AAAタイトルを支えるカプコン共通基盤

～TiDBによるスケーラブルな無停止基盤のつくりかた～

自己紹介

福井　勝貴

CAPCOM

株式会社カプコン
CS制作統括 CSシステム開発部

2023年にカプコンに入社、共通基盤インフラの設計・構築・運用を担当。

以前はCDN事業者にて、ネットワーク・インフラ・動画広告関連・サポート領域のエンジニアとして従事。
その後、ゲーム会社にてソーシャルゲームや共通基盤のインフラ設計・構築・運用、開発サポート、ブリッジエンジニアを担当。
現在はカプコンに所属。

林　正記

TiDB Powered by PingCAP

PingCAP株式会社
Japan CTO

外資系メーカーにて、プロダクトマネージャ・プロダクトマーケティングとして市場開拓に従事。その後、同社にてプリセールスとして直販・間販を通じた提案活動を実施。
PingCAP入社後は、プリセールスとしての活動だけではなくJapan CTOとして国内ユーザーのニーズを海外R&Dチームにフィードバックし、製品の認知度を高める活動を積極的にしている。

目次

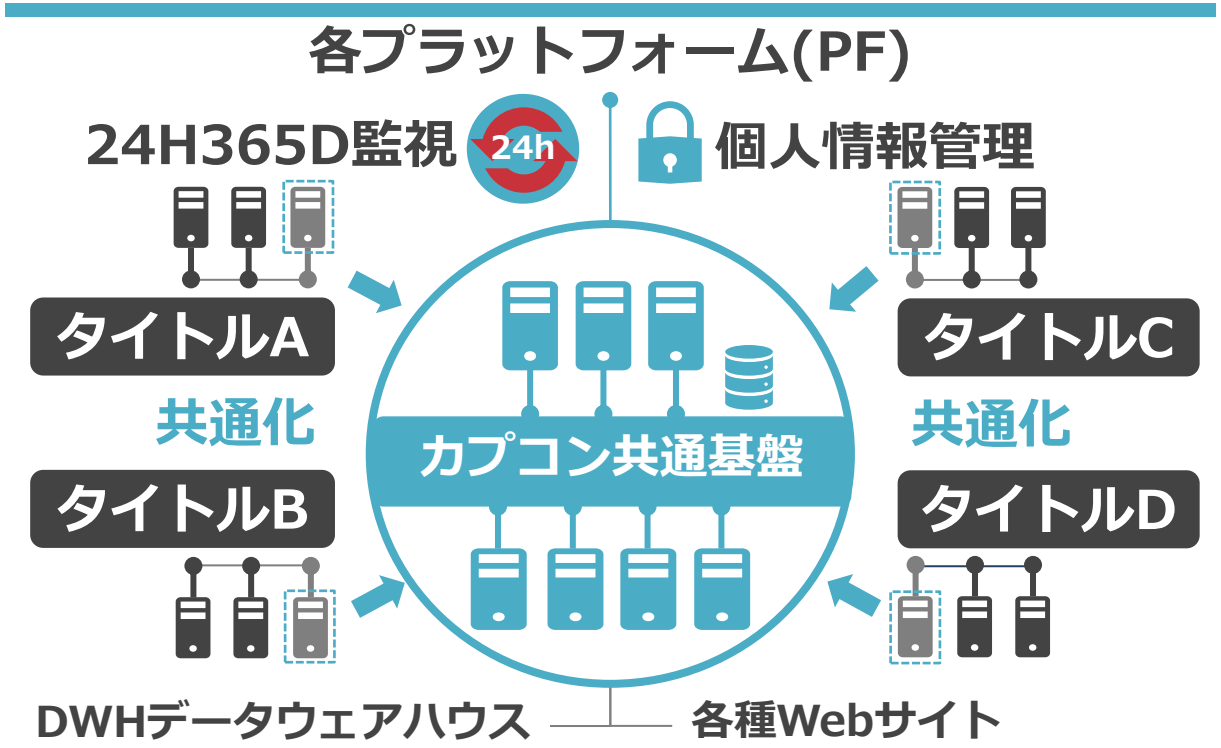
1．カプコン共通基盤とは

概要／体制

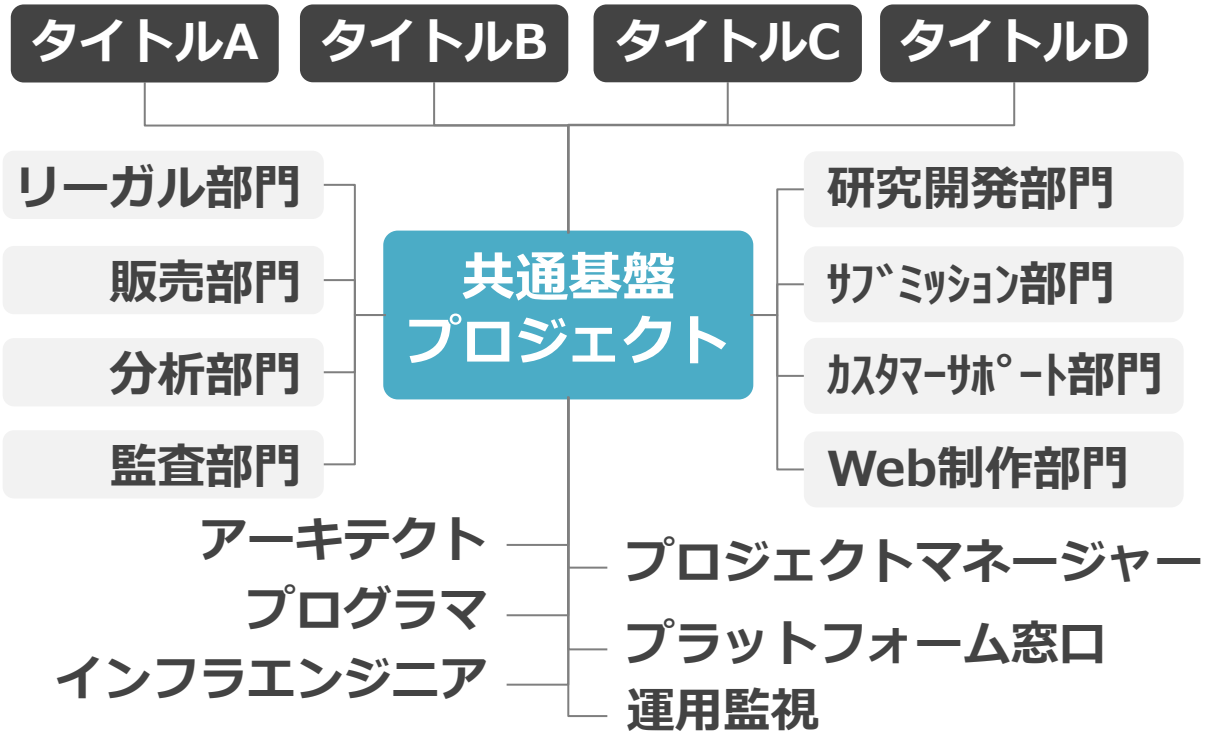
カプコン共通基盤とは

クロスプラットフォームやゲーム内通貨の実現には、ユーザー情報を扱うサーバーが必要です。
カプコンではこれを「共通基盤」として開発・運用し、個人情報管理や24時間監視を
実施することで、各タイトルがクリエイティブなものづくりに専念できることを目指しています。

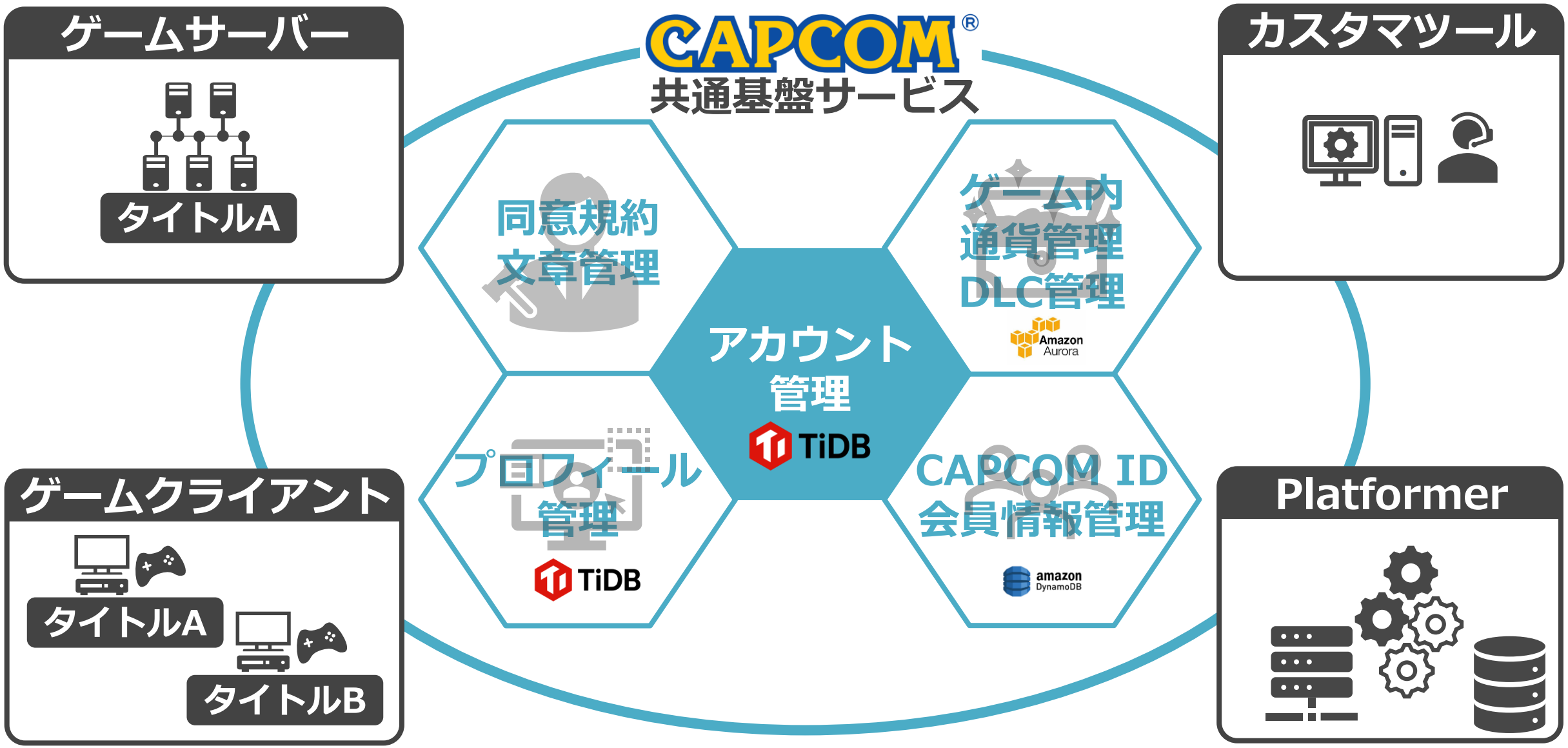
概要



体制



システム概要


ゲームサーバー
タイトルA
CAPCOM®
共通基盤サービス
同意規約
文章管理
ゲーム内
通貨管理
DLC管理
Amazon Aurora
アカウント
管理
TiDB
プロフィール
管理
TiDB
CAPCOM ID
会員情報管理
amazon DynamoDB
カスタマツール
ゲームクライアント
タイトルA
タイトルB
Platformer

共通基盤の環境について

共通基盤が利用する環境

- 共通基盤－開発
- 共通基盤－負荷試験

作業承認が必要な環境

- 検証用本番データ

タイトルへ提供している環境

作業承認が不要な環境

- タイトル－開発
- QA
- タイトル－負荷試験
- セキュリティ診断

作業承認が必要な環境

- 本番
- ステージング
- プラットフォーム審査

補足

開発環境

共通基盤の機能開発中に利用する環境とタイトル開発用の2種類あり。
※**共通基盤－開発**はローカル環境のDocker Composeを利用。

負荷試験環境

独立して分けているわけではなく空いている環境を利用・提供している。
例：負荷試験環境１セット、２、３のようにあらかじめいくつか用意あり

2．TiDB導入の背景と目的

TiDB導入の背景

導入前の課題

Amazon Aurora（MySQL）の課題

＃大規模なユーザー数を支えるにはシャーディング化が必要
＃パッチ適用・アップグレード・スペック変更によるサービスの停止を伴うメンテナンスが必要

クロスプラットフォーム向けタイトルへの対応

＃大規模なユーザー数を支えるスケーラブルなデータベースを採用したい
＃開発のコストを抑えるためにシャーディングを排除できれば嬉しい

期待する効果

1. データベースのスペック最適化によるコストの最適化と削減
2. 停止を伴うメンテナンス頻度の低下
3. シャーディングの排除

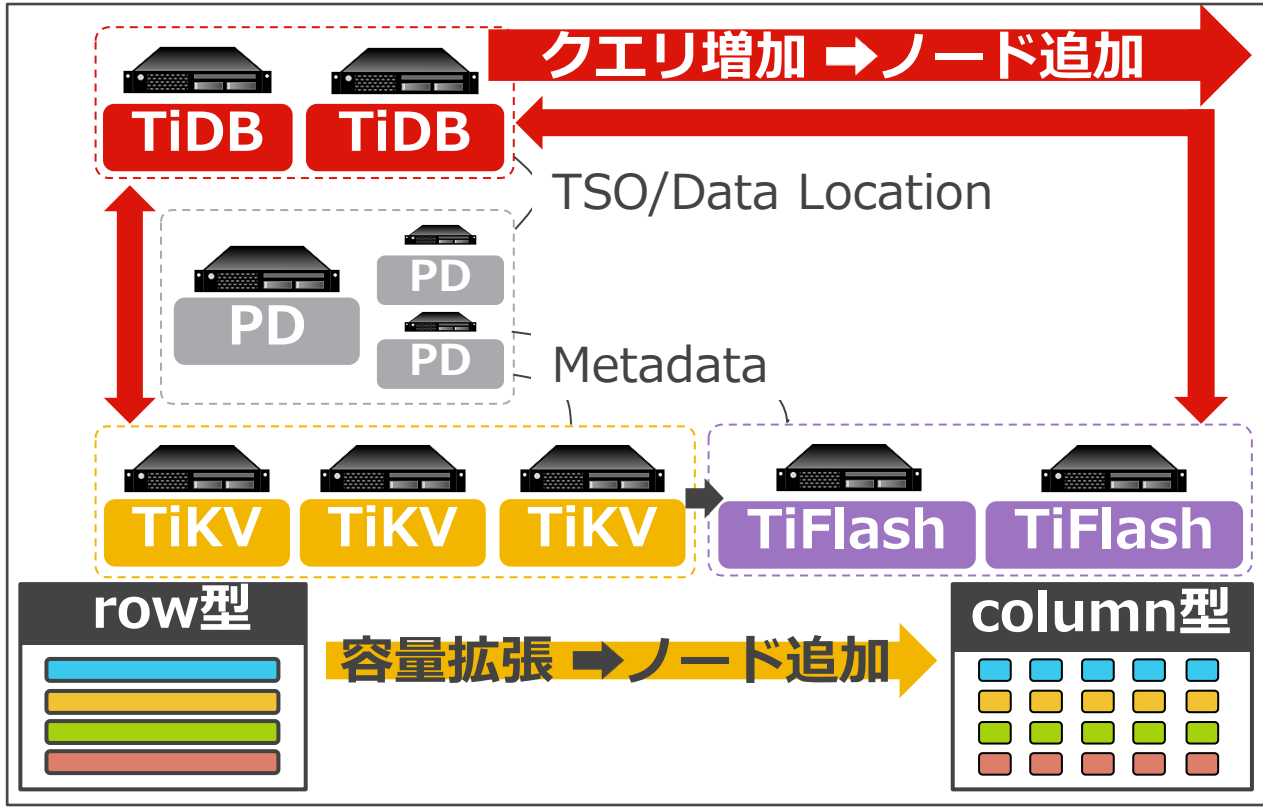
共通基盤におけるTiDBの有用性

TiDB = MySQL互換分散データベース (NewSQL) = MySQLシャーディングからの解放

TiDBのポイント

①オンラインでの性能増減
②オンラインでのメンテナンス

▶ ゲームそのものの採用だけではなく、性能・可用性が求められる
複数のゲームを支える共通基盤での採用が増加



コンピューティング TiDB

TiDB
- SQL解析を行う *MySQLプロトコルをKVに変換
- ParserやOptimizerを担う
- HTAPではTiKV(row型)かTiFlash(column型)判断

ストレージ TiKV TiFlash

TiKV
- OLTP用途の分散Key-Value Store※Rocks DB利用
- パーティショニングによるデータ分散管理
- Raftや2PCにより整合性担保

TiFlash
- HTAP利用時の追加コンポーネント
- OLAP用途のカラム型ストア

司令塔 PD

PD
- データ配置場所の管理（Region管理）
- 分散トランザクションで利用するTSOを発行

TiDBの導入における影響と懸念点

入念な事前検証が大事!!

重要

アプリのTiDB最適化や基礎検証に十分な時間をかけて検証しましょう。
タイトルへ提供している環境では影響が大きいため、TiDB導入前のテスト・検証がどれだけ実施できたかが非常に重要になります。

共通基盤が利用する環境

- 共通基盤 – 開発
- 共通基盤 – 負荷試験

作業承認が必要な環境

- 検証用本番データ

タイトルへ提供している環境

作業承認が不要な環境

- タイトル – 開発
- QA
- ステージング – 負荷試験
- セキュリティ診断

作業承認が必要な環境

- ステージング
- 本番
- シナリオテスト – 負荷試験

複数タイトルで共通利用しているため影響範囲が広い。
そのため不具合等の問題が発生した場合、
利用しているすべてのタイトル開発やQAに影響あり

TiDB導入の流れ

基本検証・追加検証・リプレースの3つに分け実施・評価すべき項目を整理し進行。

アプリのTiDB最適化	・性能試験 ・機能試験 ・データ移行試験	・実機での動作検証 ・TiDBチューニング ・高負荷試験	・リプレースの手順 ・データ移行試験 ・リプレース	リプレース
2023年12月～2月	2024年2月～4月	2024年4月～6月	2024年7月～9月	2024年9月

共通基盤が利用する環境

- 共通基盤－開発
- 共通基盤－負荷試験

作業承認が必要な環境

- 検証用本番データ

タイトルへ提供している環境

作業承認が不要な環境

- タイトル－開発
- QA
- タイトル－負荷試験
- セキュリティ診断

開発・QAなどタイトル利用環境へ導入承認
TiDB導入仮決定

作業承認が必要な環境

- ステージング
- 本番
- プラットフォーム検証

実運用環境へ導入承認
TiDB最終導入決定

3．互換性の壁：導入検討フェーズ

TiDBのMySQL互換性について

Unsupported features

- Stored procedures and functions
- Triggers
- Events
- User-defined functions
- FULLTEXT syntax and indexes #1793
- SPATIAL (also known as GIS/GEOMETRY) functions,data types and indexes #6347
- Character sets other than ascii, latin1, binary, utf8,utf8mb4, and gbk.
- Optimizer trace
- XML Functions
- X-Protocol #1109
- Column-level privileges #9766
- XA syntax (TiDB uses a two-phase commit internally, but this is not exposed via an SQL interface)
- CREATE TABLE tblName AS SELECT stmt syntax #4754
- CHECK TABLE syntax #4673
- CHECKSUM TABLE syntax #1895
- REPAIR TABLE syntax

100%MySQL互換ではないため、移行時は注意が必要

https://docs.pingcap.com/tidb/stable/mysql-compatibility/

課題：MySQL-AUTO_INCREMENT

課題

Aurora MySQLではAUTO_INCREMENTを利用
→TiDBではAUTO_INCREMENTの設定をどれに選択するか決める必要がある

検討内容

1. TiDBの推奨の設定（TiDB AUTO-INCREMENT デフォルト/Auto Random)
IDの発番が従来のMySQLのAUTO_INCREMENTとは違うため、
TiDBに切り替えた後にAuroraに切り戻すのは難しい

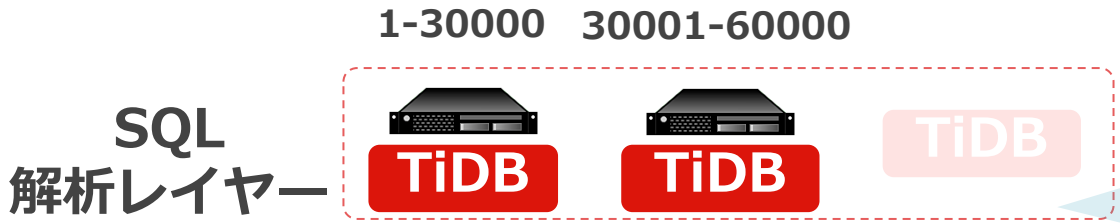
2. MySQL互換モードのAUTO_INCREMENTを設定
＃ホットスポット（※）が発生する可能性
※ストレージにデータを書き込む際に単一のポイントに負荷が集中すること
＃更新系のパフォーマンスに上限がある

※次のページでTiDBのAUTO_INCREMENTについてを解説

TiDBのAUTO_INCREMENTとは

TiDBのデフォルトAuto_increment

発番部分もスケールさせるために完全に昇順にならない



- **TiDB1**
  1-30000をキャッシュ
- **TiDB2**
  30001-60000をキャッシュ

ID(a-i)	name
1	aaa
2	bbb
30001	**ccc**
3	ddd

用途に応じた使いわけ

1. デフォルト

▶ 通常デフォルト。要件に応じて別のモードを検討する

2. MySQL互換モード（互換性重視の場合）

▶ MySQLのような昇順となる。発番性能に一定の制限あり(ホットスポットが起きやすい)

3. Auto_Random（特にWriteホットスポットを避ける場合）

▶ Insertが多くホットスポットが生じる場合はRandomな発番を本設定の実施で回避可能

その他 データ配置のポリシーを変更しホットスポットを避けるテクニックなども併用可能（Non-Clustered）

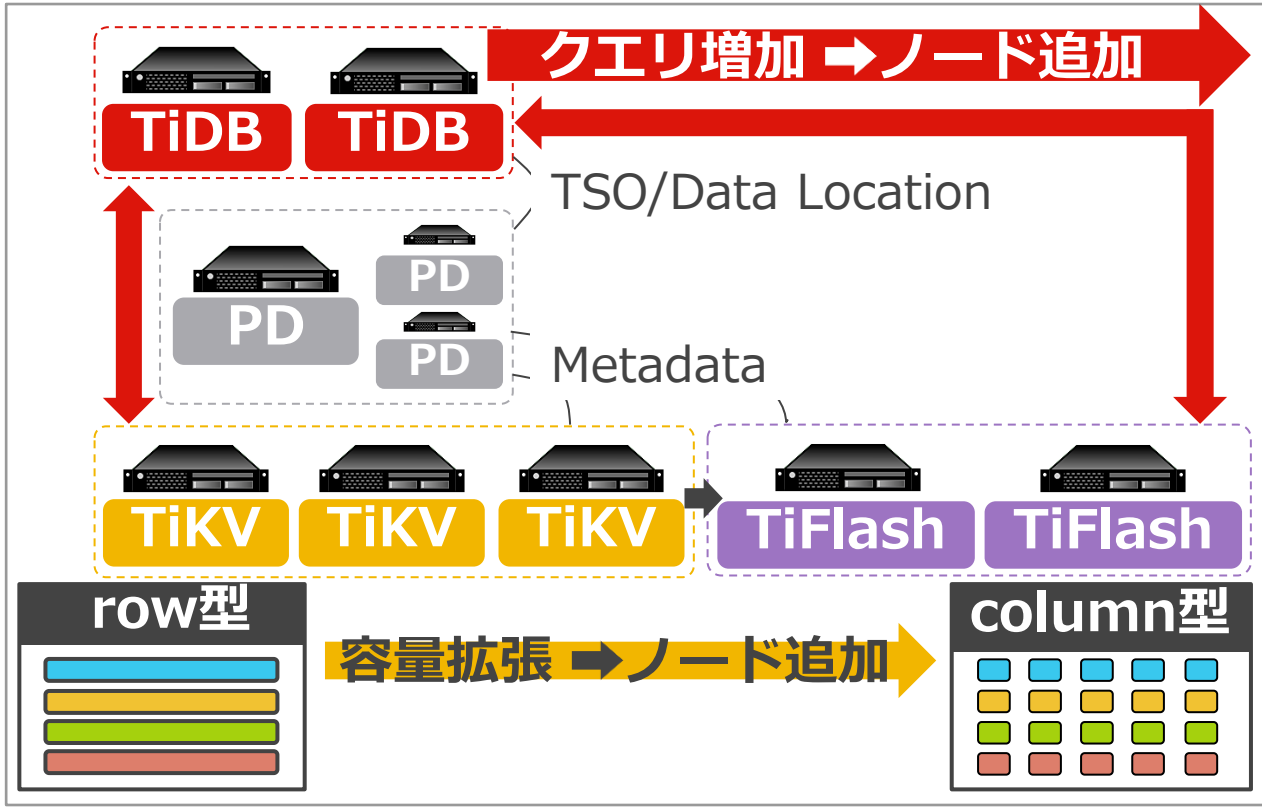
Appendix 共通基盤におけるTiDBの有用性

TiDB = MySQL互換分散データベース (NewSQL) = MySQLシャーディングからの解放

TiDBのポイント

①オンラインでの性能増減
②オンラインでのメンテナンス

▶ ゲームそのものの採用だけではなく、性能・可用性が求められる
複数のゲームを支える共通基盤での採用が増加



コンピューティング TiDB

TiDB
- SQL解析を行う *MySQLプロトコルをKVに変換
- ParserやOptimizerを担う
- HTAPではTiKV(row型)かTiFlash(column型)判断

ストレージ TiKV TiFlash

TiKV
- OLTP用途の分散Key-Value Store※Rocks DB利用
- パーティショニングによるデータ分散管理
- Raftや2PCにより整合性担保

TiFlash
- HTAP利用時の追加コンポーネント
- OLAP用途のカラム型ストア

司令塔 PD

PD
- データ配置場所の管理（Region管理）
- 分散トランザクションで利用するTSOを発行

TiDBのAUTO_INCREMENT検討＆選択

#	設定	説明	発番性能	ホットスポット	コンパチビリティ	パフォーマンス評価	スケーラビリティ評価
1	**Auto Random利用** (Clustered index+Auto-Random)	• 完全にランダムな値を発番 性能面では一番有利	◎	◎	△	**5**	**5**
2	**TiDBデフォルト** (Clustered index+Auto-increment (AUTO_ID_CACHE=30,000)	• TiDBのデフォルト • 1つのテーブルに1台のTiDBが発番し続けるケースは2000QPS程度でホットスポット判定となる	◎	○	○	4	4
3	**MySQL互換モード** (Clustered index+Auto-increment (AUTO_ID_CACHE=1))	• MySQL互換重視（移行時によく使われる） MySQL compatibility mode （Auto_id_cache=1） • 1つのテーブルに発番し続けるケースは2000QPS程度でホットスポット判定となる	○ (通常最大数万)	△	◎	3 (基準)	3 (基準)
4	**MySQL互換モード** **＋** **ホットスポット回避** (Non-Clustered index(+shard_row_id_bits) +Auto_id_cache=1)	• 内部のデータの持ち方の変化により、PKを使ったselectが非効率となり遅くなる可能性がある	○ (通常最大数万)	◎	◎	1	3

４．TiDB導入における最終評価

TiDB導入決断の最終評価

アプリケーション

〇TiDB最適化は致命的なエラーがなく、問題なし

性能

〇RPS/QPSについて想定していた結果以上の性能を確認
〇クエリレイテンシーは、Auroraと比較すると少し遅いが許容範囲内

機能

〇オンラインでのバージョンアップ可能
〇オンラインでの構成変更（拡張・縮退、スペック変更）可能
〇データの移行が正常に実施可能

費用

〇トータルとしてコストの最適化・削減が実現可能と試算

※検証項目はAppendix参照

5．工夫&パフォーマンス改善

工夫・パフォーマンス改善の概要

アプリケーションテスト

〇リストを作成するようなテスト
〇ローカル環境のDDLが遅い

TiDBのチューニング

〇Commitのレイテンシー改善

ProxySQLのチューニング

〇コネクションプーリング・維持のチューニング

アプリケーションテストの工夫

問題 PK乱数化に伴いリスト比較するような
クエリは順序違いが発生

結果 **明示的にorder Byするか集合比較するよう改修**

問題 **ローカル環境のDDLが遅い。**
Docker Compose上のTiDBがのDDLがMySQLの時と比べて時間がかかる。
〇MySQL　　約8秒
〇TiDB（v8.1.1）　約105秒

設定 **TiDBをver8.3.0を利用**
〇「@@global.tidb_enable_fast_create_table=ON」
〇「@@global.tidb_enable_dist_task=OFF」分散処理を無効
〇「@@foreign_key_checks=OFF」

結果
★設定後★
約22秒に改善

※本改善設定は、ローカル環境でのみ利用（本番環境では利用しないこと）
※Docker TiDB playgroundを利用（ https://github.com/ti-click/docker-tidb-playground）

TiKVストレージチューニング

問題

TiKVのIO遅くCommitが遅い。
TiKVのCPUが70%を超えると顕著になる。

設定1

store-io-pool-size=1（Default=0）
async io を有効化
※TiDB ver8.1.0からDefault=1

結果

Commitのレイテンシーが20～40%改善、著しく増加する現象も緩和

設定2

store-pool-size=3 (default:2)
apply-pool-size=3 (default:2)
grpc-concurrency=8 (default:5)

結果

TiKVのCPUが70%以下の状況だとQPSが10～15%増加

※TiDBはver7.5.1で検証(これらの設定はPingCAP社のみ設定可能)

コネクションプーリング・維持のチューニング

課題&解決

ProxySQLの設定によっては、ノード追加またはローリングアップデート等で再起動後のTiDBノードへの接続がない or 数が少ない状態になるのでチューニングが必用。TiDBのオンラインでの作業を活かすためには均等になることがベスト。※ProxySQL（SideCar）はAuroraの時から導入済み

コネクションプーリング・維持の効果

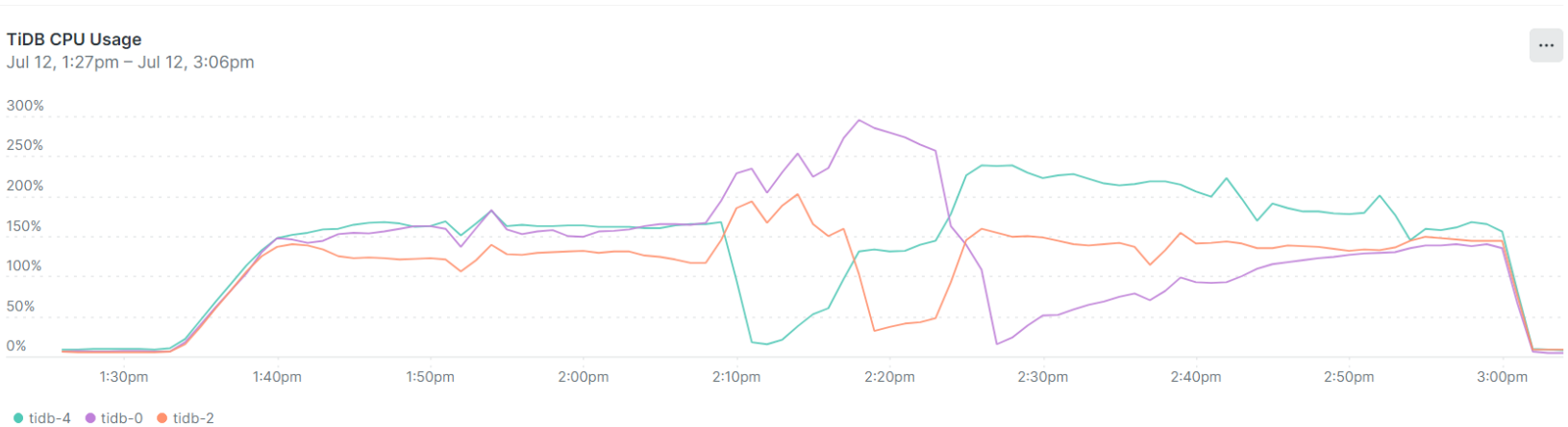
導入によりTiDBノードのCPUが25%と50%が削減。TiDBノードの台数を削減可能

困ったこと

なかなかうまく
動作する設定が
見つからない。

1回のテストに
時間がかかる。

期待通りの挙動時のCPU推移

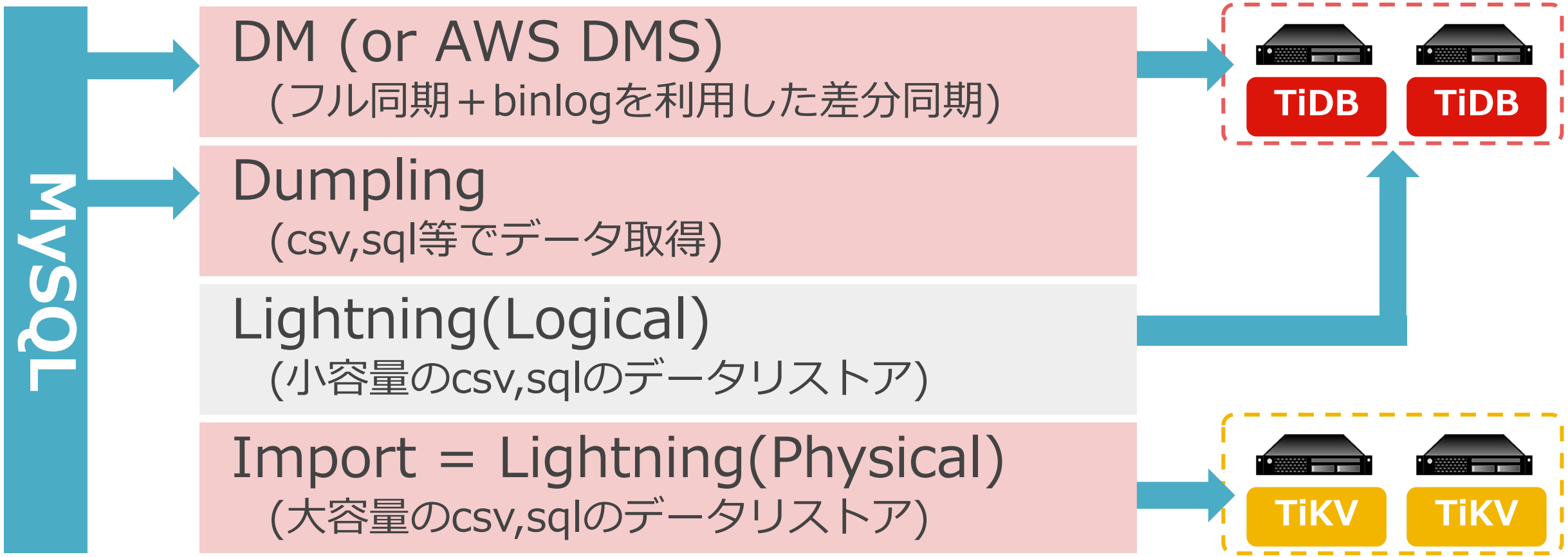


6．データ移行とリプレースの方法

TiDBの移行ツールの話

ツール

TiDBは移行関連ツールも用意している。
レプリケーションツール(DM)や高速に静的データの移行するツール(Import)など



移行ツールの選定

ツールの選定

リプレース・移行を進めていた時は、ManagedのDM機能がリリースされていない
→DumplingとImportがベストの選択
リリースされていたら選択肢の1つ

Managed DMの簡単な紹介

メイン機能

1.初期同期：dump & restore
2.差分同期：指定したbinlog positionから同期再開
※移行時間の短縮化を期待

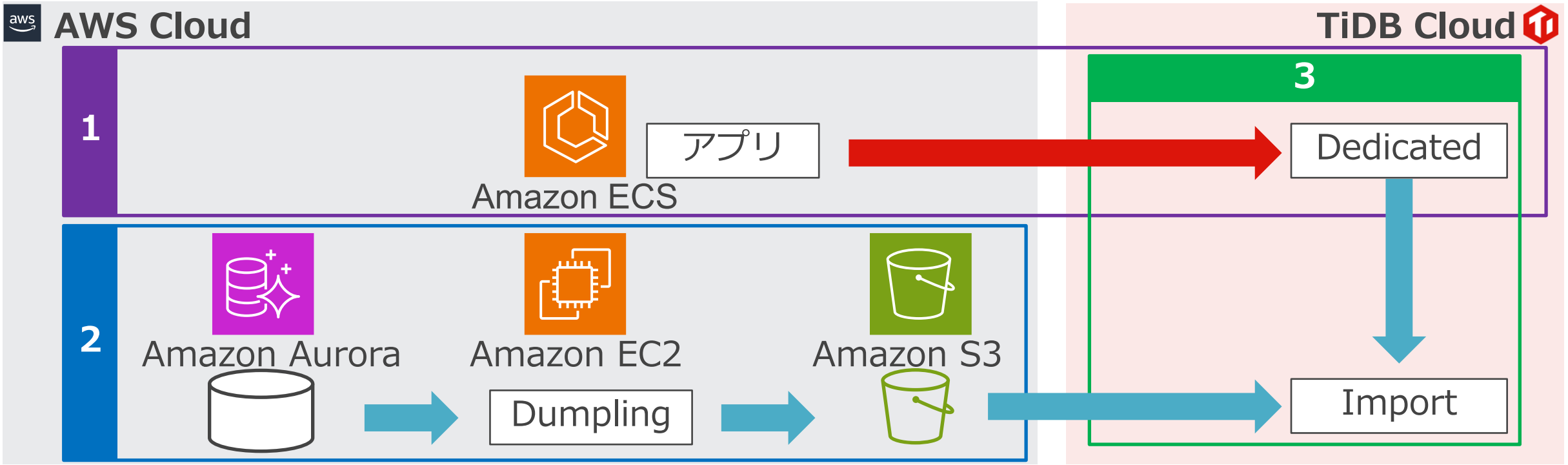
AUTO_INCREMENTのサポート

MySQL AUTO_INCREMENTからTiDB AUTO_INCREMENT（Default/Auto Random）でも同期可能
注意：MySQL側のbinlogの有効化が必用。再起動が必要なので注意

データ移行方法の検討

データ移行

1.Dedicatedにスキーマを事前作成
　アプリケーションのDBマイグレーション機能を利用
2.データはDumplingで取得しS3にアップロード
3.TiDB Cloudのクラスター機能のImportで取り込み、Dedicatedに反映



データ移行の検証ではまったこと

課題

いくつかのテーブルでImportしているときにエラーが発生
エラーの内容はプライマリーキーかユニークキーの重複と表示

要因

Aurora側のテーブルとTiDB側のテーブルのカラムの並び順が違うことが判明！！
データベースを運用していく中で発生する問題

＃**更新**（Aurora）・・・初期登録で設定した順＋追加した順
＃**新規**（TiDB）　・・・ソースコードのファイル定義順

更新	ID	Name	Company	**Workplace**	**Role**	**Division**
新規	ID	Name	Company	**Division**	**Workplace**	**Role**

普通はうまくいきそうなんだけど

Dumplingの仕様

問題点

Dumplingのデフォルトは、カラム名なしで出力することが判明
■コマンドラインのヘルプから抜粋
--complete-insert : Use complete INSERT statements that include column names

★Dumplingデフォルトの時に出力ファイル（例）

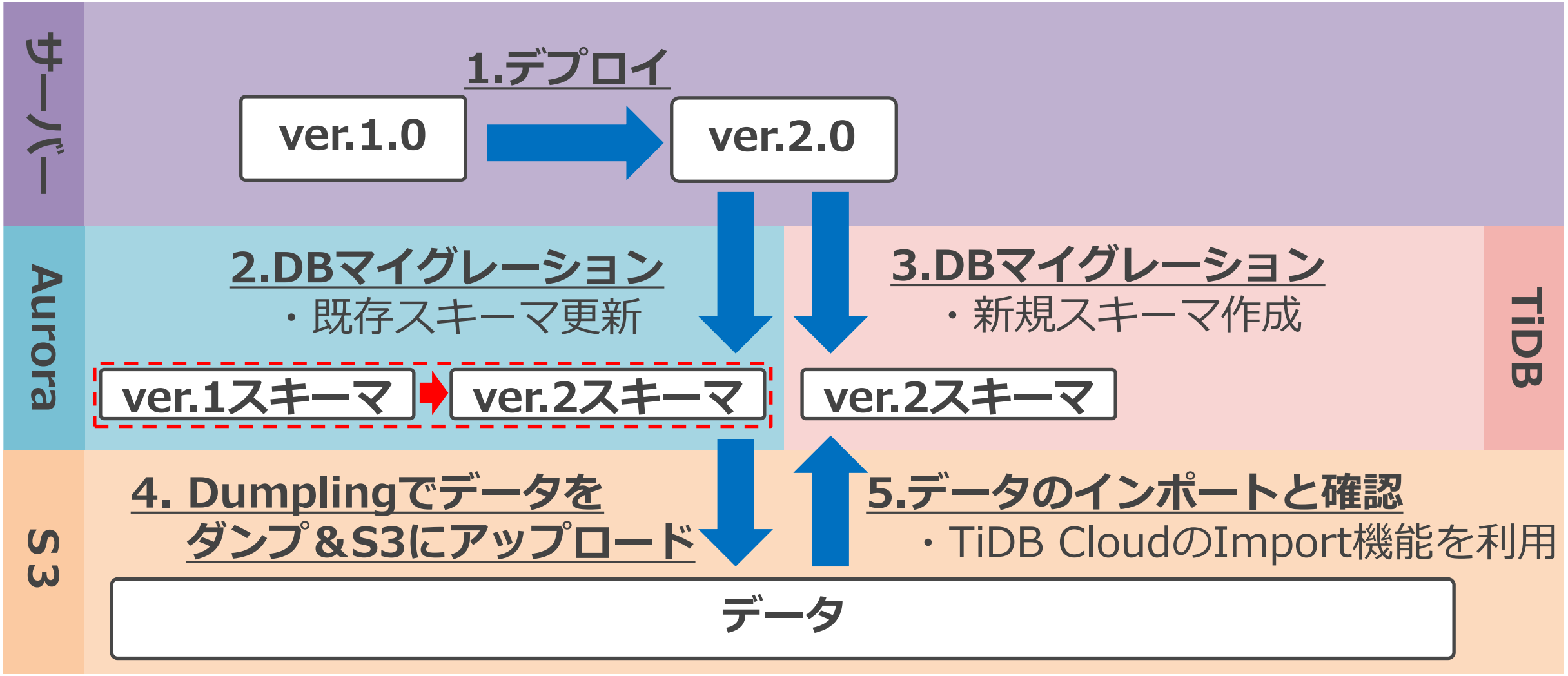
```
INSERT INTO `tidb_migration_sample` VALUES
(1,'Hayashi','pingcap','2024-03-08 06:53:40','2024-04-01 05:40:37'),
(2,'Fukui','capcom','2024-03-08 06:53:40','2024-04-01 05:40:37'),
```

★Dumplingオプション(--complete-insert付)の出力ファイル（例）

```
INSERT INTO `tidb_migration_sample` (`id`,`name`,`company`,`created`,`modified`)
VALUES
(1,'Hayashi','pingcap','2024-03-08 06:53:40','2024-04-01 05:40:37'),
(2,'Fukui','capcom','2024-03-08 06:53:40','2024-04-01 05:40:37'),
```

Dumplingを使う時は、「--complete-insert」オプションを付けることがおすすめ

移行の流れ


サーバー
1.デプロイ
ver.1.0
ver.2.0
Aurora
2.DBマイグレーション
・既存スキーマ更新
ver.1スキーマ
ver.2スキーマ
3.DBマイグレーション
・新規スキーマ作成
ver.2スキーマ
TiDB
S3
4. Dumplingでデータを
ダンプ&S3にアップロード
5.データのインポートと確認
・TiDB CloudのImport機能を利用
データ

リプレースについて

本番環境のリプレース

問題なくリプレースが実行できた。
データ移行を含むリプレース作業自体は2時間程度

リプレースの準備について

各環境において、本番を想定したリプレースの手順作成、
リプレース、手順の修正を繰り返し、精度向上をすることが重要

リプレース直前に最終確認として本番環境のデータを用いた移行テストを行う。
同時にデータ移行の必要時間を算出し、メンテナンス時間に反映

7．運用の検討


CEDEC
Computer Entertainment Developers Conference
2025

スケール戦略

TiDB・TiKVの負荷の特性

性能試験の結果より

〇TiDB・TiKVはCPU70%を超えるとパフォーマンス（レイテンシー）が劣化する傾向
〇TiKVはデータリバランスが発生するとCommitのレイテンシーが増加することもある
※データリバランスは、TiKVを増設・減設した際に発生
※TiKVは3台で1セット
〇TiKVは8Core/64GiB×12台と16Core/64GiB×6台だと16Coreのほうがパフォーマンスが良い

PingCAP社の見解

- 少ないデータ時はあまり分散しない方がよい
  16core ＞ 8core
- データリバランスの発生機会の最小化
  16core ＞ 8core
- 次のスケールタイミングのコスト
  16core ＜ 8core

★結論★
16coreが良い

スケール戦略

TiDB・TiKVのCPU70%を超えたらスケールの合図！**ファーストチョイスはコレ**

TiDBはスケールアウト

TiKVはスケールアップ

オペレーション(TiDB/Aurora)比較

項目	TiDB	Aurora
	評価	評価
バージョンアップ・パッチ適用	○	△
バージョンアップ作業担当者	△	○
Primaryのスケールアップ・ダウン	○	△
Primaryのスケールアウト・イン	○	×
Replicaのスケールアップ・ダウン	○	△
Replicaのスケールアウト・イン	○	△
Primaryのオートスケーリング	×	×
Replicaのオートスケーリング	×	○
Terraformの運用	△	○

AuroraのPrimary・Replicaは、TiDBではTiDBノードのこと

評価ポイント

バージョンアップ、パッチ適用 スケールアップ関連

- 停止メンテ不要：○
- 停止メンテ必要：△
- 対応不可　　：×

バージョンアップ作業

- 自分たちで実施可能：○
- 業者が作業を実施　：△
- 対応不可　　：×

Terraformの運用

- 十分な機能がある　：○
- 機能が不足している：△
- 対応不可　　：×

オートスケーリング

- 対応可能：○
- 対応不可：△

オペレーション考察

注意している点

1

製品としてオンライン変更サポートしているが、状況（TiDB/TiKVの負荷が高い時など）により、クラスターの構成変更はリスクを伴う可能性がある。

○CAPCOM共通基盤としては停止メンテナンスが必用な場合があると考えている
○負荷が高くなる前に対応をするのがベスト

○タイトル側のリリースやメンテナンスの計画を把握

○監視を強化し、すぐに対応できる体制づくり

2

バージョンアップはPingCAP社のサポートに依頼する必要がある

○早めにPingCAP社のサポートと日程調整

費用関連

コストの最適化と削減を期待するが、大幅なコスト増加をしなければ問題なし

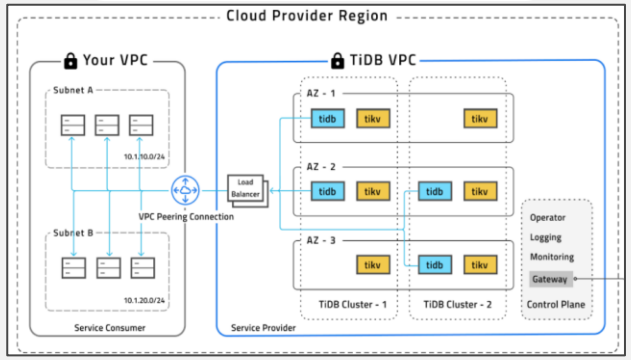
TiDBのポートフォリオ

OSS版、Cloud版など様々な選択肢を用意。
TiDB Cloudにおいては、従量課金モデルのServerlessなども利用可能

Fully managed

パターン 1

TiDB Cloud
(Serverless/Dedicated)

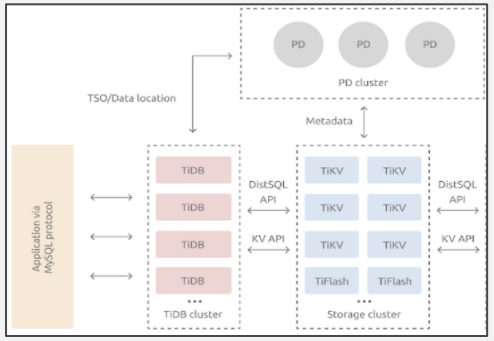


- Webコンソールからデプロイ
- Webコンソールから操作
- 弊社管理VPCへ接続して利用

Self-Hosted

パターン 2

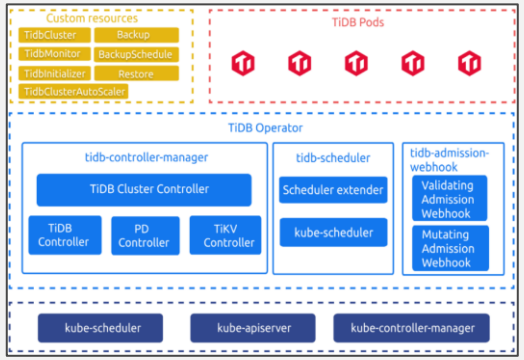
オンプレ/EC2



- Linuxにインストールしてデプロイ
- tiupコマンドを通じて操作
  ※クラスター作成/スケール etc...

パターン 3

Kubernetes
(EKS等含む)



- コンテナとしてデプロイ
- tidb Operatorを通じて操作
  ※クラスター作成/スケール etc...

本番環境以外の検討結果

構成の検討（当初想定）

開発・QA・セキュリティ診断

db.t3.medium（レプリカ無し）
¥19,000/台　×　6

→ **Serverless**
¥9,000/台　×　6

リプレースすると
￥136,000増加

プラットフォーム審査

db.t3.medium（レプリカ有り）
¥19,000/台　×　2

→ **Serverless**
¥9,000/台　×　2

採用してませんが・・・
Dedicated1クラスターでの
集約もコスト削減としてはあり

ステージング環境

db.t3.medium（レプリカ有り）
¥19,000/台　×　2

→ **Dedicated**
TiDB 4Core 16GiB×2
TiKV 4Core 16GiB×3
￥330,000

※注意　コストは2024年4月にAWS社/PingCAP社のサイトの定価および当時の為替を元に作成したものを掲載。
Serverlessの無償枠は費用に含んでおりません。

実際の費用

- Serverlessは5台まで無償枠あり
- 従量課金で実際は当初想定より安価な費用

→ **当初想定より差額が小さい**

本番環境の費用

構成の検討

Auroraを採用しているときは、スペック変更に伴うメンテナンスを回避するために1つ上のスペックを採用。TiDBにすることでスペック変更が容易なったため最適化でき、コスト削減が可能となった。

実際

〇リプレース直後の比較（2024年9月）

マイクロサービス１　リプレース直後　大体同じ金額
マイクロサービス２　リプレース直後　46%コストカット

〇2025年6月（新タイトルがリリースしたため、負荷増加し現在の状況）

マイクロサービス１　リプレース直前と比較　60%コスト増（負荷上昇による構成拡張）
マイクロサービス２　リプレース直前と比較　46%コストカット

全環境のトータル費用

費用については、コストの最適化・削減ができた

8. まとめ


CEDEC
Computer Entertainment Developers Conference
2025

よかったこと

1

メンテナンスを意識せず作業ができるようになった。

停止メンテナンスの調整が不要となり、工数削減

★2024年9月～2025年6月までの実績（本番のみ）

- TiDBの構成変更　（※1クラスター1回と計算）
- 拡張4回／縮退4回
- アップグレード　v8.1.0　→　v8.1.2　計：2回

2

負荷がかかるクエリかどうかでPrimary/Replicaを切り替えるようなシーンがあったところ、TiDBでは何も考えず一本化できるので楽になった。

苦労したこと

TiDBノード数が多い時のTiDBノードのスペック変更に時間がかかる

理由

ローリングで1台ずつ変更を行うため、時間がかかる。
負荷試験などでローリングアップデートしないオプションが欲しい。

回避策

ノードを1台にしてから実施することで回避
ただし、負荷試験環境など自由にノードを変更できる環境のみ可能。
　　→キャンセルができない　3～4時間待機が必要

リプレースについて

1. MySQLでAUTO_INCREMENTを採用していてもリプレースが実現可能。
TiDBのAUTO_INCREMENTの選択によって、注意点が変わるので検証はしっかり行ったほうがよい。
★新規開発するのであれば、アプリ内に別の仕組みの導入が良いと思う

   注意
   Auto Randomはランダムながらもノード数分のランダムベース数値に対してそれぞれインクリメントしているため、Randomにならないことも

2. TiDBのメリットのオンラインでの作業について、一度テストを行いTiDBのノードへの接続の確認などはしたほうが良い

3. データの移行を含めたリハーサルはしっかり行ったほうが良い。

改善できてたら嬉しいこと

1. DedicatedとServerlessとの間の製品が欲しい(金額、バージョン管理)
   ステージング環境の改善
2. Terraform・APIがなかなか癖があるので、できる幅を増やしてほしい。
   #APIキーにきめ細やかな権限設定をしたい
   #Terraform・APIだとServerlessが5台までしか作れない（CLIは可能）
   #Terraformだとスペック変更できない、台数変更はOK
   →一部GUI作業が残ってしまう。
3. 監視の3rdParty（New Relic、DataDogなど）連携部分の強化
   CPUデータのバースト(スパイク)の改善やメトリクスの追加
4. ServerlessにもDB Audit Loggingが欲しい
5. コンソールのAuditログの自動アップロード機能が欲しい

Appendix 検証項目

項目	内容	評価方法
APPのTiDB最適化	ユニットテストでの確認	エラーがない
基礎検証	● クラスター構成別パフォーマンス検証 ● パフォーマンの劣化タイミング確認 ● 長時間の負荷試験	● クラスター構成別のパフォーマンス確認 ● TiDB/TiKVの利用率とパフォーマンスの関係調査 ● メモリリークや時間経過における性能劣化がない
製品比較検証	Aurora/TiDBのコストが近い構成で性能比較 ● Aurora r5.4xlarge ● TiDB 8core/16GiB-3～5ノード TiKV 8core/32GiB-3ノード	1. 性能比較 • DB：QPS、レイテンシー • APP：RPS、レスポンスタイム 2. 費用対性能評価
機能検証	操作試験：負荷がかかっている状態※で以下実施 ① TiDB/TiKVスペック変更 ② TiDB台数拡張・縮退 ③ TiKV台数拡張・縮退 ④ バージョンアップ ※最大パフォーマンスの半分で実施	1. 挙動の確認および許容範囲評価 • アプリケーションの正常性およびエラーの状況 2. 実行時の負荷の確認・評価 • 負荷分散状態の確認 • TiKVのストレージリバランス確認
データ移行試験	AuroraからTiDBへのデータ移行	Dumplingでデータを取得し、 TiDB Cloudのimport機能の正常確認

Appendix 検証時のクラスター構成（1/2）

試験概要	試験項目	TiDBクラスター構成			
		TiDBノード		TiKVノード	
Aurora比較	• 対Auroraと比較の検証構成 • TiDBとTiKVのCPU利用率とパフォーマンスの関係 • TiDBとTiKVのCPU利用率がどの時点で性能が劣化するか • TiKVのCPU利用率に余裕がある状態でTiDBノードを追加すると性能が向上するか • TiKVのCPU利用率に余裕がない状態でTiDBノードを追加すると性能は向上するか ※2つのマイクロサービスで試験	8Core 16GiB	2台	8Core 32GiB	3台
		8Core 16GiB	3台		
		8Core 16GiB	4台		
		8Core 16GiB	5台		
クラスター構成別 TiKVクラスター スケール性能比較	TiKVのクラスター（3台1セット）を増設することで性能はリニアに増加するのか ※1つのマイクロサービスで試験	8Core 16GiB	5台	8Core 32GiB	3台
		8Core 16GiB	10台	8Core 32GiB	6台
		8Core 16GiB	20台	8Core 32GiB	12台
クラスター構成別 ハイスペック性能検証	ハイスペック構成でどこまで性能がでるか ※1つのマイクロサービスで試験	8Core 16GiB	12台	16Core 64GiB	12台
		8Core 16GiB	16台		
		8Core 16GiB	20台		

Appendix 検証時のクラスター構成（2/2）

試験概要	試験項目	TiDBクラスター構成			
		TiDBノード		TiKVノード	
TiKV8Coreと16Core性能比較	8CoreのTiKVと16CoreのTiKVの性能比較 （CPUコア数を同じにすると同じ性能がでるのか） ※1つのマイクロサービスで試験	8Core 16GiB	10台	**8Core 32GiB**	12台
		8Core 16GiB	12台		
		8Core 16GiB	14台		
		8Core 16GiB	20台		
		8Core 16GiB	10台	**16Core 64GiB**	6台
		8Core 16GiB	12台		
		8Core 16GiB	14台		
		8Core 16GiB	16台		
		8Core 16GiB	18台		
		8Core 16GiB	20台		