring>

▶ Search Logs

クラスター内で稼働しているサーバーのログを表示できます。INFOやWARNなどのログレベルでの絞り込みや時間帯、サーバー単位での絞り込みが可能です。もちろんキーワードによる絞り込みもできます。わざわざ各サーバーにログインしてログを参照することなく、このページからログを参照できますので、運用時にはとても役に立ちます。こちらで検索可能なログはサーバーのプロセスが出力するログになり、データベースのスローログとは異なりますので注意してください。スローログの参照は「Slow Queries」ページで行います。

「search history」をクリックすることで過去に行った検索の履歴を参照できます。不要になった検索履歴は削除できますので、右上隅の「Delete All」をクリックするか、削除する行を選択して「Delete selected」をクリックして履歴を削除してください。

▶ Resource Manager

リソースグループの設定状況、リクエストユニットの見積もり、リクエストユニットに関連するメトリクスが確認できます。リクエストユニットの見積もりは、ハードウェア性能に基づいた見積もりとクラスター内ワークロードに基づいた見積もりを表示できます。ハードウェア性能を元にした見積もりに関しては、さらに細かく次のワークロードモデルごとに見積もりが表示されます。

- **tpcc**
大量のデータ書き込みを伴うワークロードモデル
- **oltp_write_only**
sysbenchのoltp_write_onlyと同様のワークロードモデル
- **oltp_read_write**
sysbenchのoltp_read_writeと同様のワークロードモデル
- **oltp_read_onlysysbench**
sysbenchのoltp_read_onlyと同様のワークロードモデル

このページはクラスター内のリソースグループの最適化をしたい場合に役に立ちます。

▶ Advanced Debugging

パフォーマンスデータを取得するためのプロファイルを実行できます。プロファイルでは各種コンポーネントのサーバーごとの現在のパフォーマンスデータを収集します。「Manual Profiling」はオンデマンドでプロファイルを収集します。定期的にバックグラウンドで収集する場合は「Continuous Profiling」を選択してください。「Continuous Profiling」を有効化した場合は「Manual Profiling」が実行できなくなるので注意してください。ドキュメントのTiDB Dashboard Instance Profiling - Continuous Profiling^{注33}によれば「Continuous Profiling」を有効にした場合に、TiDB全体のパフォーマンスに与える影響は通常1%

注33 <https://docs.pingcap.com/tidb/stable/continuous-profiling#frequently-asked-questions>

未満であるとされています。

収集されたパフォーマンスデータはFlameGraphまたはDAGとして視覚化できます。パフォーマンス劣化時のボトルネックの調査に役立ちます。

定期的なプロファイル収集はTiDBのNgMonitoringコンポーネントが稼働している場合に利用可能な機能です。TiUPでTiDBを構築した場合やTiDB Playgroundでは、デフォルトでNgMonitoringが有効になっています。TiDB Operatorを使用して構築した場合は別途TidbNGMonitoringリソースを定義する必要があります。定義の仕方は第4章内の「NgMonitoringを有効化する」を参照してください。

7-12 TiUPによる データベースパフォーマンス測定

TiUPを使用してクラスターに対してベンチマークを実行し、パフォーマンスを計測できます。執筆時点では表7.6に記述したベンチマークに対応しています。

表7.6 TiUPによるベンチマーク方式一覧

ベンチマーク	概要
TPC-C	一般的なデータベースのトランザクション性能検証
TPC-H	OLAP向けの性能検証
CH	TPC-CとTPC-Hの混合検証
YCSB	オープンソースで公開されているNoSQLデータベースを対象としたベンチマーク
RawSQL	SQLが書かれたファイルを読み込み実行

各ベンチマークの実行時には接続先の設定として`--host`と`--port`オプションを指定可能です。複数の接続先を指定する場合は`--host`オプションにカンマ区切りで指定します。`--port`オプションもカンマ区切りの指定を受け付けます。たとえば`--host 10.0.0.1,10.0.0.2 --port 4000,4001`と指定した場合は、`10.0.0.1:4000`、`10.0.0.1:4001`、`10.0.0.2:4000`、`10.0.0.2:4001`の4つに接続します。また、ベンチマークのデータセットはtestデータベースが使用されますが、こちらは`--db`オプションで変更できます。ベンチマーク実行時に使用するスレッド数を変更する場合は`--threads`オプションを使用してください。デフォルトではスレッド数は1です。

TPC-C

TPC-C^{注34}のベンチマークの実行は次のように行います。prepareを実行したあとにrunを実行します。

注34 <https://www.tpc.org/tpcc/>

TPC-Cのベンチマーク準備

```

$ tiup bench tpcc prepare
creating table warehouse
creating table district
creating table customer
creating table history
creating table new_order
creating table orders
creating table order_line
creating table stock
creating table item
load to item
load to warehouse in warehouse 1
load to stock in warehouse 1
load to district in warehouse 1
load to warehouse in warehouse 2
load to stock in warehouse 2
load to district in warehouse 2
...
(省略)
...
begin to check warehouse 10 at condition 3.3.2.6
begin to check warehouse 10 at condition 3.3.2.7
begin to check warehouse 10 at condition 3.3.2.11
Finished

```

ベンチマーク用のデータセットの規模はwarehouseの数によって決まります。--warehouses オプションで変更できます。デフォルトは10です。ですので、--warehouses 1と指定すれば件数はデフォルトの10分の1に、--warehouses 100と指定すれば件数はデフォルトの10倍になります。デフォルトで作成される各テーブルの件数は表7.7を参照してください。order_lineテーブルの件数はランダム (ordersの件数×5~15件の間のランダム)になります。

表7.7 warehouseが10の場合のテーブル件数

テーブル名	件数
warehouse	10
customer	300000
district	100
history	300000
new_order	90000
order_line	3000873
orders	300000
stock	1000000

prepareが正常に終了したらrunでベンチマークを実行します。--timeオプションで実行時間を指定

するか、`--count` オプションで回数を指定すると良いでしょう。

TPC-Cのベンチマークを10分間実行

```
$ tiup bench tpcc run --time 10m
Starting component bench: ...
[Current] DELIVERY - Takes(s): 9.7, Count: 9, ...
[Current] NEW_ORDER - Takes(s): 9.8, Count: 160, ...
[Current] ORDER_STATUS - Takes(s): 8.7, Count: 13, ...
[Current] PAYMENT - Takes(s): 9.9, Count: 132, ...
[Current] STOCK_LEVEL - Takes(s): 8.3, Count: 16, ...
...
(省略)
...
Finished
[Summary] DELIVERY - Takes(s): 59.7, Count: 84, ...
[Summary] NEW_ORDER - Takes(s): 59.8, Count: 1151, ...
[Summary] ORDER_STATUS - Takes(s): 58.7, Count: 103, ...
[Summary] PAYMENT - Takes(s): 59.9, Count: 1024, ...
[Summary] STOCK_LEVEL - Takes(s): 58.3, Count: 93, ...
tpmC: 1154.0, tpmTotal: 2465.0, efficiency: 897.4%
```

ベンチマークが終了するとサマリーが表示されます。

ベンチマークに使用したデータセットは`cleanup`で削除できます。

ベンチマークデータの削除

```
$ tiup bench tpcc cleanup
Starting component bench: ...
DROP TABLE IF EXISTS item
DROP TABLE IF EXISTS customer
DROP TABLE IF EXISTS district
DROP TABLE IF EXISTS history
DROP TABLE IF EXISTS new_order
DROP TABLE IF EXISTS order_line
DROP TABLE IF EXISTS orders
DROP TABLE IF EXISTS stock
DROP TABLE IF EXISTS warehouse
Finished
```

TPC-H

TPC-H^{注35}のベンチマークの実行はTPC-Cの時と同じように、`prepare`を実行したあとに`run`を実行します。

注35 <https://www.tpc.org/tpch/>

TPC-Hのベンチマーク準備

```
$ tiup bench tpch prepare
creating nation
creating region
creating part
creating supplier
creating partsupp
creating customer
creating orders
creating lineitem
generating nation table
...
(省略)
...
generate part/partsupplier tables done
generating orders/lineitem tables
generate orders/lineitem tables done
Finished
```

ベンチマーク用のデータセットの規模はScale Factorの数で指定します。コマンド実行時に--sfオプションで指定でき、デフォルトのScale Factorは1です。1の場合のテーブルサイズは表7.8になります。たとえば--sf 2と指定すればデータ件数は2倍になります。

表7.8 Scale Factorが1の場合のテーブル件数

テーブル名	件数
customer	150000
lineitem	6000000
nation	25
orders	1500000
part	200000
partsupp	800000
region	5
supplier	10000

prepareが正常に終了したらrunでベンチマークを実行しますが、ベンチマークを実行する前にデータセットがTiFlash コンポーネントにコピーされているか確認します。デフォルトではデータセットはTiFlashへコピーされません。たとえばtestデータベースをベンチマークで使用する場合は、次のように設定してTiFlashへデータをコピーさせます。TPC-HのシナリオはTiFlash コンポーネントを使用しないと失敗する場合がありますので注意してください。

```
mysql> ALTER DATABASE test SET TIFLASH REPLICA 1;
Query OK, 0 rows affected (1.65 sec)
In total 8 tables: 8 succeed, 0 failed, 0 skipped
```

データコピーの進捗は次のように確認します。AVAILABLE が 1 であれば同期は完了しています。

```
mysql> SELECT TABLE_SCHEMA, TABLE_NAME, AVAILABLE, PROGRESS
-> FROM INFORMATION_SCHEMA.TIFLASH_REPLICA WHERE TABLE_SCHEMA = 'test';
```

TABLE_SCHEMA	TABLE_NAME	AVAILABLE	PROGRESS
test	supplier	1	1
test	lineitem	1	1
test	customer	1	1
test	partsupp	1	1
test	nation	1	1
test	orders	1	1
test	region	1	1
test	part	1	1

8 rows in set (0.01 sec)

TiFlash の準備が終わったらベンチマークを実行します。--time オプションで実行時間を指定するか、--count オプションで回数を指定すると良いでしょう。

```
TPC-Hのベンチマークを10分間実行
$ tiup bench tpch run --time 10m
Starting component bench: ...
[Current] Q1: 3.29s
[Current] Q2: 6.34s
[Current] Q3: 4.87s
[Current] Q4: 1.58s
[Current] Q5: 24.53s
...
(省略)
...
Finished
[Summary] 1: 3.10s
[Summary] 2: 4.12s
...
(省略)
...
```

TPC-Hはベンチマークに使用されるSQLの定義がQ1からQ22まであり、仕様が公開^{注36}されています。ベンチマークに使用するSQLは選択することができ、--queriesで指定できます。たとえばQ1からQ5までを使用してベンチマークする時は次のように指定します。デフォルトではQ22まですべての定義が使用されます。

注36 https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-H_v3.0.1.pdf

TPC-HのQ1からQ5のみ使用

```
$ tiup bench tpch run --queries q1,q2,q3,q4,q5
```

ベンチマークが終了するとサマリーが表示されます。

ベンチマークに使用したデータセットはcleanupで削除できます。

ベンチマークデータの削除

```
$ tiup bench tpch cleanup
Starting component bench: ...
DROP TABLE IF EXISTS lineitem
DROP TABLE IF EXISTS partsupp
DROP TABLE IF EXISTS supplier
DROP TABLE IF EXISTS part
DROP TABLE IF EXISTS orders
DROP TABLE IF EXISTS customer
DROP TABLE IF EXISTS region
DROP TABLE IF EXISTS nation
Finished
```

CH

TPC-CとTPC-Hの両方のシナリオを使用した混合ベンチマークを実行します。そのため、準備段階で用意するデータセットはTPC-CとTPC-H両方用意する必要があります。データセットの作成の仕方は「TPC-C」や「TPC-H」の項を参照してください。

TPC-CとTPC-Hのベンチマーク準備

```
$ tiup bench tpcc prepare
...
(省略)
...
Finished

$ tiup bench tpch prepare
...
(省略)
...
Finished
```

両データセットが準備できたらrunで実行します。TPC-Hシナリオに使用するクライアントの数は--acThreadsオプションで指定します。デフォルトは1です。TPC-Cシナリオに使用するクライアントの数は--threadsオプションで指定します。こちらもデフォルトは1です。

クライアント数を変更して混合シナリオのベンチマーク実行

```
$ tiup bench ch run --acThreads 2 -threads 5
Finished: 5 OLTP workers, 2 OLAP workers
```

```
[Summary] DELIVERY - Takes(s): 148.9, Count: 264, ...
[Summary] NEW_ORDER - Takes(s): 149.0, Count: 2908, ...
[Summary] ORDER_STATUS - Takes(s): 147.5, Count: 249, ...
[Summary] PAYMENT - Takes(s): 149.1, Count: 2851, ...
[Summary] STOCK_LEVEL - Takes(s): 146.5, Count: 257, ...
tpmC: 1170.9, tpmTotal: 2631.3, efficiency: 9104.9%
[Summary] Q1      - Count: 1, Sum(ms): 31.5, Avg(ms): 31.5
[Summary] Q10     - Count: 1, Sum(ms): 341.0, Avg(ms): 340.9
[Summary] Q11     - Count: 1, Sum(ms): 80.9, Avg(ms): 80.9
[Summary] Q12     - Count: 1, Sum(ms): 153.4, Avg(ms): 153.4
[Summary] Q13     - Count: 1, Sum(ms): 38.8, Avg(ms): 38.8
[Summary] Q14     - Count: 1, Sum(ms): 149.8, Avg(ms): 149.7
...
(省略)
...
QphH: 24.5
```

出力結果はTPC-CとTPC-H両方表示されます。

ベンチマークに使用したデータセットを削除する際には、TPC-CとTPC-Hの両方でcleanupで削除してください。

ベンチマークデータの削除

```
$ tiup bench tpcc cleanup
$ tiup bench tpch cleanup
```

YCSB

YCSB^{注37}のシナリオでは、TiDB クラスターおよびストレージクラスターに対してベンチマークを実行できます。

▶ TiDB クラスターに対するベンチマーク

データセットの準備はloadで行います。TiDB サーバー(tidb.instances)およびデータ件数(recordcount)を--prop オプションで指定して準備します。複数のTiDB サーバーを使用する場合はカンマ区切りで指定してください。

10000 件のデータセットを準備

```
$ tiup bench ycsb load tidb \
  --prop tidb.instances="10.0.0.11:4000,10.0.0.12:4000" --prop recordcount=10000
Starting component bench: ...
***** properties *****
"tidb.instances"="10.0.0.11:4000,10.0.0.12:4000"
```

注37 <https://ycsb.site/>

```
"recordcount"="10000"
"dotransactions"="false"
"command"="load"
*****
INSERT - Takes(s): 10.0, Count: 2336, OPS: 233.7, Avg(us): 4228, ...
INSERT - Takes(s): 20.0, Count: 4608, OPS: 230.5, Avg(us): 4289, ...
INSERT - Takes(s): 30.0, Count: 6714, OPS: 223.8, Avg(us): 4419, ...
INSERT - Takes(s): 40.0, Count: 8817, OPS: 220.5, Avg(us): 4487, ...
INSERT - Takes(s): 50.0, Count: 10000, OPS: 200.0, Avg(us): 4485, ...
INSERT - Takes(s): 50.5, Count: 10000, OPS: 197.9, Avg(us): 4485, ...
Run finished, takes 50.539956986s
```

投入されたデータを確認します。

```
データセット件数の確認
mysql> select COUNT(*) from test.usertable;
+-----+
| COUNT(*) |
+-----+
|    10000 |
+-----+
1 row in set (0.01 sec)
```

デフォルトではテーブル名はusertableが使用されます。変更したい場合は--tableオプションで指定してください。

ベンチマークの実行はrunで行います。--prop オプションで実行回数(operationcount)を指定します。

```
10000回実行
$ tiup bench ycsb run tidb \
  --prop tidb.instances="10.0.0.11:4000,10.0.0.12:4000" --prop operationcount=10000
Starting component bench: ...
***** properties *****
"dotransactions"="true"
"command"="run"
"tidb.instances"="10.0.0.11:4000,10.0.0.12:4000"
"operationcount"="10000"
*****
READ   - Takes(s): 10.0, Count: 9399, OPS: 940.0, Avg(us): 969, ...
UPDATE - Takes(s): 10.0, Count: 484, OPS: 48.5, Avg(us): 1644, ...
READ   - Takes(s): 10.1, Count: 9511, OPS: 939.0, Avg(us): 970, ...
UPDATE - Takes(s): 10.1, Count: 489, OPS: 48.3, Avg(us): 1647, ...
Run finished, takes 10.131539684s
```

データセットの削除は手動で行う必要があります。

データセットの削除

```
mysql> DROP TABLE test.usertable;  
Query OK, 0 rows affected (0.33 sec)
```

▶ ストレージクラスターに対するベンチマーク

データセットの準備はloadで行います。TiKVサーバーが使用するPDサーバー(tikv.pd)およびデータ件数(recordcount)を--prop オプションで指定して準備します。複数のPDサーバーを使用する場合はカンマ区切りで指定してください。

10000件のデータセットを準備

```
$ tiup bench ycsb load tikv \  
  --prop tikv.pd="10.0.0.4:2379" --prop recordcount=10000  
Starting component bench: ...  
...  
(省略)  
...  
***** properties *****  
"command"="load"  
"tikv.pd"="10.0.0.4:2379"  
"recordcount"="10000"  
"dotransactions"="false"  
*****  
INSERT - Takes(s): 9.9, Count: 4470, OPS: 450.2, Avg(us): 2200, ...  
INSERT - Takes(s): 19.9, Count: 8925, OPS: 447.8, Avg(us): 2204, ...  
INSERT - Takes(s): 22.4, Count: 10000, OPS: 446.9, Avg(us): 2208, ...  
Run finished, takes 22.449186295s
```

ベンチマークの実行はrunで行います。--prop オプションで実行回数(operationcount)を指定します。

10000回実行

```
$ tiup bench ycsb run tikv \  
  --prop tikv.pd="10.0.0.4:2379" --prop operationcount=10000  
Starting component bench: ...  
...  
(省略)  
...  
***** properties *****  
"tikv.pd"="10.0.0.4:2379"  
"operationcount"="10000"  
"dotransactions"="true"  
"command"="run"  
*****  
READ   - Takes(s): 6.6, Count: 9514, OPS: 1434.3, Avg(us): 538, ...  
UPDATE - Takes(s): 6.6, Count: 486, OPS: 73.5, Avg(us): 3114, ...  
Run finished, takes 6.714611058s
```

RawSQL

ファイルに任意のSQLクエリを記述してテストに使用できます。

```
demo.sql
-- Prepare data
CREATE TABLE IF NOT EXISTS t (a int);
INSERT INTO t VALUES (1), (2), (3);

-- Execute SQL
SELECT a, sleep(rand()) FROM t WHERE a < 4*rand();
```

ファイルを`--query-files`オプションで指定して実行します。

```
demo.sqlのクエリを実行
$ tiup bench rawsql run --query-files demo.sql
[Current] DEMO: 4.09s
[Current] DEMO: 7.12s
[Current] DEMO: 6.07s
[Current] DEMO: 9.13s
[Current] DEMO: 12.91s
[Current] DEMO: 10.64s
[Current] DEMO: 9.37s
[Current] DEMO: 11.29s
Finished
[Summary] DEMO: 9.27s
```

先ほど使用したdemo.sql用のデータの削除は手動で行う必要があります。

```
データセットの削除
mysql> DROP TABLE test.t;
Query OK, 0 rows affected (0.33 sec)
```

より詳細な使い方

TiUPによるパフォーマンス計測は内部でgo-tpc^{注38}およびgo-ycsb^{注39}が使用されます。より詳細な使用方法は各ツールのGitHubのREADMEを参照してください。

注38 <https://github.com/pingcap/go-tpc>

注39 <https://github.com/pingcap/go-ycsb>

7-13

TiDB AI Assistantによる
ナレッジの検索

TiDBに関するドキュメントは大量に公開されていますが、必要な情報を見つけるのが難しい場合があります。その場合はTiDB AI Assistantを使用することで効率よく検索できる可能性があります。TiDB AI Assistantは自然言語で質問を投げることで関連するナレッジを表示するサービスで、日本語による検索も可能です。TiDB AI Assistantは<https://tidb.ai/>にアクセスすることで使用できます(図7.9)。

たとえば「BRを使用した場合にクラスター側に与えるパフォーマンス影響はどれくらい?」という質問を投げると、関連するナレッジが表示されます(図7.10)。

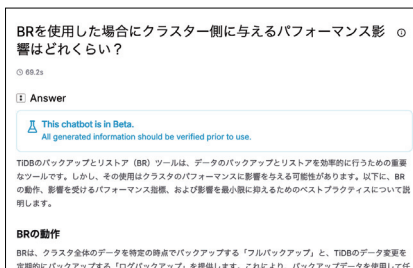
TiDB AI Assistantは執筆時点ではベータ版であり、表示されたナレッジ情報は必ずしも正しいとは限りませんので注意してください。

TiDB AI Assistantの仕組みはオープンソースとして公開されています。詳細はGitHubのリポジトリ^{注40}を参照してください。

図7.9 TiDB AI Assistantのトップ画面



図7.10 TiDB AI Assistantの質問例



7-14

まとめ

本章ではTiDBを徹底的に活用するために役に立つさまざまなテクニックを紹介しました。まだ紹介しきれていないテクニックもたくさんありますので、リリースノートやドキュメントを参照して、最新の情報を取得してください。TiDBの開発は非常に活発で、新しい機能が次々とリリースされています。ぜひ実際に動かしてみて、ご自身で体験してください。

注40 <https://github.com/pingcap/autoflow>

付 録

逆引き コマンドリファレンス

本章ではTiUPとTiDB Cloud CLIの使い方を目的ごとにまとめて紹介します。逆引きコマンドリファレンスとして活用してください。よく使うと思われるコマンドについては具体的な実行例も記述しています。

TiUP

TiUPをインストールする

```
$ curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/install.sh | sh
```

TiUPをアップデートする

```
$ tiup update --self
```

TiUPのバージョンを確認する

```
$ tiup --version
```

TiUPのヘルプを表示する

```
$ tiup --help
```

TiUPのサブコマンドのヘルプを表示する

```
$ tiup help <サブコマンド>  
$ tiup <サブコマンド> --help
```

実行例

```
$ tiup cluster help  
$ tiup cluster --help
```

TiUPをアンインストールする

```
$ tiup uninstall --self
```

コンポーネントの操作

TiUPのコンポーネント一覧を表示する

```
$ tiup list
```

インストール済みのTiUPのコンポーネント一覧を表示する

```
$ tiup list --installed
```


TiUPのコンポーネント一覧をインストールしている場合はバージョンと共に表示する

```
$ tiup list --verbose
```

TiUPのコンポーネントの利用可能なバージョンを表示する

```
$ tiup list <コンポーネント>
```

実行例

```
$ tiup list cluster
```

インストール済みのTiUPのコンポーネントのバージョンを表示する

```
$ tiup list <コンポーネント> --installed
```

TiUPのコンポーネントをインストールする

```
$ tiup install <コンポーネント>
```

実行例

```
$ tiup install dmctl
```

TiUPのコンポーネントをバージョン指定してインストールする

```
$ tiup install <コンポーネント>:<バージョン>
```

実行例

```
$ tiup install dmctl:v8.5.0
```

TiUPのコンポーネントを最新のリリース版にアップデートする

```
$ tiup update <コンポーネント>
```

すべてのTiUPのコンポーネントを最新のリリース版にアップデートする

```
$ tiup update --all
```

TiUPのコンポーネントを最新の開発版にアップデートする

```
$ tiup update <コンポーネント> --nightly
```

すべてのTiUPのコンポーネントを最新の開発版にアップデートする

```
$ tiup update --all --nightly
```

使用するTiUPのコンポーネントのバージョンを切り替える

```
$ tiup link <コンポーネント>:<バージョン>
```

TiUPのコンポーネントをアンインストールする

```
$ tiup uninstall <コンポーネント>:<バージョン>
```

TiUPのコンポーネントのすべてのバージョンをアンインストールする

```
$ tiup uninstall <コンポーネント> --all
```

TiUPでインストールしたすべてのコンポーネントをアンインストールする

```
$ tiup uninstall --all
```

シェル補完の操作

シェル補完を有効にするとTabキーを押すことでコマンドやオプションを補完できます。

TiUPのシェル補完を有効にする (Bash)

```
$ source <(tiup completion bash)
```

TiUPのシェル補完を有効にする (Z shell)

```
% source <(tiup completion zsh); compdef _tiup tiup
```

TiUPのシェル補完を有効にする (fish)

```
> tiup completion fish | source
```

TiUPのシェル補完を有効にする (PowerShell)

```
> tiup completion powershell | Out-String | Invoke-Expression
```

操作履歴の操作

TiUPの操作履歴を表示する (100件)

```
$ tiup history
```

TiUPの操作履歴を表示する (N件)

```
$ tiup history <N>
```

TiUPの操作履歴をすべて表示する

```
$ tiup history --all
```

TiUPの操作履歴をJSON形式で表示する

```
$ tiup history --format json
```

TiUPの60日間より前の操作履歴を削除する

```
$ tiup history cleanup
```

TiUPのN日間より前の操作履歴を削除する

```
$ tiup history cleanup --retain-days <N>
```

TiUPのすべての操作履歴を削除する

```
$ tiup history cleanup --all
```

```
$ tiup history cleanup --retain-days 0
```

TiDBの操作

TiDBやTiDB Playgroundの状態を確認する

```
$ tiup status
```

トポロジーファイルのテンプレートを表示する

```
$ tiup cluster template
```

```
$ tiup cluster template --full
```

```
$ tiup cluster template --multi-dc
```

```
$ tiup cluster template --local
```

任意のホストを指定してトポロジーファイルのテンプレートを表示する

```
$ tiup cluster template --local --tidb-servers ... --pd-servers ... --tikv-servers ...
```

実行例

```
$ tiup cluster template --local \  
  --tidb-servers 10.0.0.1,10.0.0.2 \  
  --pd-servers 10.0.0.3 \  
  --tikv-servers 10.0.0.4,10.0.0.5,10.0.0.6
```

TiDBを構築する前に確認する

```
$ tiup cluster check <トポロジーファイル>
```

実行例

```
$ tiup cluster check topology.yaml
```

TiDBを構築する前に確認項目の自動修正を試みる

```
$ tiup cluster check <トポロジーファイル> --apply
```

TiDBを構築する

```
$ tiup cluster deploy <クラスター名> <バージョン> <トポロジーファイル>
```

実行例

```
$ tiup cluster deploy sample-cluster v8.5.0 topology.yaml
```

TiDBの一覧を表示する

```
$ tiup cluster list
```

▶ 起動と停止

TiDB全体を起動する

```
$ tiup cluster start <クラスター名>
```

データベースの初期パスワードを生成してTiDBを起動する

```
$ tiup cluster start <クラスター名> --init
```

TiDBの特定のノードを起動する

```
$ tiup cluster start <クラスター名> --node <IPアドレス>:<ポート番号>
```

実行例

```
$ tiup cluster start sample-cluster --node 10.0.0.1:4000
```

TiDBの特定のロールを起動する

```
$ tiup cluster start <クラスター名> --role <ロール名>
```

実行例

```
$ tiup cluster start sample-cluster --role tidb  
$ tiup cluster start sample-cluster --role prometheus,grafana
```

TiDB全体を停止する

```
$ tiup cluster stop <クラスター名>
```

TiDBの特定のノードを停止する

```
$ tiup cluster stop <クラスター名> --node <IPアドレス>:<ポート番号>
```

TiDBの特定のロールを停止する

```
$ tiup cluster stop <クラスター名> --role <ロール名>
```

TiDB全体を再起動する

```
$ tiup cluster restart <クラスター名>
```

TiDB の特定のノードを再起動する

```
$ tiup cluster restart <クラスター名> --node <IPアドレス>:<ポート番号>
```

TiDB の特定のロールを再起動する

```
$ tiup cluster restart <クラスター名> --role <ロール名>
```

OS 起動時に TiDB のプロセスの自動起動を有効にする

```
$ tiup cluster enable <クラスター名>
```

OS 起動時に TiDB のプロセスの自動起動を無効にする

```
$ tiup cluster disable <クラスター名>
```

▶ スケールアウトとスケールイン

TiDB をスケールアウトする前に確認する

```
$ tiup cluster check <クラスター名> <スケールアウト用のポリシーファイル> --cluster
```

実行例

```
$ tiup cluster check sample-cluster scale_out.yaml --cluster
```

TiDB をスケールアウトする前に確認項目の自動修正を試みる

```
$ tiup cluster check <クラスター名> <スケールアウト用のポリシーファイル> --cluster --apply
```

TiDB をスケールアウトする

```
$ tiup cluster scale-out <クラスター名> <スケールアウト用のポリシーファイル>
```

TiDB をスケールインする

```
$ tiup cluster scale-in <クラスター名> --node <IPアドレス>:<ポート番号>
```

実行例

```
$ tiup cluster scale-in sample-cluster --node 10.0.0.1:4000
```

▶ 削除

TiDB を削除する

```
$ tiup cluster destroy <クラスター名>
```

データを保持して TiDB を削除する

```
$ tiup cluster destroy <クラスター名> --retain-node-data <IPアドレス>:<ポート番号>
```

```
$ tiup cluster destroy <クラスター名> --retain-role-data <ロール名>
```

▶ バージョンアップ

TiDB をバージョンアップする

```
$ tiup cluster upgrade <クラスター名> <バージョン>
```

実行例

```
$ tiup cluster upgrade sample-cluster v8.5.1
```

オフラインでTiDB をバージョンアップする

```
$ tiup cluster stop <クラスター名>  
$ tiup cluster upgrade <クラスター名> <バージョン> --offline
```

▶ 情報の表示

TiDB の情報を表示する

```
$ tiup cluster display <クラスター名>
```

TiDB の情報をバージョンと共に表示する

```
$ tiup cluster display <クラスター名> --versions
```

TiDB のバージョンのみ表示する

```
$ tiup cluster display <クラスター名> --version
```

TiDB のダッシュボードの URL のみ表示する

```
$ tiup cluster display <クラスター名> --dashboard
```

TiDB の現在のトポロジー情報を表示する

```
$ tiup cluster show-config <クラスター名>
```

▶ 監査ログの操作

TiDB の監査ログを表示する

```
$ tiup cluster audit
```

TiDB の監査ログの詳細を表示する

```
$ tiup cluster audit <ID>
```

TiDB の監査ログの内容を再実行する

```
$ tiup cluster replay <ID>
```

TiDBの60日間より前の監査ログを削除する

```
$ tiup cluster audit cleanup
```

TiDBのN日間より前の監査ログを削除する

```
$ tiup cluster audit cleanup --retain-days <N>
```

メタデータのバックアップと復元

TiDBのメタデータをバックアップする

```
$ tiup cluster meta backup <クラスター名>
```

TiDBのメタデータをファイル名を指定してバックアップする

```
$ tiup cluster meta backup <クラスター名> --file <ファイル名>
```

TiDBのメタデータを復元する

```
$ tiup cluster meta restore <クラスター名> <ファイル名>
```

TiDB Playgroundの操作

TiDB Playgroundを起動する

```
$ tiup playground
```

バージョン指定してTiDB Playgroundを起動する

```
$ tiup playground <バージョン>
```

実行例

```
$ tiup playground v8.5.0
```

タグを指定してTiDB Playgroundを起動する

```
$ tiup playground --tag <tag>
```

モニタリング環境を無効化してTiDB Playgroundを起動する

```
$ tiup playground --without-monitor
```

コンポーネントの台数を指定してTiDB Playgroundを起動する

```
$ tiup playground --db <N> --pd <N> --kv <N> --tiflash <N>
```

実行例

```
$ tiup playground --pd 3  
$ tiup playground --db 2 --pd 1 --kv 3 --tiflash 1
```

TiProxy を有効化して TiDB Playground を起動する

```
$ tiup playground --tiproxy <N>
```

DM を有効化して TiDB Playground を起動する

```
$ tiup playground --dm-master <N> --dm-worker <N>
```

起動しているプロセスを確認する

```
$ tiup playground display
```

コンポーネントをスケールアウトする

```
$ tiup playground scale-out --db <N> --pd <N> --kv <N> --tiflash <N>
```

実行例

```
$ tiup playground scale-out --db 1
```

コンポーネントをスケールインする

```
$ tiup playground scale-in --pid <プロセスID>
```

TiDB Playground へ接続する

```
$ tiup client
```

TiDB Playground のデータを削除する

```
$ tiup clean <tag>
```

すべての TiDB Playground のデータを削除する

```
$ tiup clean --all
```

デモ用データの操作

デモ用データを投入する

```
$ tiup demo bookshop prepare
```

デモ用データを削除する

```
$ tiup demo bookshop cleanup
```

ベンチマークの実行

TPC-C用のデータを投入する

```
$ tiup bench tpcc prepare
```


TPC-Cを実行する

```
$ tiup bench tpcc run
```

TPC-C用のデータを削除する

```
$ tiup bench tpcc cleanup
```

TPC-H用のデータを投入する

```
$ tiup bench tpch prepare
```

TPC-Hを実行する

```
$ tiup bench tpch run
```

TPC-H用のデータを削除する

```
$ tiup bench tpch cleanup
```

CH(TPC-CとTPC-Hの混合)用のデータを投入する

```
$ tiup bench tpcc prepare  
$ tiup bench tpch prepare
```

CH(TPC-CとTPC-Hの混合)を実行する

```
$ tiup bench ch run
```

CH(TPC-CとTPC-Hの混合)用のデータを削除する

```
$ tiup bench tpcc cleanup  
$ tiup bench tpch cleanup
```

YCSBのTiDBクラスター用のデータを投入する

```
$ tiup bench ycsb load tidb \  
--prop tidb.instances="<IPアドレス>:<ポート番号>" --prop recordcount=<N>
```

TiDBクラスターに対してYCSBを実行する

```
$ tiup bench ycsb run tidb \  
--prop tidb.instances="<IPアドレス>:<ポート番号>" --prop operationcount=<N>
```

YCSBのTiKV用のデータを投入する

```
$ tiup bench ycsb load tikv \  
--prop tikv.pd="<IPアドレス>:<ポート番号>" --prop recordcount=<N>
```

TiKVに対してYCSBを実行する

```
$ tiup bench ycsb run tikv \  
--prop tikv.pd="<IPアドレス>:<ポート番号>" --prop operationcount=<N>
```

クライアントツールの実行

pd-ctl

```
$ tiup ctl:<バージョン> pd ...
```

tidb-ctl

```
$ tiup ctl:<バージョン> tidb ...
```

tikv-ctl

```
$ tiup ctl:<バージョン> tikv ...
```

tidb-lightning-ctl

```
$ tiup ctl:<バージョン> tidb-lightning ...
```

cdc

```
$ tiup ctl:<バージョン> cdc ...
```

dmctl

```
$ tiup dmctl ...
```

etcdctl

```
$ tiup ctl:<バージョン> etcd ...
```

ticloud

```
$ tiup cloud ...
```

TiDB Cloud CLI

TiDB Cloud CLIは、ほとんどの場合でオプションを指定しない実行方式とオプションを指定する実行方式のどちらもサポートしています。オプションを指定しない実行方式では、インタラクティブモードで必要な情報を入力することでコマンドを実行できます。

TiDB Cloud CLIをインストールする

```
$ curl https://raw.githubusercontent.com/tidbcloud/tidbcloud-cli/main/install.sh | sh
```

```
$ tiup install cloud
```

TiDB Cloud CLIをアップデートする

```
$ ticloud upgrade
```

TiDB Cloud CLIのヘルプを表示する

```
$ ticloud --help
```

TiDB Cloud CLIのサブコマンドのヘルプを表示する

```
$ ticloud help <サブコマンド>  
$ ticloud <サブコマンド> --help
```

実行例

```
$ ticloud help serverless  
$ ticloud serverless --help
```

TiDB Cloud CLIをアンインストールする

```
$ tiup uninstall cloud
```

シェル補完の操作

シェル補完を有効にするとTabキーを押すことでコマンドやオプションを補完できます。

TiDB Cloud CLIのシェル補完を有効にする (Bash)

```
$ source <(ticloud completion bash)
```

TiDB Cloud CLIのシェル補完を有効にする (Z shell)

```
% source <(ticloud completion zsh); compdef _tiup tiup
```

TiDB Cloud CLIのシェル補完を有効にする (fish)

```
> ticloud completion fish | source
```

TiDB Cloud CLIのシェル補完を有効にする (PowerShell)

```
> ticloud completion powershell | Out-String | Invoke-Expression
```

プロファイル操作

プロファイルを作成する

```
$ ticloud config create
```

```
$ ticloud config create --profile-name <プロファイル名> \  
--public-key <public-key> \  
--private-key <private-key>
```

プロファイルの一覧を表示する

```
$ ticloud config list
```

プロファイルを切り替える

```
$ ticloud config use <profile-name>
```

プロファイルを削除する

```
$ ticloud config delete <profile-name>
```

OAuth認証

OAuth 認証する

```
$ ticloud auth login
```

OAuth 認証を解除する

```
$ ticloud auth logout
```

TiDB Cloud Serverlessの操作

有効なリージョンを表示する

```
$ ticloud serverless region
```

TiDB Cloud Serverlessを作成する

```
$ ticloud serverless create
```

```
$ ticloud serverless create \  
--display-name <表示名> --region <リージョン>
```

```
$ ticloud serverless create \  
--display-name <表示名> --region <リージョン> \  
--spending-limit-monthly <金額(USD cents)>
```

実行例

```
$ ticloud serverless create \  
--display-name sample-serverless --region aws-ap-northeast-1 \  
--spending-limit-monthly 10000
```

TiDB Cloud Serverlessの一覧を表示する

```
$ ticloud serverless list
```

TiDB Cloud Serverlessの詳細を表示する

```
$ ticloud serverless describe
```

```
$ ticloud serverless describe --cluster-id <クラスターID>
```

TiDB Cloud Serverlessの表示名を変更する

```
$ ticloud serverless update --cluster-id <クラスターID> --display-name <新しい表示名>
```

TiDB Cloud Serverlessに接続する

```
$ ticloud serverless shell
```

```
$ ticloud serverless shell --cluster-id <クラスターID> \  
--user <ユーザー名> --password <パスワード>
```

TiDB Cloud Serverlessを削除する

```
$ ticloud serverless delete
```

```
$ ticloud serverless delete --cluster-id <クラスターID>
```

▶ ブランチの操作

ブランチを作成する

```
$ ticloud serverless branch create
```

```
$ ticloud serverless branch create --cluster-id <クラスターID> \  
--display-name <ブランチ名> --parent-id <派生元のブランチID> \  
--parent-timestamp <タイムスタンプ (RFC3339形式)>
```

実行例 現在のクラスターデータからブランチを作成

```
$ ticloud serverless branch create --cluster-id 01234567890123456789 \  
--display-name newbranch
```

実行例 特定のブランチからブランチを作成

```
$ ticloud serverless branch create --cluster-id 01234567890123456789 \  
--display-name newbranch --parent-id bran-abcdefghijklmnpqrstuvwxyz
```

実行例 特定の日時のデータからブランチを作成

```
$ ticloud serverless branch create --cluster-id 01234567890123456789 \  
--display-name newbranch --parent-timestamp 2024-12-14T00:00:00Z
```

ブランチの一覧を表示する

```
$ ticloud serverless branch list
```

```
$ ticloud serverless branch list --cluster-id <クラスターID>
```

ブランチに接続する

```
$ ticloud serverless branch shell
```

```
$ ticloud serverless branch shell --cluster-id <クラスターID> \  
--branch-id <ブランチID> --user <ユーザー名> --password <パスワード>
```

ブランチを削除する

```
$ ticloud serverless branch delete
```

```
$ ticloud serverless branch delete --cluster-id <クラスターID> \  
--branch-id <ブランチID>
```

▶ SQLユーザーの操作

SQLユーザーを作成する

```
$ ticloud serverless sql-user create
```

```
$ ticloud serverless sql-user create --cluster-id <クラスターID> \  
--user <ユーザー名> --password <パスワード> --role <ロール>
```

実行例

```
$ ticloud serverless sql-user create --cluster-id 01234567890123456789 \  
--user newuser --password newpassword --role role_readwrite
```

SQLユーザー一覧を表示する

```
$ ticloud serverless sql-user list
```

```
$ ticloud serverless sql-user list --cluster-id <クラスターID>
```

SQLユーザーのパスワードを変更する

```
$ ticloud serverless sql-user update
```

```
$ ticloud serverless sql-user update --cluster-id <クラスターID> \  
--user <ユーザー名> --password <新しいパスワード>
```

SQLユーザーのロールを変更する

```
$ ticloud serverless sql-user update
```

```
$ ticloud serverless sql-user update --cluster-id <クラスターID> \  
--user <ユーザー名> --role <新しいロール>
```

SQLユーザーを削除する

```
$ ticloud serverless sql-user delete
```

```
$ ticloud serverless sql-user delete --cluster-id <クラスターID> \  
--user <ユーザー名>
```

▶ 月額限度額の操作

月額限度額を設定する

```
$ ticloud serverless spending-limit
```

```
$ ticloud serverless spending-limit --cluster-id <クラスターID> \  
--monthly <USセント>
```

月額限度額を確認する

jq コマンドは別途インストール^{注1}が必要です。

```
$ ticloud serverless describe --cluster-id <クラスターID> | \  
jq -r '.spendingLimit.monthly'
```

▶ データインポートの操作

データをインポートするタスクを作成する

```
$ ticloud serverless import start
```

ローカルファイルからデータをインポートするタスクを作成する

```
$ ticloud serverless import start --cluster-id <クラスターID> \  
--local.file-path <ファイルパス> --file-type <ファイルタイプ> \  
--local.target-database <インポート対象のデータベース> \  
--local.target-table <インポート対象のテーブル>
```

Amazon S3からデータをインポートするタスクを作成する

```
$ ticloud serverless import start --cluster-id <クラスターID> \  
--source-type S3 --s3.uri <s3 URI> --s3.role-arn <ロールARN> \  
--file-type <ファイルタイプ>
```

Google Cloud Storageからデータをインポートするタスクを作成する

```
$ ticloud serverless import start --cluster-id <クラスターID> \  
--source-type GCS --gcs.uri <GCS URI> \  
--gcs.service-account-key <サービスアカウントキー> \  
--file-type <ファイルタイプ>
```

注1 <https://jqlang.github.io/jq/download/>

Azure Blob Storage からデータをインポートするタスクを作成する

```
$ ticloud serverless import start --cluster-id <クラスターID> \  
--source-type AZURE_BLOB --azblob.uri <Azure Blob Storage URI> \  
--azblob.sas-token <SASTークン> \  
--file-type <ファイルタイプ>
```

インポートタスクの一覧を表示する

```
$ ticloud serverless import list
```

```
$ ticloud serverless import list --cluster-id <クラスターID>
```

インポートタスクの詳細を表示する

```
$ ticloud serverless import describe
```

```
$ ticloud serverless import describe --cluster-id <クラスターID> \  
--import-id <インポートID>
```

インポートタスクを停止する

```
$ ticloud serverless import cancel
```

```
$ ticloud serverless import cancel --cluster-id <クラスターID> \  
--import-id <インポートID>
```

データエクスポートの操作

データをエクスポートするタスクを作成する

```
$ ticloud serverless export create
```

ローカルファイルにデータをエクスポートするタスクを作成する

```
$ ticloud serverless export create --cluster-id <クラスターID>
```

Amazon S3にデータをエクスポートするタスクを作成する

```
$ ticloud serverless export create --cluster-id <クラスターID> \  
--target-type S3 --s3.uri <S3 URI> \  
--s3.access-key-id <アクセスキーID> --s3.secret-access-key <シークレットアクセスキー>
```

Google Cloud Storageにデータをエクスポートするタスクを作成する

```
$ ticloud serverless export create --cluster-id <クラスターID> \  
--target-type GCS --gcs.uri <GCS URI> \  
--gcs.service-account-key <サービスアカウントキー>
```


Azure Blob Storageにデータをエクスポートするタスクを作成する

```
$ ticloud serverless export create --cluster-id <クラスターID> \  
  --target-type AZURE_BLOB --azblob.uri <Azure Blob Storage URI> \  
  --gcs.service-account-key <サービスアカウントキー>
```

エクスポートタスクの一覧を表示する

```
$ ticloud serverless export list
```

```
$ ticloud serverless export list --cluster-id <クラスターID>
```

エクスポートタスクの詳細を表示する

```
$ ticloud serverless export describe
```

```
$ ticloud serverless export describe --cluster-id <クラスターID> \  
  --export-id <エクスポートID>
```

エクスポートタスクを停止する

```
$ ticloud serverless export cancel
```

```
$ ticloud serverless export cancel --cluster-id <クラスターID> \  
  --export-id <エクスポートID>
```

エクスポートされたデータをダウンロードする

```
$ ticloud serverless export download
```

```
$ ticloud serverless export download --cluster-id <クラスターID> \  
  --export-id <エクスポートID>
```

プロジェクトの確認

プロジェクトの一覧を表示する

```
$ ticloud project list
```

TiDB Botに問い合わせる

TiDB Botに問い合わせる

```
$ ticloud ai
```

```
$ ticloud ai -q <質問文(日本語も可能)>
```

おわりに

本書では、TiDBというNewSQLデータベースの概要から実践的な運用方法まで幅広く解説してきました。TiDBについてまだ知らない方も、すでにTiDBをさわっている方も、本書を通じて新たな知見を得ていただけることを期待しています。

本書を書くにあたり、著者自身もTiDBについて大いに学ぶ機会となりました。もっと紹介したい機能やテクニックはありますが、最初にこれだけだと思うことは盛り込めたと思います。機会があれば、イベントや勉強会でさらに深くTiDBについて学び合えたらと考えています。

TiDBは今後もさらなる進化を続けていくことでしょう。クラウドネイティブな環境に最適化されたデータベースとして、新しい機能の追加やパフォーマンスの向上だけにとどまらず、AIとの統合やさらなる自動化機能の充実など、技術の進化に合わせて発展していくことも期待できます。

最後になりましたが、本書をお読みいただき、誠にありがとうございます。本書がTiDBへの理解を深め、皆様のデータベース運用における課題解決の一助となれば幸いです。TiDBの持つ可能性を最大限に活かし、皆様のシステムがより堅牢で効率的なものになることを願っています。

さらなる学習リソース

本書を通じてTiDBに興味を持っていただいた方に向けて、さらに理解を深めるための学習リソースをいくつか紹介します。今後も継続的にTiDBについて学びたい方は、ぜひ活用してください。

- **TiDB公式オンライントレーニング**
<https://pingcap.co.jp/education/>
- **イベントとウェビナー**
<https://pingcap.co.jp/event/>
- **TiDB Community Slack(#tidb-japan チャンネル)**
<https://slack.tidb.io/invite?team=tidb-community&channel=everyone>
- **TiDB Community Discord サーバー(#japan-community チャンネル)**
<https://discord.com/invite/vYU9h56kAX>

コミュニティ活動

有志によるTiDBコミュニティ活動としてTiUG(TiDB User Group)があります。定期的にイベントが開催されていますので、こちらも興味がありましたらぜひ参加してください。

- **TiUG 活動ページ**
<https://tiug.connpass.com/>
- **TiUG Discord サーバー**
<https://discord.gg/TCAPGanAAh>

索引

記号 数字

_tidb_rowid	24, 296
--mode=tikv-slim	69
--tag	56, 58, 62, 68
--without-monitor	62, 89
2要素認証	161

A

alertmanager_servers	88, 312
ANALYZE TABLE	103
Apache Kafka	232
Apache Pulsar	232
APIキー	180, 301
AS OF TIMESTAMP	12, 269
AUTO_ID_CACHE	292
AUTO_INCREMENT	289, 292
AUTO_RANDOM	163, 289, 293

B

BACKGROUND	279
BDRモード	
(Bi-Directional Replication mode)	232
BIGINT	163, 291, 293
BURSTABLE	275

C

canal-json	231
CDC(Change Data Capture)	225
cdc_servers	230
Chat2Query	163
CLUSTER_SLOW_QUERY	272, 320
CNCF	
(Cloud Native Computing Foundation)	34
CNCJ(Cloud Native Community Japan)	35
component_versions	42

CR→カスタムリソース
CRD→カスタムリソース定義

Cron	210

D

DDL(Data Definition Language)	234, 237, 279, 283
Distributed SQL	8
DML(Data Manipulation Language)	237, 284
DMR(Development Milestone Release)	2

E

Executor	39
EXPLAIN ANALYZE	282

F

FLASHBACK	13
Free Cluster	150, 184

G

graceful-wait-before-shutdown	41, 66, 131
grafana_servers	88, 310

H

Helmチャート	114, 141
Helmリポジトリ	114
HNSWインデックス	298
HTAP(Hybrid Transactional/ Analytical Processing)	15

I

IMPORT INTO	220, 268
indexID	24

K

Keyspace名	151

kind(コマンド).....106, 122

L

limits.....128

LSM ツリー (Log Structured Merge-Tree) 35

LTS(Long Term Support) 2

M

min-blob-size.....36

minikube.....106, 110, 122

monitoring_servers.....88, 309

MPP(Massively Parallel Processing) 27

MVCC
(Multi-Version Concurrency Control) 12, 270

N

NewSQL.....3

O

OLAP(Online Analytical Processing) 15

OLTP(Online Transaction Processing) 15

OSS Insight.....2

P

pd_enable_follower_handle_region.....33

PD クラスター.....20, 33, 41, 125

PRIORITY.....275

Process_keys.....271

PROCESSED_KEYS.....277

Q

QPS(Queries Per Second).....14, 43, 84, 317

QUERY_LIMIT.....277

R

Raft.....31, 35, 37, 153

RBAC(Role-Based Access Control).....314

READ_FROM_STORAGE.....25

Regional high availability.....151

Replication Capacity Units(RCU).....239

request_unit_by_group.....275

requests.....128

RESOURCE_GROUPS.....275

RFC 3339.....186

RocksDB.....35

RowID.....21, 296

RU_PER_SEC.....275

RU→リクエストユニット

RUNAWAY_WATCHES.....278

S

Scalable Cluster.....150, 184

SEM(Security Enhanced Mode).....313

server_configs.....79

SLOW_QUERY.....272, 320

SQL ユーザー.....187

SST(Sorted String Table).....194, 197

StatefulSet.....127

STATEMENTS_SUMMARY.....318

STATEMENTS_SUMMARY_HISTORY.....318

T

tableID.....21

TiCDC.....16, 20, 225

TiDB Cloud Dedicated.....17, 153

TiDB Cloud Serverless.....17, 150

TiDB Cloudのサービス比較.....190

TiDB Community.....17

TiDB Dashboard.....84, 313

TiDB Playground.....52

TiDB Self-Managed.....17

tidb_enable_dist_task.....220

tidb_enable_instance_plan_cache	14
tidb_enable_metadata_lock	285
tidb_enable_resource_control	283
tidb_enable_slow_log	321
tidb_enable_top_sql	318
tidb_gc_enable	274
tidb_gc_life_time	218, 271
tidb_gc_run_interval	273
tidb_isolation_read_engines	25
tidb_md1_view	285
TIDB_PARSE_TSO()	29
TIDB_PARSE_TSO_LOGICAL()	29
TIDB_PK_TYPE	22
tidb_resource_control_strict_mode	280
tidb_runaway_queries	278
tidb_slow_log_threshold	321
tidb_snapshot	12
tidb_stmt_summary_enable_persistent	319
tidb_tso_client_rpc_mode	28
tidb_ttl_job_enable	287
tidb_ttl_job_history	288
tidb_ttl_running_tasks	288
TiDB クラスター	20
TiDB の採用事例	15
TiFlash	15, 25, 37
TIFLASH_REPLICA	37, 328
TiProxy	20, 40
Titan (RocksDB)	36
TIUP_HOME	51
Top SQL	138, 317
Total_keys	271
TPC-C	324
TPC-H	324
TSO (TimeStamp Oracle)	27
TTL (Time to Live)	16, 286
TTL_JOB_INTERVAL	287

U

UUID	227

Y

YCSB	324

Z

Zonal high availability	151

あ

アラート機能	154, 171
インデックス ID	24
オンライン DDL ステートメント	279

か

カスタムリソース (CR)	112
カスタムリソース定義 (CRD)	112
カラムファミリー	36
キーバリュー形式	21
キャッシュテーブル	14
行指向ストレージ	15
クォーラム	32
クラスター化テーブル	21, 293
月額限度額	150, 184
コピーオンライト	168
コプロセッサキャッシュ	14
コンピュータ層	7, 14, 21
コンポーネントの停止の順番	91

さ

サーバーレス	150
システムテーブル	202
実行計画	14, 282, 317
シャードビット	295
スケールアウト	5, 41, 63, 80

スケールアップ	5, 11, 128
スケールイン	12, 41, 63, 80
ステータスページ	175
ストレージクラスター	20, 32, 34
ストレージ層	7
スナップショットバックアップ	171, 194
スプリットブレイン	32, 125
スローログ	142, 144

た

タグ(識別子)機能	56
チェンジフィード	225, 232
デュアルリージョンバックアップ	171
テーブルID	21
統計情報	103, 196
トポロジーファイル	70

は

バージョンアップ	10, 42, 96, 139
バージョン文字列	53, 60
ハートビート	32
非クラスター化テーブル	21, 296
ピンちゃん	4
フォロワー	31
プッシュダウン	38
物理インポートモード	219, 268
ベクトル検索	152, 298
ポイントインタイムリストア	171, 197, 202, 212
補完機能	48
ホットスポット	27, 288

ま

マニフェストファイル	112, 117
マルチテナント	15, 151, 274

や

予約ビット	295

ら

ラーナー	32, 37
リージョン(ストレージレイヤー)	10, 35, 288
リーダー	31, 35, 77, 87
リクエストユニット(RU)	15, 150, 162, 274, 282
リソースグループ	274
列指向ストレージ	15
ログバックアップ	171, 197
論理インポートモード	219

逆引きコマンドリファレンスの

索引

TiUPをインストールする	336	TiUPでインストールしたすべてのコンポーネントをアンインストールする	338
TiUPをアップデートする	336	TiUPのシェル補完を有効にする(Bash)	338
TiUPのバージョンを確認する	336	TiUPのシェル補完を有効にする(Z shell)	338
TiUPのヘルプを表示する	336	TiUPのシェル補完を有効にする(fish)	338
TiUPのサブコマンドのヘルプを表示する	336	TiUPのシェル補完を有効にする(PowerShell)	338
TiUPをアンインストールする	336	TiUPの操作履歴を表示する(100件)	338
TiUPのコンポーネント一覧を表示する	336	TiUPの操作履歴を表示する(N件)	338
インストール済みのTiUPのコンポーネント一覧を表示する	336	TiUPの操作履歴をすべて表示する	338
TiUPのコンポーネント一覧をインストールしている場合はバージョンと共に表示する	337	TiUPの操作履歴をJSON形式で表示する	338
TiUPのコンポーネントの利用可能なバージョンを表示する	337	TiUPの60日間より前の操作履歴を削除する	338
インストール済みのTiUPのコンポーネントのバージョンを表示する	337	TiUPのN日間より前の操作履歴を削除する	339
TiUPのコンポーネントをインストールする	337	TiUPのすべての操作履歴を削除する	339
TiUPのコンポーネントをバージョン指定してインストールする	337	TiDBやTiDB Playgroundの状態を確認する	339
TiUPのコンポーネントを最新のリリース版にアップデートする	337	トポロジーファイルのテンプレートを表示する	339
すべてのTiUPのコンポーネントを最新のリリース版にアップデートする	337	任意のホストを指定してトポロジーファイルのテンプレートを表示する	339
TiUPのコンポーネントを最新の開発版にアップデートする	337	TiDBを構築する前に確認する	339
すべてのTiUPのコンポーネントを最新の開発版にアップデートする	337	TiDBを構築する前に確認項目の自動修正を試みる	339
使用するTiUPのコンポーネントのバージョンを切り替える	337	TiDBを構築する	340
TiUPのコンポーネントをアンインストールする	337	TiDBの一覧を表示する	340
TiUPのコンポーネントのすべてのバージョンをアンインストールする	338	TiDB全体を起動する	340
		データベースの初期パスワードを生成してTiDBを起動する	340
		TiDBの特定のノードを起動する	340
		TiDBの特定のロールを起動する	340
		TiDB全体を停止する	340
		TiDBの特定のノードを停止する	340

TiDB の特定のロールを停止する	340	TiDB のメタデータを復元する	343
TiDB 全体を再起動する	340	TiDB Playground を起動する	343
TiDB の特定のノードを再起動する	341	バージョン指定して TiDB Playground を起動する	343
TiDB の特定のロールを再起動する	341	タグを指定して TiDB Playground を起動する	343
OS 起動時に TiDB のプロセスの 自動起動を有効にする	341	モニタリング環境を無効化して TiDB Playground を起動する	343
OS 起動時に TiDB のプロセスの 自動起動を無効にする	341	コンポーネントの台数を指定して TiDB Playground を起動する	343
TiDB をスケールアウトする前に確認する	341	TiProxy を有効化して TiDB Playground を起動する	344
TiDB をスケールアウトする前に 確認項目の自動修正を試みる	341	DM を有効化して TiDB Playground を起動する	344
TiDB をスケールアウトする	341	起動しているプロセスを確認する	344
TiDB をスケールインする	341	コンポーネントをスケールアウトする	344
TiDB を削除する	341	コンポーネントをスケールインする	344
データを保持して TiDB を削除する	341	TiDB Playground へ接続する	344
TiDB をバージョンアップする	342	TiDB Playground のデータを削除する	344
オフラインで TiDB をバージョンアップする	342	すべての TiDB Playground の データを削除する	344
TiDB の情報を表示する	342	デモ用データを投入する	344
TiDB の情報をバージョンと共に表示する	342	デモ用データを削除する	344
TiDB のバージョンのみ表示する	342	TPC-C用のデータを投入する	344
TiDB のダッシュボードの URL のみ表示する	342	TPC-C を実行する	345
TiDB の現在のトポロジー情報を表示する	342	TPC-C用のデータを削除する	345
TiDB の監査ログを表示する	342	TPC-H用のデータを投入する	345
TiDB の監査ログの詳細を表示する	342	TPC-H を実行する	345
TiDB の監査ログの内容を再実行する	342	TPC-H用のデータを削除する	345
TiDB の60日間より前の監査ログを削除する	343	CH (TPC-C と TPC-H の混合) 用の データを投入する	345
TiDB のN日間より前の監査ログを削除する	343	CH (TPC-C と TPC-H の混合) を実行する	345
TiDB のメタデータをバックアップする	343		
TiDB のメタデータを ファイル名を指定してバックアップする	343		

CH(TPC-CとTPC-Hの混合)用の データを削除する	345	プロファイルを切り替える	348
YCSBのTiDB クラスター用の データを投入する	345	プロファイルを削除する	348
TiDB クラスターに対してYCSBを実行する	345	OAuth 認証する	348
YCSBのTiKV用のデータを投入する	345	OAuth 認証を解除する	348
TiKVに対してYCSBを実行する	345	有効なリージョンを表示する	348
pd-ctl	346	TiDB Cloud Serverlessを作成する	348
tidb-ctl	346	TiDB Cloud Serverlessの一覧を表示する	348
tikv-ctl	346	TiDB Cloud Serverlessの詳細を表示する	349
tidb-lightning-ctl	346	TiDB Cloud Serverlessの表示名を変更する	349
cdc	346	TiDB Cloud Serverlessに接続する	349
dmctl	346	TiDB Cloud Serverlessを削除する	349
etcdctl	346	ブランチを作成する	349
ticloud	346	ブランチの一覧を表示する	349
TiDB Cloud CLIをインストールする	346	ブランチに接続する	350
TiDB Cloud CLIをアップデートする	347	ブランチを削除する	350
TiDB Cloud CLIのヘルプを表示する	347	SQL ユーザーを作成する	350
TiDB Cloud CLIのサブコマンドの ヘルプを表示する	347	SQL ユーザー一覧を表示する	350
TiDB Cloud CLIをアンインストールする	347	SQL ユーザーのパスワードを変更する	350
TiDB Cloud CLIのシェル補完を有効にする (Bash)	347	SQL ユーザーのロールを変更する	350
TiDB Cloud CLIのシェル補完を有効にする (Z shell)	347	SQL ユーザーを削除する	351
TiDB Cloud CLIのシェル補完を有効にする (fish)	347	月額限度額を設定する	351
TiDB Cloud CLIのシェル補完を有効にする (PowerShell)	347	月額限度額を確認する	351
プロファイルを作成する	347	データをインポートするタスクを作成する	351
プロファイルの一覧を表示する	348	ローカルファイルからデータを インポートするタスクを作成する	351
		Amazon S3からデータを インポートするタスクを作成する	351
		Google Cloud Storageからデータを インポートするタスクを作成する	351

Azure Blob Storageからデータを インポートするタスクを作成する	352
インポートタスクの一覧を表示する	352
インポートタスクの詳細を表示する	352
インポートタスクを停止する	352
データをエクスポートするタスクを作成する	352
ローカルファイルにデータを エクスポートするタスクを作成する	352
Amazon S3にデータを エクスポートするタスクを作成する	352
Google Cloud Storageにデータを エクスポートするタスクを作成する	352
Azure Blob Storageにデータを エクスポートするタスクを作成する	353
エクスポートタスクの一覧を表示する	353
エクスポートタスクの詳細を表示する	353
エクスポートタスクを停止する	353
エクスポートされたデータをダウンロードする	353
プロジェクトの一覧を表示する	353
TiDB Botに問い合わせる	353

著者プロフィール

長谷川 誠（はせがわ まこと）

インフラエンジニアとして豊富な経験を持ち、これまでに大規模なクラウド移行プロジェクトやオンプレミスとパブリッククラウドのハイブリッド環境構築など、さまざまなインフラ構築、運用を経験。特に可用性とスケーラビリティの高いシステム設計に注力し、多くの信頼性の高いインフラ環境を実現してきた。その豊富な経験を基に、クラウドネイティブ技術やモダナイゼーションにも精通し、さまざまなコミュニティ活動およびオープンソースへの貢献に積極的に取り組んでいる。また、国内外でのカンファレンス登壇も積極的に行っている。

2021年よりPingCAP日本法人において技術顧問を務め、TiDBユーザーコミュニティを立ち上げ、多くの技術情報を発信している。

共著書として『Kubernetesの知識地図』(<https://gihyo.jp/book/2023/978-4-297-13573-7>)を執筆。

趣味はベースの演奏であり、技術と音楽の双方に情熱を持つエンジニアとして活動中。

監修者プロフィール

本多 康夫（ほんだ やすお）

データベース製品のコンサルタント、DBA、Webアプリケーションエンジニア、BI製品のテクニカルサポートエンジニアなど、開発から運用まで幅広い業務に携わる経験を持つ。2020年に始まったDatabase Internals 輪読会をきっかけにTiDBを知り、その後2021年にPingCAP株式会社に入社、現在はTechnical Support EngineerとしてTiDBユーザーのサポートや資格試験、トレーニング資料の準備にあたる。また、データベースに関連する深い知識やこれまでの経験を生かし、国内におけるTiDBの普及活動を積極的に行っている。

装丁.....石田 昌治 (株式会社マップス)
本文デザイン、DTP、図版.....石田 昌治 (株式会社マップス)
編集.....菊池 猛

タイデービー じっ せんにゆうもん
TiDB 実践入門
アールデービーエムエス か だ い かいしょう ニューエスキューエル
RDBMSのスケール課題を解消するNewSQL

2025年5月16日 初版 第1刷発行



著者.....は せ が わ まこと **長谷川 誠**
監修.....ほんだ やすお ピンキャップ かぶしきがいしゃ **本多 康夫 PingCAP 株式会社**
発行者.....片岡 巖
発行所.....株式会社技術評論社
東京都新宿区市谷左内町 21-13
電話 03-3513-6150 販売促進部
 03-3513-6177 第5編集部
印刷／製本 TOPPAN クロレ株式会社

- 定価はカバーに表示してあります。
- 本書の一部または全部を著作権法の定める範囲を越え、無断で複製、複製、転載、あるいはファイルに落とすことを禁じます。
- 造本には細心の注意を払っておりますが、万一、乱丁(ページの乱れ)や落丁(ページの抜け)がございましたら、小社販売促進部までお送りください。送料小社負担にてお取り替えいたします。

©2025 長谷川誠
ISBN 978-4-297-14832-4 C3055
Printed in Japan

● お問い合わせ

本書の内容に関するご質問につきましては、下記の宛先まで書面にてお送りいただくか、小社ホームページの該当書籍コーナーからお願いいたします。お電話によるご質問、および本書に記載されている内容以外のご質問には、一切お答えできません。あらかじめご了承ください。
また、ご質問の際には「書籍名」と「該当ページ番号」、「お客様のパソコンなどの動作環境」、「お名前とご連絡先」を明記してください。

【宛先】
〒162-0846
東京都新宿区市谷左内町 21-13
株式会社技術評論社 第5編集部
『TiDB 実践入門』質問係
URL <https://gihyo.jp/book/>

お送りいただきましたご質問には、できる限り迅速にお答えするよう努力しておりますが、ご質問の内容によってはお答えするまでに、お時間をいただくこともございます。回答の期日をご指定いただいても、ご希望にお応えできかねる場合もありますので、あらかじめご了承ください。
なお、ご質問の際に記載いただいた個人情報は質問の返答以外の目的には使用いたしません。また、質問の返答後は速やかに破棄いたします。



本コンテンツは、Gihyo Digital Publishing において、
電子データとして再発行されたものです。

電子初版発行日 2025 年 05 月 07 日

最新版発行日 2025 年 05 月 07 日

電子版ISBN 978-4-297-14833-1