Amazon RDS Proxy のご紹介

2020/7/28

アマゾン ウェブ サービス ジャパン株式会社
データベース スペシャリスト ソリューション アーキテクト

内容についての注意点

本資料では2020年7月1日時点のサービス内容および価格についてご説明しています。
最新の情報はAWS公式ウェブサイト (http://aws.amazon.com/) にてご確認ください。

- 資料作成には十分注意しておりますが、資料内の価格とAWS公式ウェブサイト記載の価格に相違があった場合、AWS公式ウェブサイトの価格を優先とさせていただきます
- 価格は税抜表記となっています。日本居住者のお客様がご利用される場合、別途消費税をご請求させていただきます

AWS does not offer binding price quotes. AWS pricing is publicly available and is subject to change in accordance with the AWS Customer Agreement available at http://aws.amazon.com/agreement/. Any pricing information included in this document is provided only as an estimate of usage charges for AWS services based on certain information that you have provided. Monthly charges will be based on your actual use of AWS services, and may vary from the estimates provided.

アジェンダ

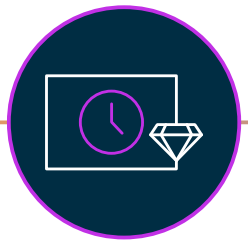
- RDS Proxy 全体像
- 接続プーリング
- シームレスフェイルオーバー
- アプリケーション セキュリティの向上
- 監視 / 考慮事項など

Amazon Relational Database Service (RDS)

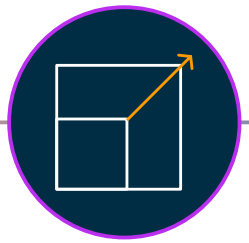
人気のある6つのデータベースエンジンの選択による管理されたリレーショナルデータベースサービス

Amazon Aurora　MySQL　PostgreSQL　MariaDB　Microsoft SQL Server　ORACLE

インフラストラクチャのプロビジョニング、データベースのインストール、メンテナンスは不要



マルチAZデータレプリケーション、自動バックアップ、スナップショット、自動フェイルオーバー



数クリックでデータベースのコンピュートとストレージを拡張可能; アプリケーションのダウンタイムは最小限

SSDストレージと保証されたプロビジョンドI/O; 保存時、転送中のデータ暗号化

モダンアプリケーション対応のチャレンジ

サーバーレスアプリケーションなどのモダンアプリケーションは大量のDB接続、短期間での接続の開閉リクエストを要求する場合があり、データベースも対応が求められている

数千レベルのDB接続が張られる事もあり、DBリソースを大量消費してしまう

セキュアなアプリケーションコードの開発
(ユーザー名/パスワードの埋め込み等への対応)

セルフマネージド型のプロキシサーバーは実装、運用にコストがかかる

Amazon RDS Proxy

Amazon RDS 向けの高可用性フルマネージド型データベースプロキシ
アプリケーションのスケーラビリティやデータベース障害に対する回復力と
安全性の向上を実現

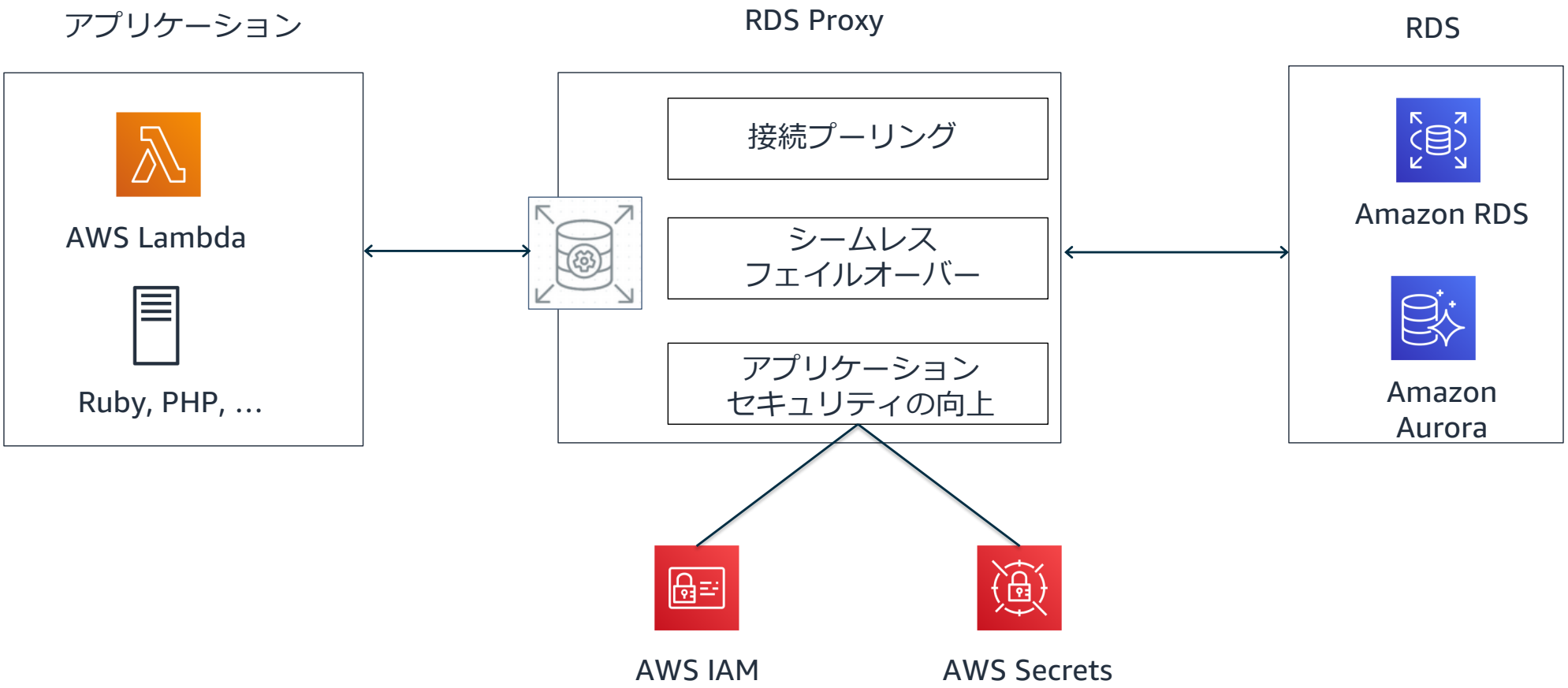
データベース接続をプールおよび共有する事でアプリケーションのスケーリングを改善

アプリケーションの可用性を高め、データベースのフェイルオーバー時間を短縮

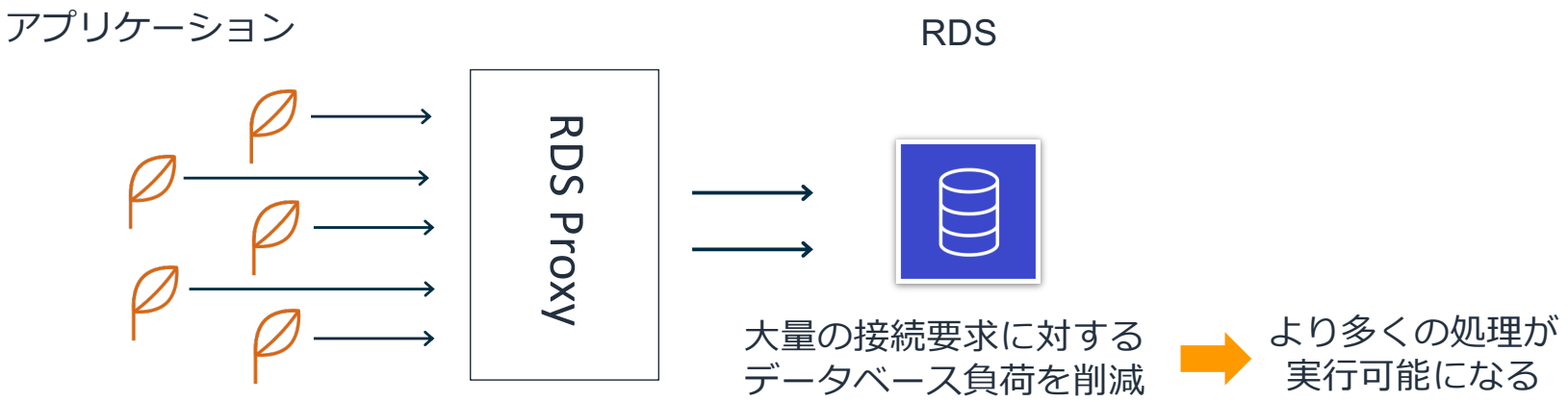
データベースアクセス制御で、アプリケーションデータのセキュリティを管理

フルマネージドデータベースプロキシ、データベースとの完全な互換性

Amazon RDS Proxy 全体像



接続プーリング



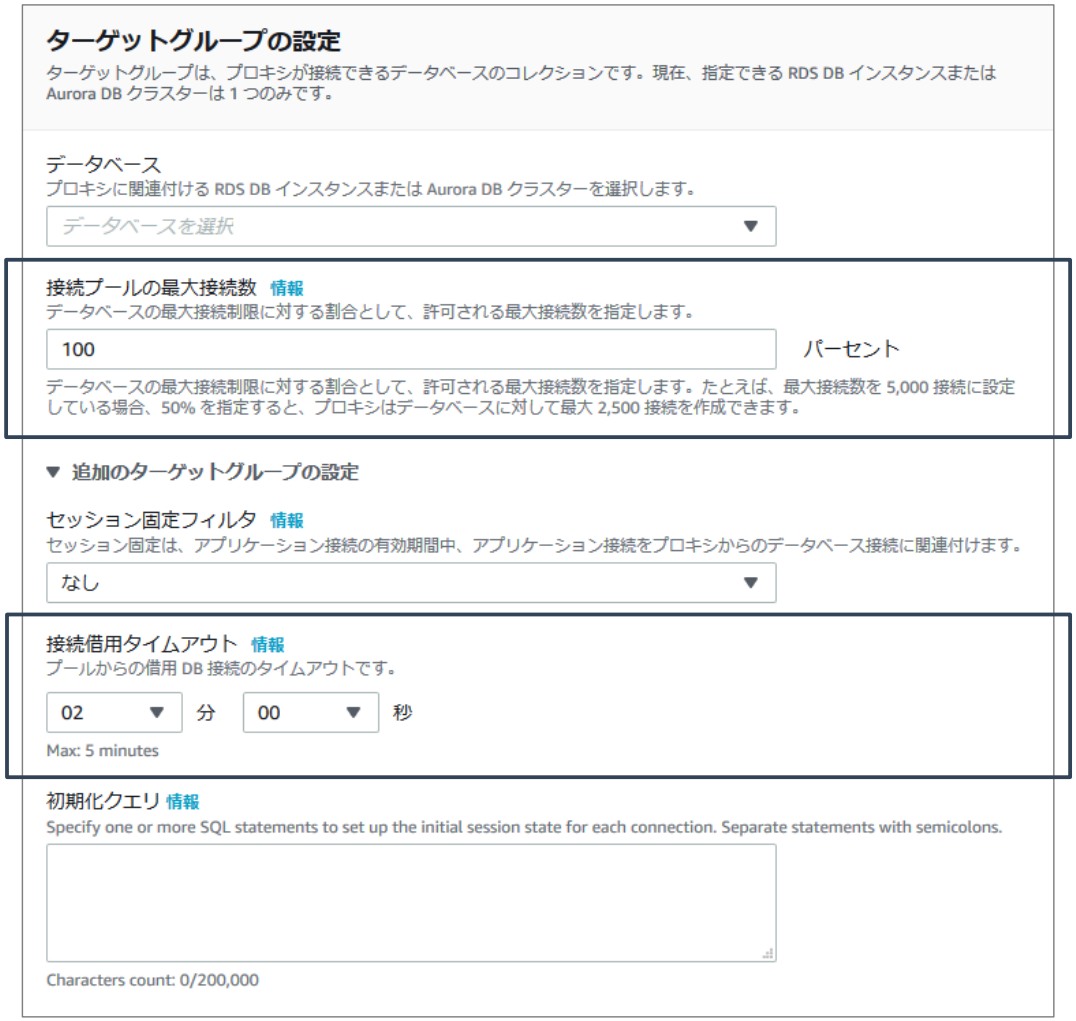
接続プーリング

- 接続の開閉に伴うデータベースの負荷 (TLS/SSL のハンドシェイク、認証、ネゴシエーション機能などのCPU負荷など) を削減

接続の多重化

- 接続の再利用により、データベース接続に必要なコンピューティングリソース (主にメモリ) を削減
- max_connections エラーの発生頻度の抑制。

設定イメージ



- max_connections に対する割合 (%)
  - 1～100 の値を指定
- 1つのデータベースに複数のプロキシを割り当てる事が可能
  - 同じデータベースに割り当てるプロキシの割合の合計を100%以下にする事

- Connection borrow timeout
- アプリケーションからの接続要求に対して接続プールが割当てられない場合に待機する時間
  - 接続数が最大値に達し、接続プールで利用可能な接続がなくなった場合
  - フェイルオーバーオペレーションが進行中であるためにWriter インスタンスが使用できない場合

設定イメージ

- コネクションのアイドル時間
- アイドル状態がこの設定時間を超えると接続はプールに戻される

RDS Proxy と従来のエンドポイント

- RDS Proxy は Writer インスタンスに対する Proxy
  - Reader インスタンスに対する接続は読み取りエンドポイントを利用
  - 従来のクラスターエンドポイント / カスタムエンドポイント / インスタンスエンドポイントも利用可能

aws

接続の多重化における考慮事項 (1)
トランザクション中の接続

1つのトランザクション内のステートメントは同じデータベース接続を使用

- autocommit 無効
  - トランザクションが終了するまで (commit / rollback等) 、接続の再利用は行われない
- autocommit 有効
  - 接続はステートメント終了後、再利用可能な状態になる

DDL ステートメントの動作に注意

- MySQL の場合は DDL ステートメント完了後にトランザクションは終了
- PostgreSQL の場合はトランザクションは終了しない

接続の多重化 (再利用) という観点では、ロングトランザクションより複数のショートトランザクションでの実装が有効

aws

接続の多重化における考慮事項 (2)
ピン留め

- データベース接続を特定のクライアントに占有させ、セッションが終了するまで同じデータベース接続を使い続ける状態
  - 先行するクエリの状態情報に依存すると判断した場合、ピン留めが行われる
  - ピン留めされたセッションは接続の多重化が行われない

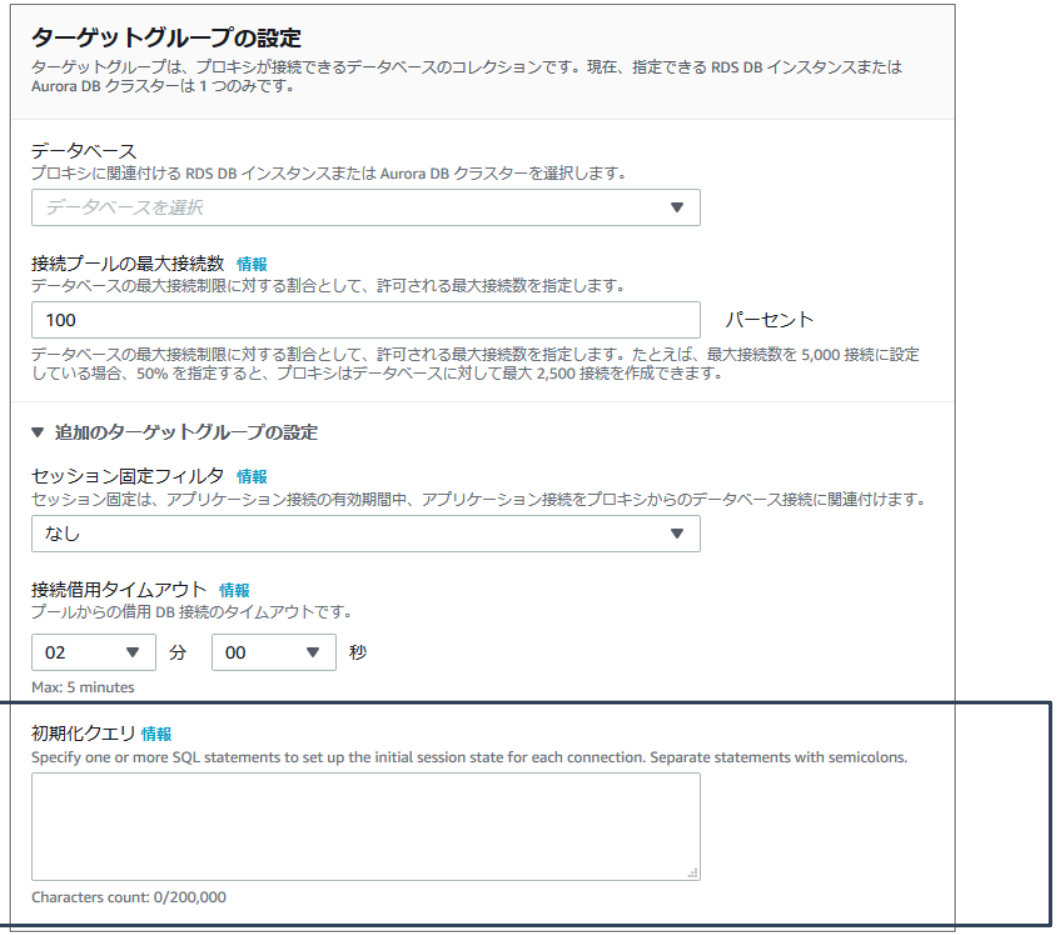
【例】

- SET ステートメントでセッションレベルの変数を変更した場合 (一部例外あり)
  - 全ての接続で共通のSET ステートメントを使う場合は「初期化クエリ」を使う事でピン留めを回避可能 (次ページ)
- 一時テーブルの作成
- プリペアドステートメント (サーバーサイド)
- PostgreSQL でJDBC を利用する際に、デフォルトのpreferQueryMode(=extended)の場合、セッションはピン留めされる。これを回避するにはpreferQueryMode をextendedForPrepared に設定(詳細は下記URL)
- その他細かい条件あり、詳細は下記をご参照下さい

  https://docs.aws.amazon.com/ja_jp/AmazonRDS/latest/UserGuide/rds-proxy.html#rds-proxy-pinning

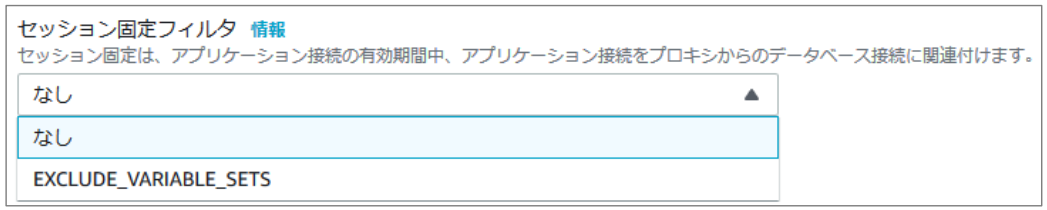
接続の多重化 (再利用) という観点では、不必要なピン留めが行われる処理の回避を検討

初期化クエリ



- 複数のSQLステートメントを記載可能
  (セミコロンで区切る)

(ご参考資料) セッション固定フィルター

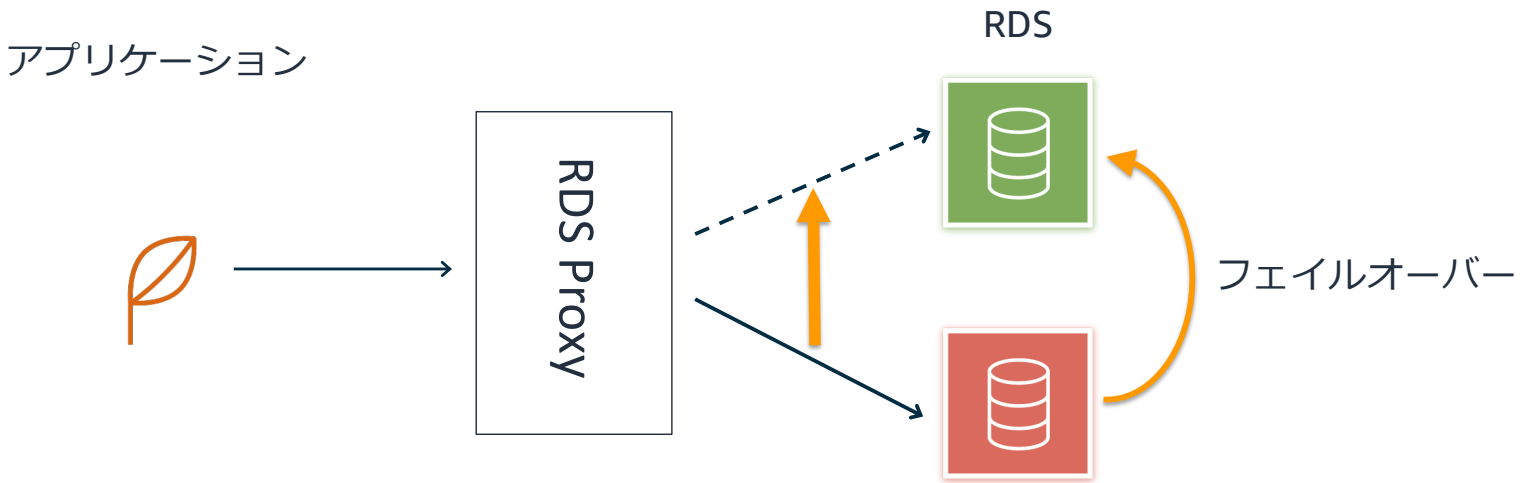


- 特定のアプリケーションのパフォーマンス問題に対してのトラブルシューティング用
- 設定にあたっては、アプリケーションが正常に動作をする事を要確認

設定値

- なし (デフォルト)
- EXCLUDE_VARIABLE_SETS
  - 下記のアプリケーションステートメント についてピン留め操作を行わない
    - セッションの変数と構成設定の設定

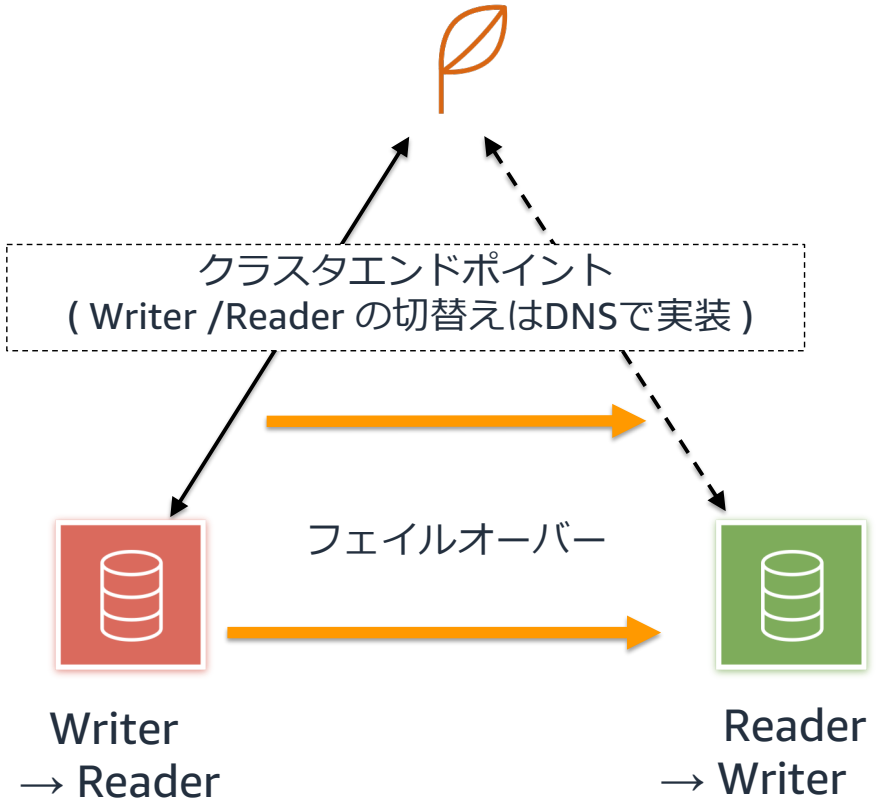
シームレスかつ高速なフェイルオーバー



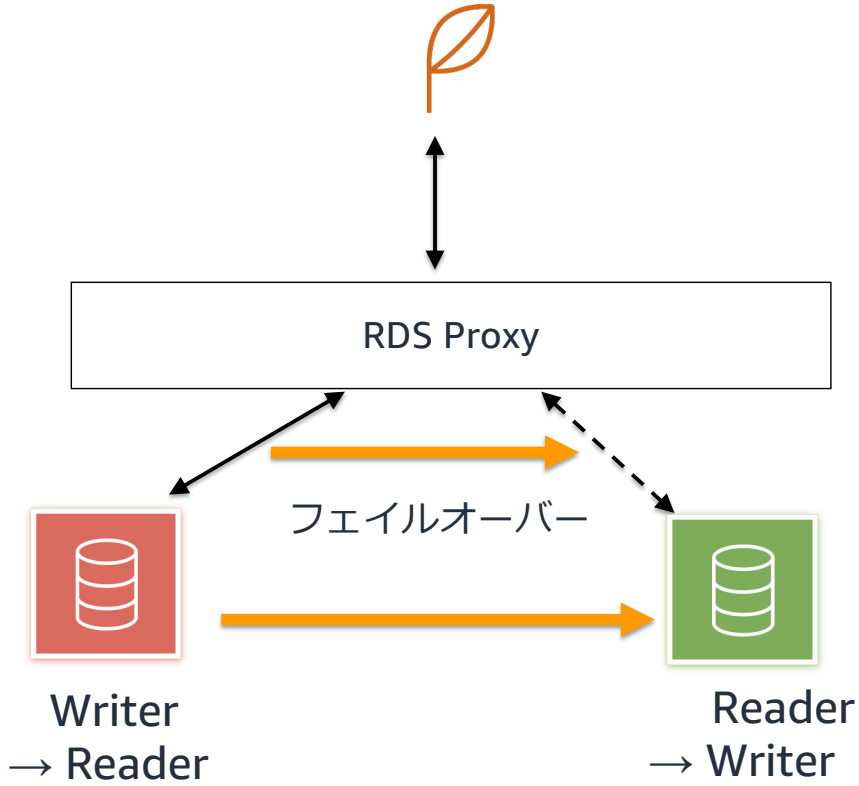
- DNS の変更に依存しない高速なフェイルオーバーを実現
- RDS Proxy がバックエンドデータベースのフェイルオーバーを検出。アイドル状態の接続はアプリケーション－RDS Proxy 間の接続を維持した状態でフェイルオーバー先のデータベースに接続。アプリケーションからの再接続は不要。

フェイルオーバー動作 概念図

【RDS Proxy を利用していない時】

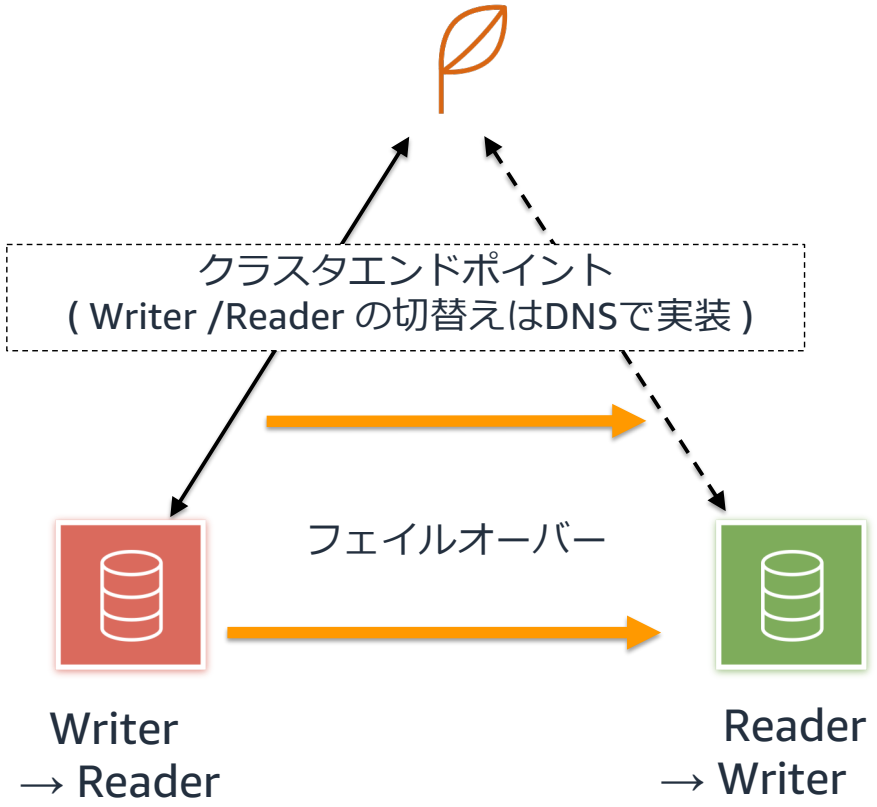


【RDS Proxy 利用時】



フェイルオーバー動作 概念図

【RDS Proxy を利用していない時】



- DNS レコード変更に伴う影響を考慮する必要がある
- DNS キャッシュの考慮が必要
  - キャッシュする時間が長い
    → フェールオーバー先に接続するまでの時間がかかる
  - キャッシュする時間が短い
    → フェイルオーバー後のReaderに接続してしまう
- フェイルオーバー後、既存の接続の再接続が必要

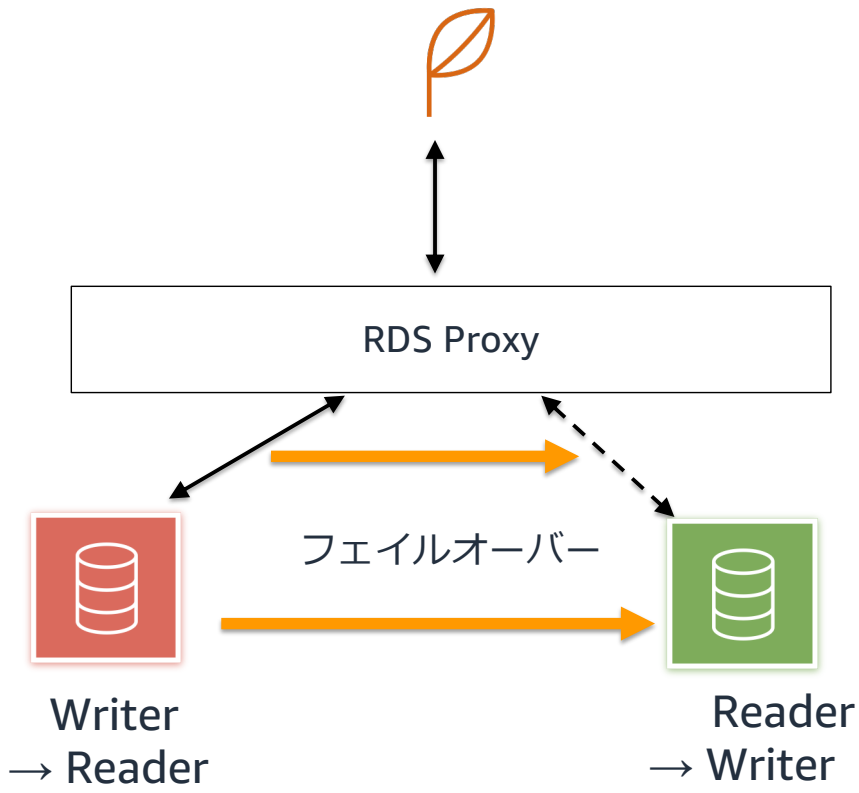
フェイルオーバー動作 概念図

- DNS レコード変更に伴う影響を考慮する必要がない
- DNS キャッシュの考慮不要

→ 高速なフェイルオーバーの実現

- RDS Proxy がバックエンドデータベースの フェイルオーバーを検知
  - 確実にWriter インスタンスに接続
  - アイドル状態の接続はアプリケーション－RDS Proxy間の接続を維持した状態で、フェイルオーバー先のデータベースに接続。アプリケーションの再接続は不要
  - クエリ処理中・トランザクション中等の接続に関しては再接続が必要

【RDS Proxy 利用時】



フェイルオーバー時間 測定結果例

Aurora MySQL

Test	Min (ms)	Max(ms)	Avg(ms)	Proxy Advantage
RDS Proxy with Aurora (Maria Driver)	1,644	11,642	2,913	79%
Direct to Auroa (Maria Aurora Driver)	5,146	30,782	13,783	

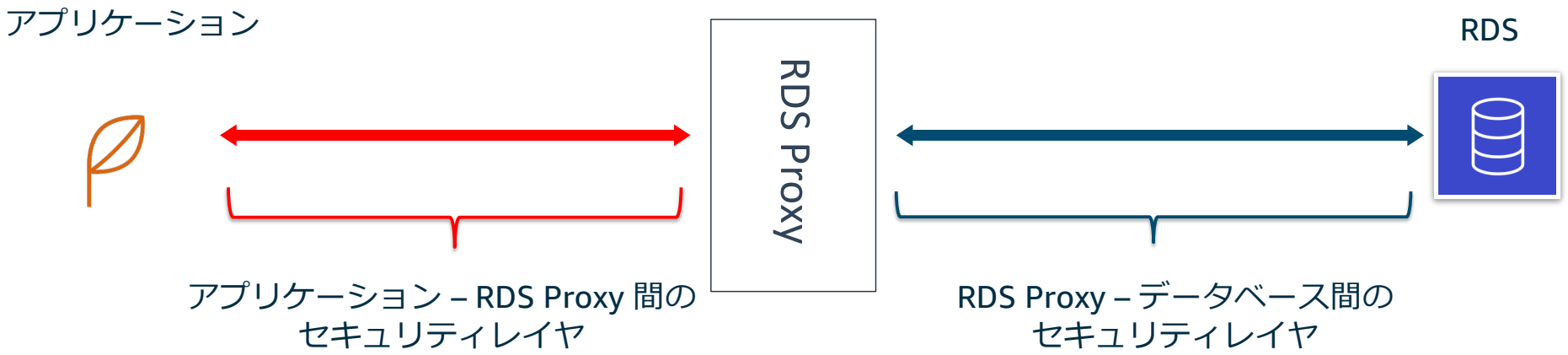
RDS for MySQL

Test	Min(ms)	Max(ms)	Avg(ms)	Proxy Advantage
RDS Proxy Multi-AZ (Maria Driver)	21,485	29,176	25,075	32%
Direct to Multi-AZ (Maria Driver)	27,240	52,234	36,849	

RDS Proxy を利用する事で Aurora MySQL の場合は 79%
RDS for MySQL の場合は 32% のフェイルオーバー時間の短縮

詳細は下記 AWS ブログ 「Amazon RDS Proxy を使用したアプリケーションの可用性の向上」をご参照下さい
https://aws.amazon.com/jp/blogs/news/improving-application-availability-with-amazon-rds-proxy/

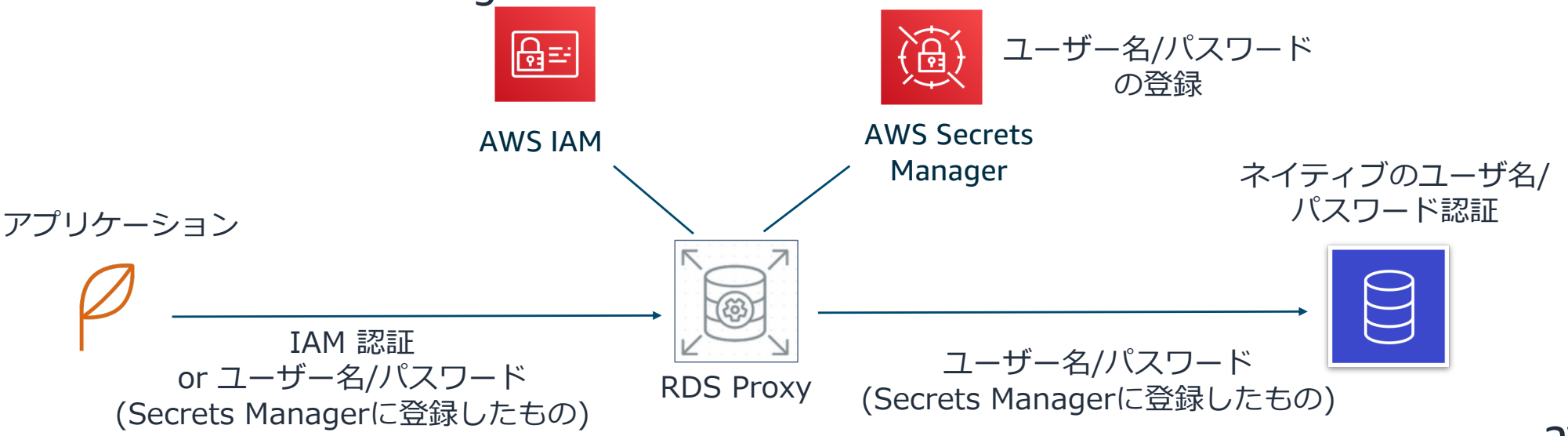
アプリケーションセキュリティの向上



- データベースサーバーがネイティブのユーザー名 / パスワード認証しか対応していない場合でも、RDS Proxy への接続はIAM認証を利用可能。アプリケーションコードに認証情報を埋め込まない事によるセキュリティ向上を実現
- データベースサーバーが TLS1.0 / 1.1 のみをサポートしている場合でも、RDS Proxy への接続は TLS 1.2 を利用可能

アプリケーションセキュリティの向上 (1)

- データベースサーバーがネイティブのユーザー名 / パスワード認証しか対応していない場合でも、RDS Proxy への接続は IAM 認証を利用可能。アプリケーションコードに認証情報を埋め込まない事によるセキュリティ向上を実現
- AWS Secrets Manager によるデータベース認証情報の一元管理



設定イメージ
Secrets Manager / IAM ロール

- Secrets Manager を設定
- 各 RDS Proxyに対し、最大 200個の Secrets Manager を設定可能

- IAM ロールを設定

- 必須: IAM 認証のみを許可
- 無効: DB 認証情報のみを許可 (Secrets Manager に登録した DB認証情報)

設定イメージ
Secrets Manager の作成

ユーザー名 / パスワードを登録

接続先のRDSを設定

認証情報を埋め込まないアプリケーション
コードサンプル

```
・・・
ENDPOINT="postgresmydb.123456789012.us-east-1.rds.amazonaws.com"
PORT="5432"
USR="jane_doe"
REGION="us-east-1"
DBNAME="postgres"

#gets the credentials from .aws/credentials
session = boto3.Session(profile_name='RDSCreds')
client = boto3.client('rds')
```

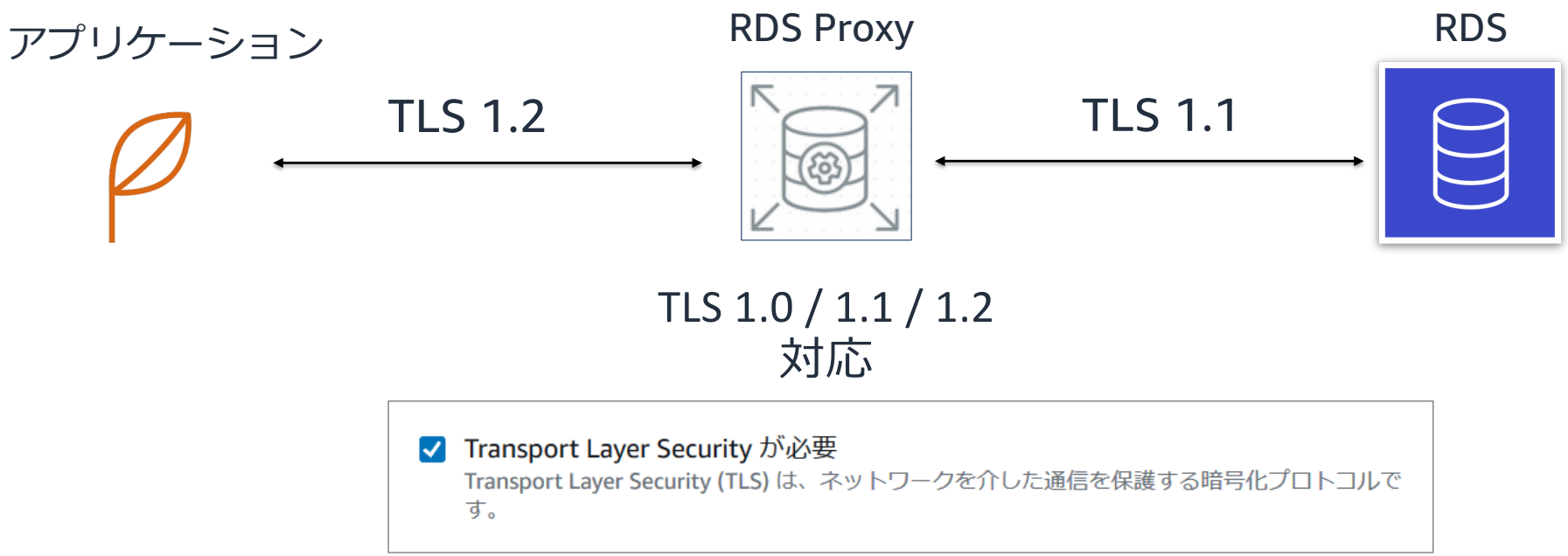
IAM認証トークンの生成

```
token = client.generate_db_auth_token(DBHostname=ENDPOINT, Port=PORT, DBUsername=USR, Region=REGION)

try:
    conn = psycopg2.connect(host=ENDPOINT, port=PORT, database=DBNAME, user=USR, password=token)
    cur = conn.cursor()
・・・
```

アプリレーションセキュリティの向上 (2)

データベースサーバーが TLS1.0 / 1.1 しか対応していない場合でも、RDS Proxy とApplication間は TLS1.2 を利用する事が可能



監視

CloudWatch メトリクス
(代表的なメトリクス)

メトリクス	説明
ClientConnections	現在のクライアント接続の数。
DatabaseConnections	現在のデータベース接続の数
DatabaseConnectionsCurrentlySessionPinned	クライアントリクエストにおけるセッション状態変更オペレーションのために現在ピン留めされているデータベース接続の数。
ClientConnectionsReceived	受信したクライアント接続リクエストの数
DatabaseConnectionsCurrentlyInTransaction	トランザクションでのデータベース接続の現在の数。
・・・	・・・

その他のメトリクスに関して下記をご参照下さい

https://docs.aws.amazon.com/ja_jp/AmazonRDS/latest/AuroraUserGuide/rds-proxy.html#rds-proxy.monitoring

ユースケース

ユースケース	特性	RDS Proxy
利用ユーザー数等の増加が予測不可能なアプリケーション	大量で急激なデータ接続要求	• データベース接続の開閉に伴うデータベースリソース使用の削減 (主にCPU) • データベース接続を共有する事で、データベースが接続に使用するデータベースリソース 使用の削減 (主にメモリ) • データベース接続の数を制御し予測可能なデータベースパフォーマンスを維持
データベース接続を頻繁に開閉するアプリケーション	サーバーレス、PHP、Ruby on Rails などのアプリケーションはデータベース接続の頻繁な開閉を行う場合がある	
SaaS や eコマースなどのアプリケーション	データベース接続をアイドル状態に維持して応答時間のパフォーマンスを図る事をしているケースがある	• データベース接続を共有する事で、データベースが接続に使用するデータベースリソース 使用の削減
可用性を求められるシステム	フェイルオーバー時のシステム停止を極力少なくしたい	• フェイルオーバー時間の削減
アプリケーションセキュリティが重要なシステム	TLS1.2 を利用したい アプリケーションに認証情報を埋め込みたくない	• アプリケーションとRDS Proxy間を IAM認証や TLS1.2 を利用する事でセキュリティの向上

RDS Proxy サポートリージョン と サポートエンジン

サポートリージョン

- アジアパシフィック(東京) リージョン 利用可能
- その他利用可能リージョンに関しては下記をご参照下さい。
  - https://docs.aws.amazon.com/ja_jp/AmazonRDS/latest/AuroraUserGuide/rds-proxy.html

サポートエンジン

- MySQL
  - RDS for MySQL 5.6 / 5.7
    Aurora MySQL バージョン1.x / 2.x
- PostgreSQL
  - RDS for PostgreSQL / Aurora PostgreSQL 10.10 以降 、 11.5 以降

aws

注意事項

- Aurora クラスターにおいて、RDS Proxy が対応しているのはWriter インスタンスのみ
  - 読み取りのワークロードの負荷分散には従来のReaderエンドポイントを利用
- Aurora サーバーレスクラスター、マルチマスタークラスターにおいては RDS Proxy は利用不可
- RDS Proxy はデータベースと同じVPCに存在する必要があり、パブリックアクセス不可 (データベースのパブリックアクセスは可能)

- その他の制約事項等に関しては下記をご参照下さい
  https://docs.aws.amazon.com/ja_jp/AmazonRDS/latest/AuroraUserGuide/rds-proxy.html#rds-proxy.limitations

aws

料金の考え方

有効化されている各データベースインスタンスの vCPU毎の1時間あたりの料金
: 0.018USD/時間　(アジアパシフィック(東京) 2020/7/1時点)

シングルAZ インスタンス
- 2 vCPU m5.large インスタンス
- 0.018 USD × 2 vCPU × 24時間 ×30日 = 25.92 USD

マルチ AZ インスタンス
- 2 vCPU m5.large インスタンス で マルチAZ構成
- 0.018 USD × 2 vCPU × 24時間 ×30日 = 25.92 USD

Aurora クラスター
- クラスター全体のvCPU数
- 2 vCPU r5.large インスタンス (Writer) 、2 vCPU r5.large インタスタンス (Reader)
- 0.018 USD × (2 vCPU + 2 vCPU) × 24時間 ×30日 = 51.84

※) 料金計算に利用する最小 vCPU は 2 vCPU
(対象データベースインスタンスが 1 vCPU の場合は、料金計算に利用する vCPU数 は 2 vCPUとなります)

https://aws.amazon.com/jp/rds/proxy/pricing/

まとめ

- Amazon RDS 向けの高可用性フルマネージド型データベースプロキシ
- 接続プーリングにより、大量の接続要求によるデータベースの負荷を削減
- 高速フェイルオーバーによる可用性向上
- アプリケーションのセキュリティ向上

Thank you!