Amazon Aurora Deep Dive (Part-3)
Amazon Aurora でのパフォーマンスチューニング

程　家（てい　か）

アマゾン ウェブ サービス ジャパン合同会社
シニアソリューション アーキテクト

内容についての注意点

- 本資料では 2022年8月23日時点のサービス内容および価格について説明しています。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.

自己紹介

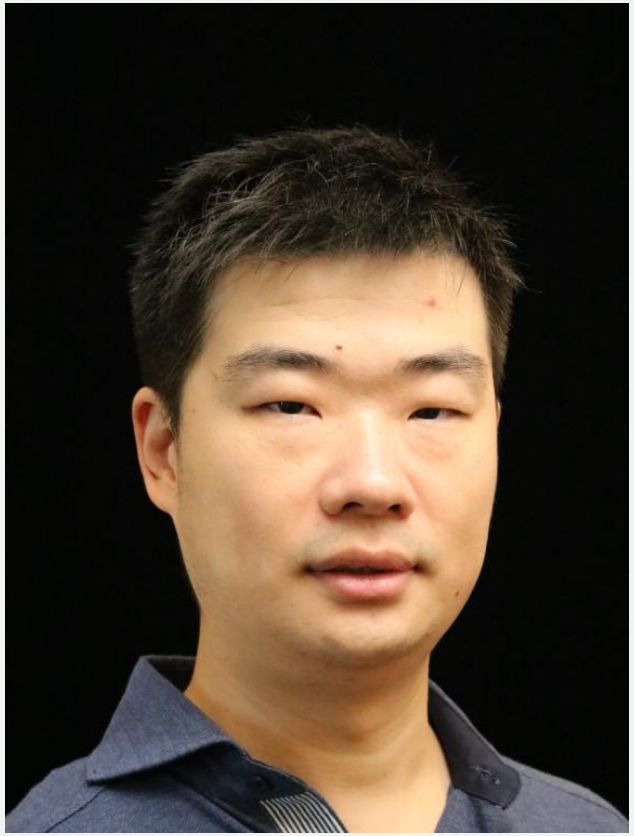
程　家（てい　か）

シニアソリューションアーキテクト

- 外食、サービス業界のお客様を担当

お気に入りのAWSサービス

- Amazon RDS / Amazon Aurora
- Amazon RDS Performance Insights
- Amazon DevOps Guru



アジェンダ

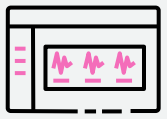
- データベースにおけるパフォーマンス分析の課題
- Amazon RDS Perfomrance Insights の登場
- Amazon DevOps Guru for RDS の概要
- Amazon DevOps Guru for RDS の利用イメージ
- クエリーの実行計画を管理する
- まとめ

データベースにおける
パフォーマンス分析の課題

データベースにおけるパフォーマンス分析の課題

現在、何十万ものお客様が Amazon Aurora を利用していますが、アプリケーションの規模や複雑さが増すにつれて、お客様が運用やパフォーマンスの問題を迅速に検出し、解決することは難しくなってきています。

OS やデータベースの様々はメトリクスの組み合わせを効率的に分析するのは経験豊富なDBAが必要



細かい粒度のパフォーマンスデータを長期間にわたり、収集、管理していくのは運用コストが高い

パフォーマンスの異常検出、通知の仕組みを構築するコストが高い

Amazon RDS Perfomrance Insights の登場

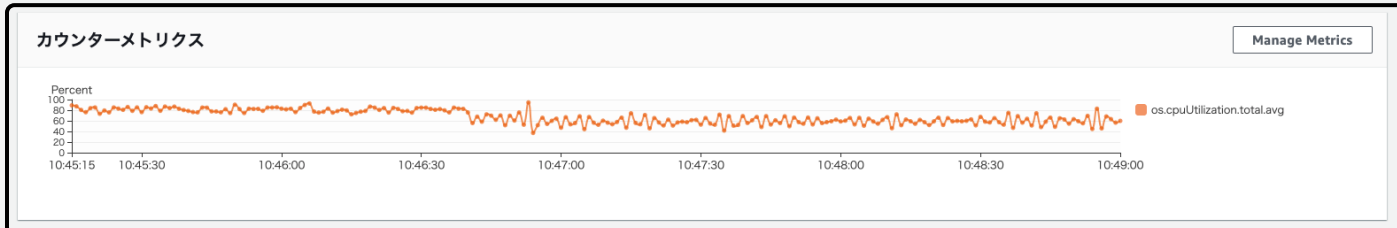
Amazon RDS Performance Insights の特徴

シンプル	簡単	自動	高機能
データベースロード(AAS)を使って、単一のコアKPIでパフォーマンス監視、分析	数クリックでパフォーマンスKPIの取得からダッシュボードへの表示をデプロイ可能	特別な設定やメンテナンスは不要。また最大2年(*1)は問題のあった時点に遡って分析が可能	Amazon RDS / Auaroa がサポートする6つのデータベースエンジン全て(*2)で同じ操作感で利用可能

(*1) 7日分は無料で利用可能。有料で保存期間を 1 か月から 24 か月まで指定可能
(*2) 全てのデータベースエンジンに対応していますが、一部のバージョンやインスタンスタイプではPerformance Insightsはサポートされていません
https://docs.aws.amazon.com/ja_jp/AmazonRDS/latest/UserGuide/USER_PerfInsights.html

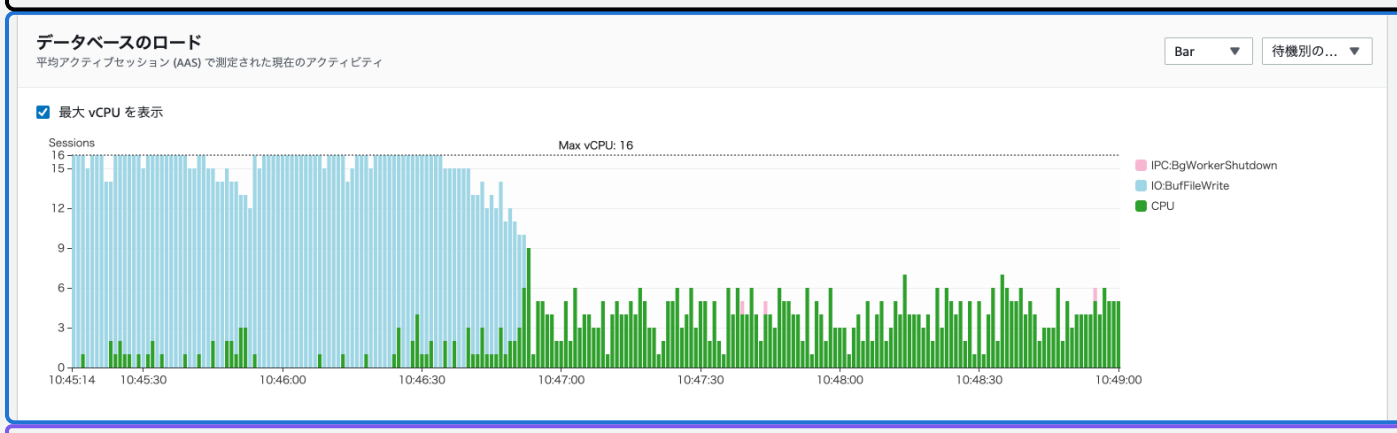
Amazon RDS Performance Insights の利用イメージ

データベース内のパフォーマンスデータを蓄積してボトルネックを特定



カウンターメトリクス

- OSのリソース情報(CPU、Memoryなど)
- DBの統計情報(セッション数など)



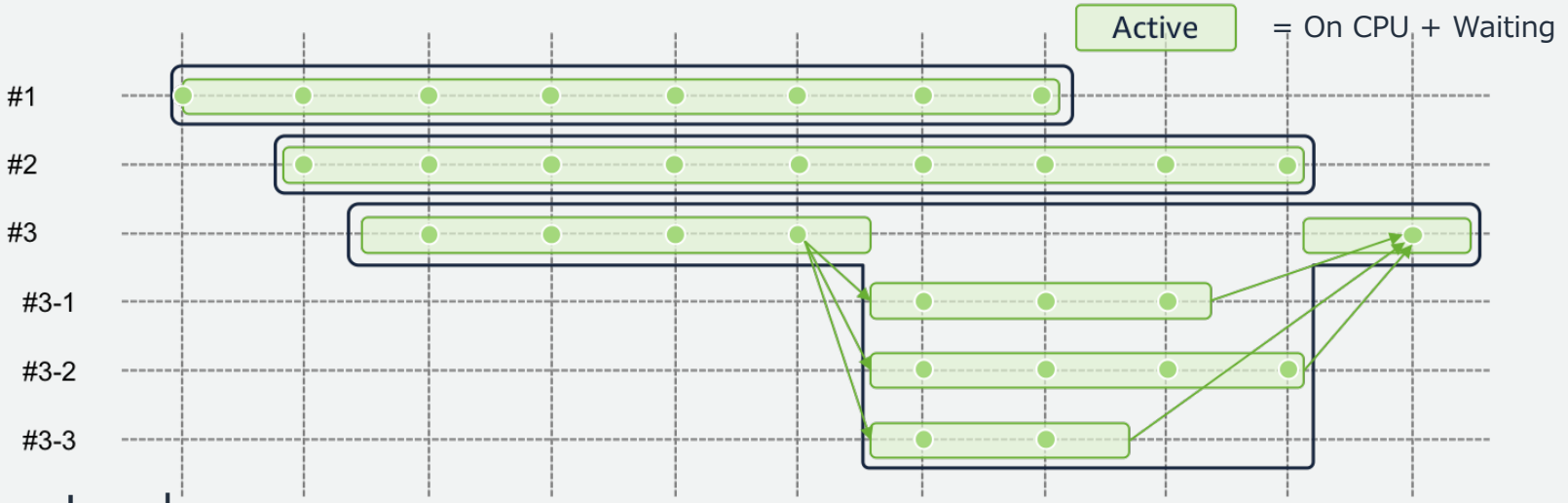
データベースのロード

- 平均アクティブセッション数(AAS)
- CPU時間と待機イベント内訳
- RDS/Aurora全てのエンジンをサポート

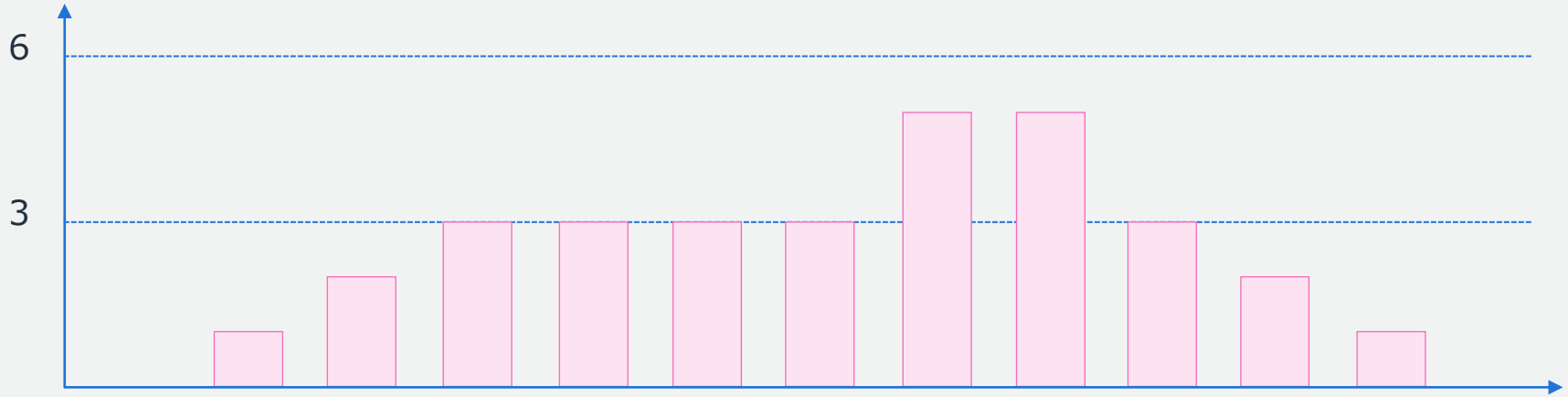
ボトルネックの分析軸

- ボトルネックの原因となっている待機
- ボトルネックとなっているSQL
- 性能影響の高いホスト、ユーザー

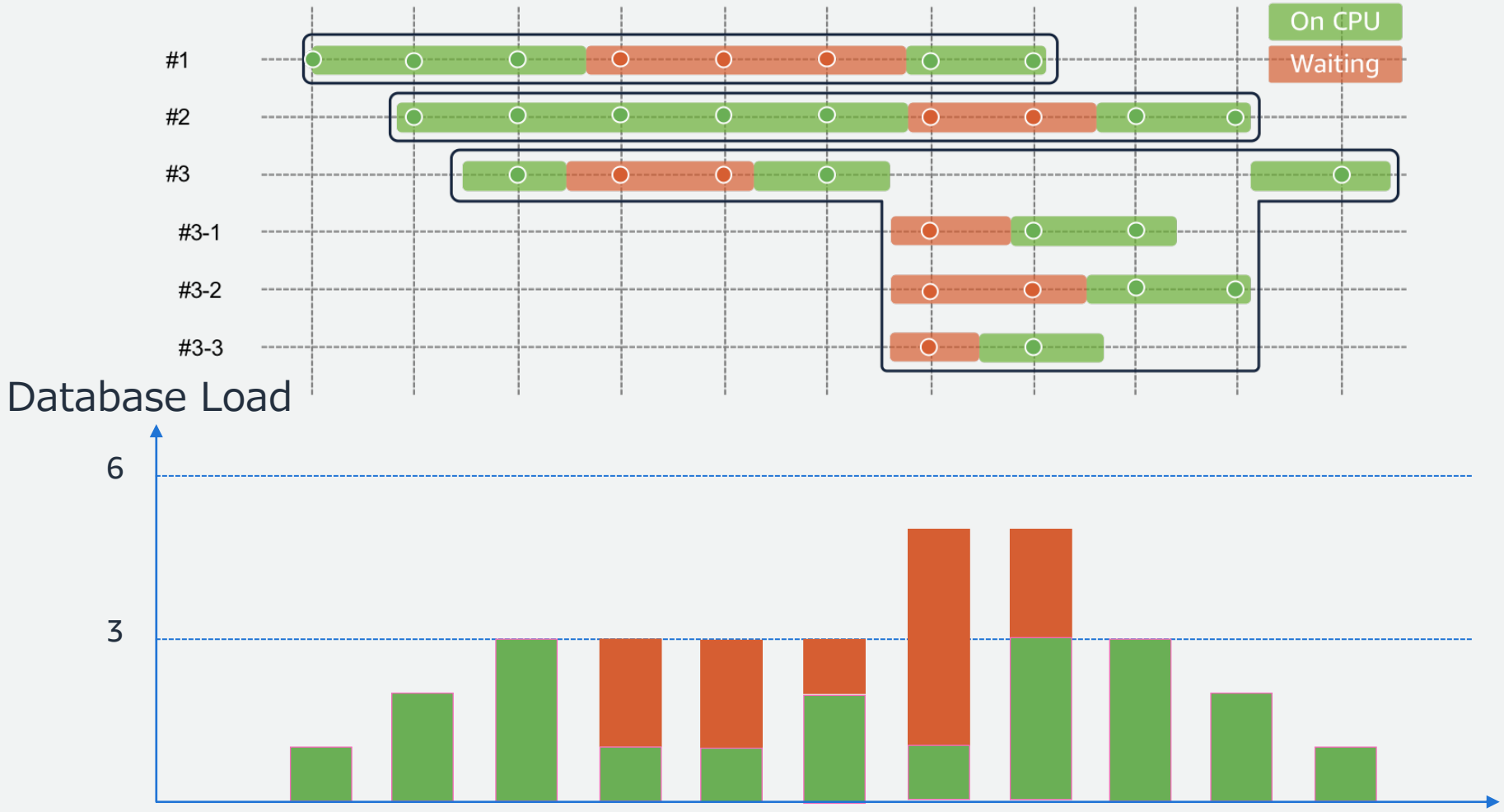
データベースロード (Average Active Sessions)



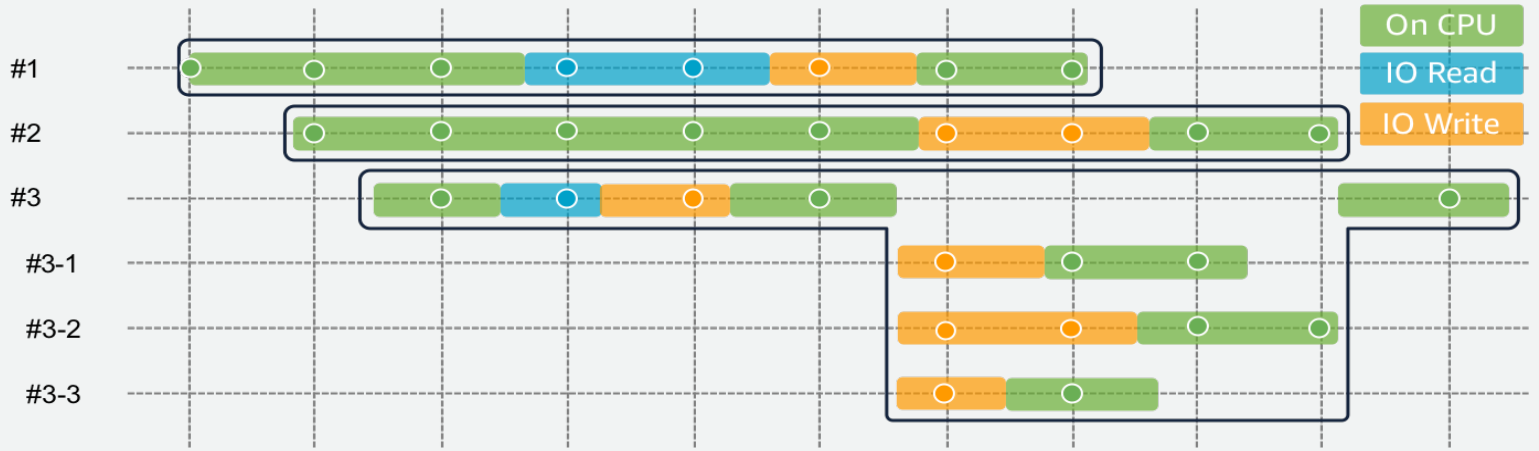
Database Load



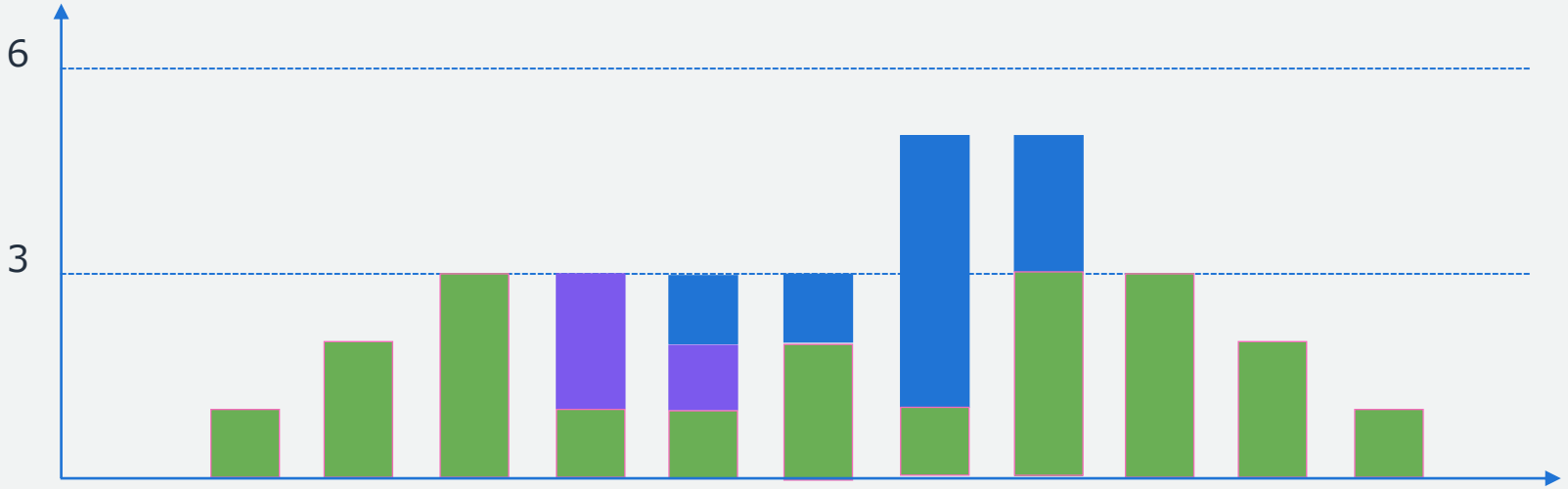
データベースロード (Average Active Sessions)



データベースロード (Average Active Sessions)



Database Load

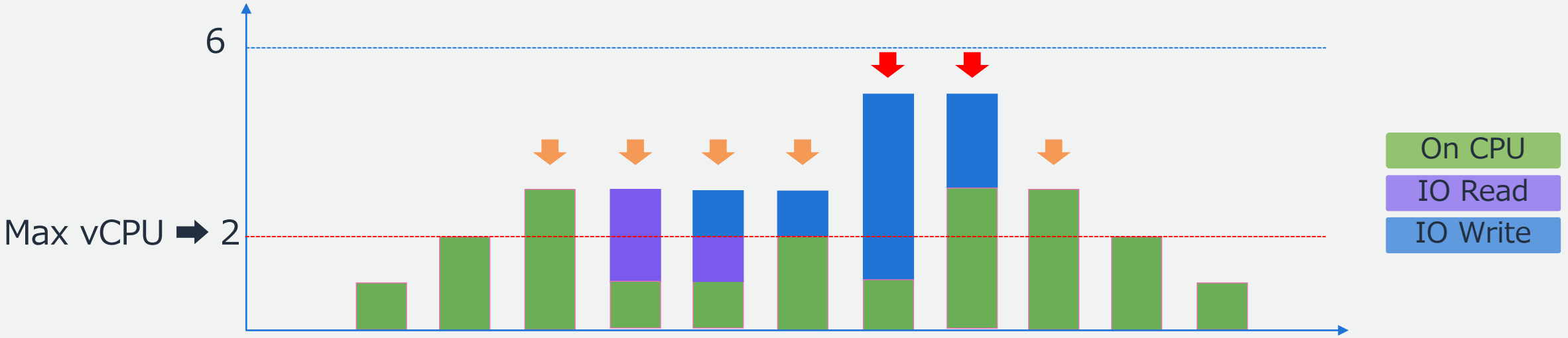


AASを使ったパフォーマンス問題の特定

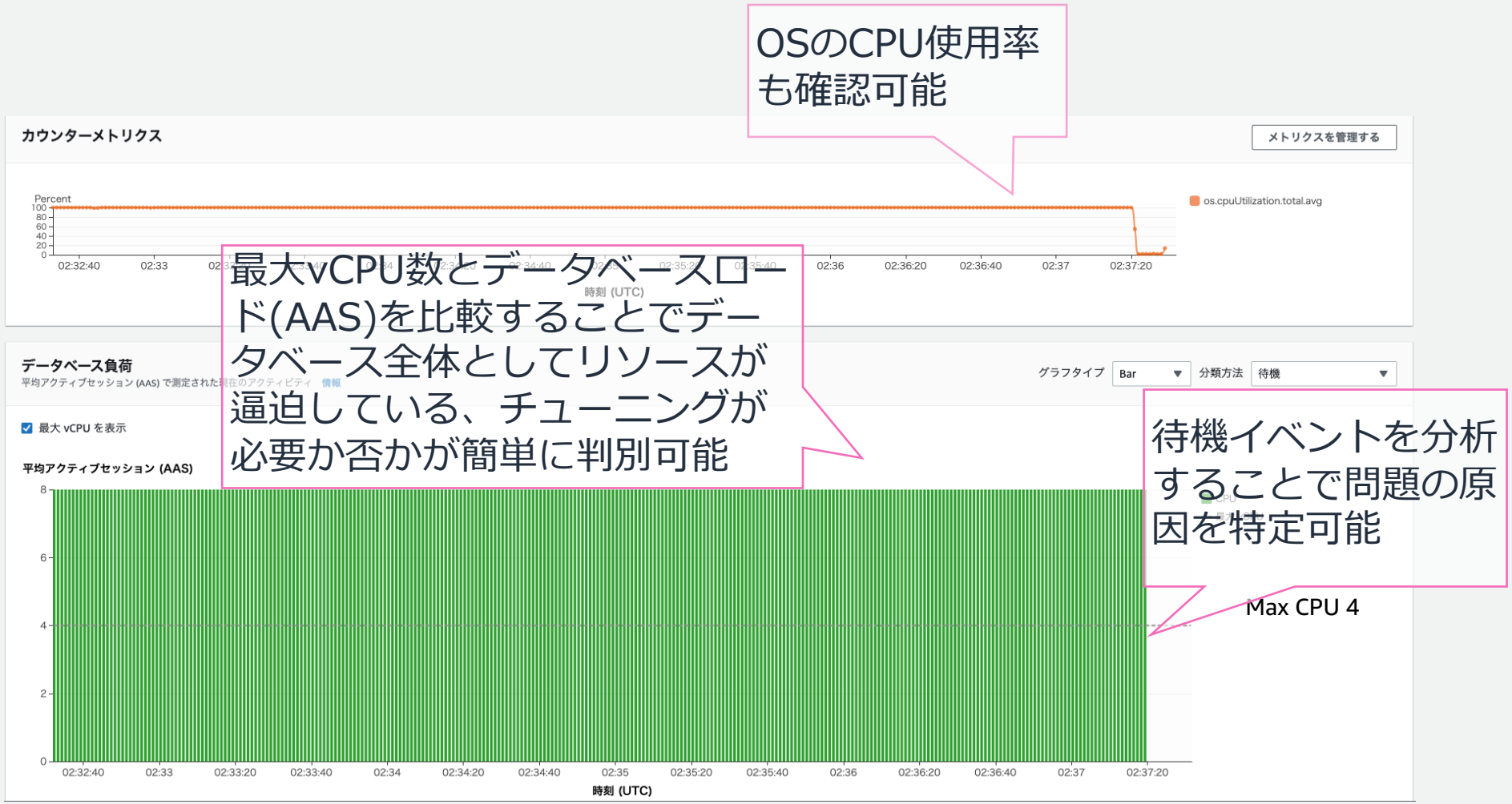
- ✓ Database Load(AAS) ~= 0
  基本的にデータベースが使用されていない

- ✓ Database Load(AAS) > # of vCPUs
  パフォーマンス問題の可能性がある
- ✓ Database Load(AAS) >> # of vCPUs
  パフォーマンス問題

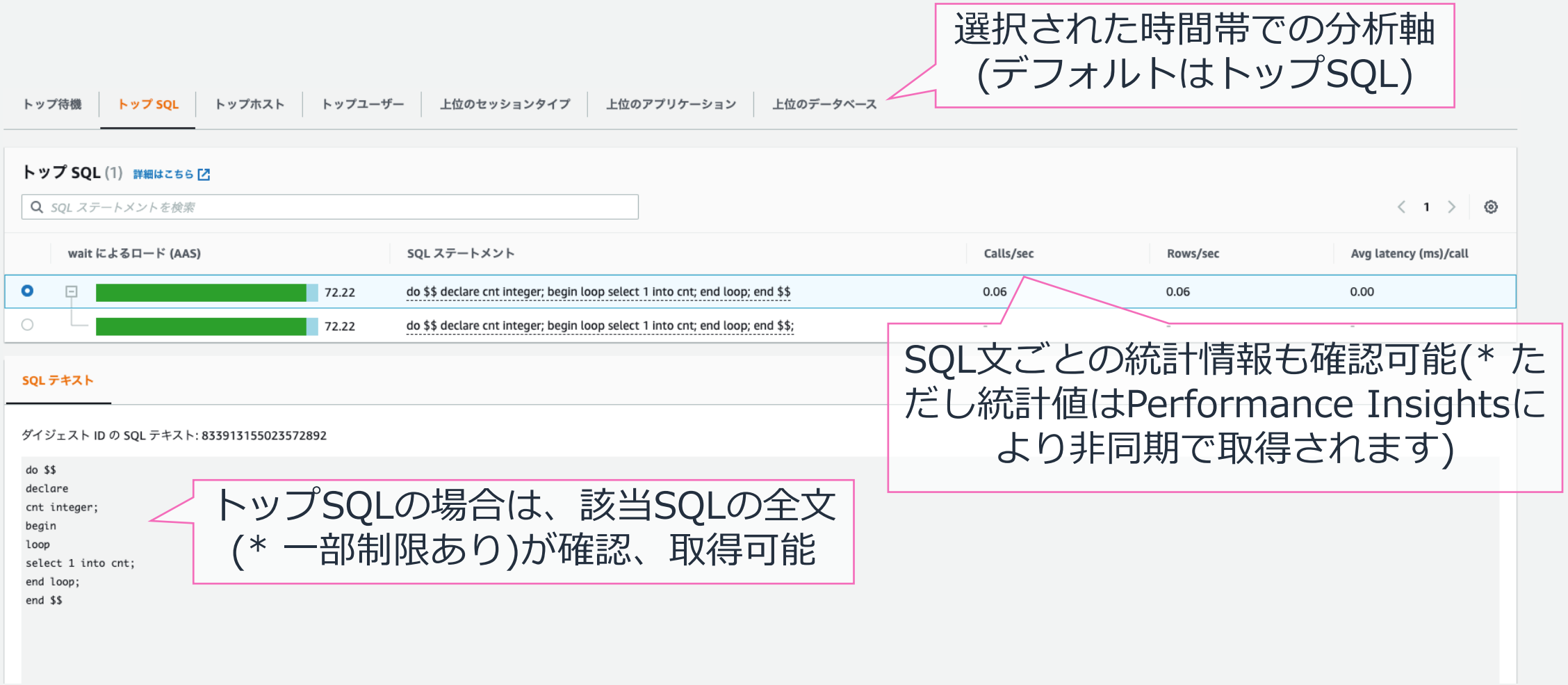
➡ パフォーマンス問題の有無は、Database Load が vCPU を超えているかどうかが基準
➡ 支配的な待機イベント、TopSQL を調査することでパフォーマンス問題の原因を特定可能



Performance Insights を使ったパフォーマンス分析



Performance Insights を使ったパフォーマンス分析



Amazon RDS Performance Insights の位置付け

- Performance Insightsはパフォーマンス分析に必要な情報を自動収集、可視化することができますが、異常の検知、原因の特定、チューニング方法の検討はDBAによるアクションが必要です。

Performance Insightsのカバー範囲	**DBAによるアクションが必要**
• 粒度の細かいパフォーマンスデータの自動収集と長期間保存 • ダッシュボードによるパフォーマンスデータの可視化 • データベースロード(AAS)を元にしたパフォーマンス問題の特定	• パフォーマンス異常の検知 • パフォーマンス異常原因の分析、特定 • チューニング  Amazon DevOps Guru for RDS がこれらのお手伝いをします

Amazon DevOps Guru for RDS の概要

Amazon DevOps Guru の登場

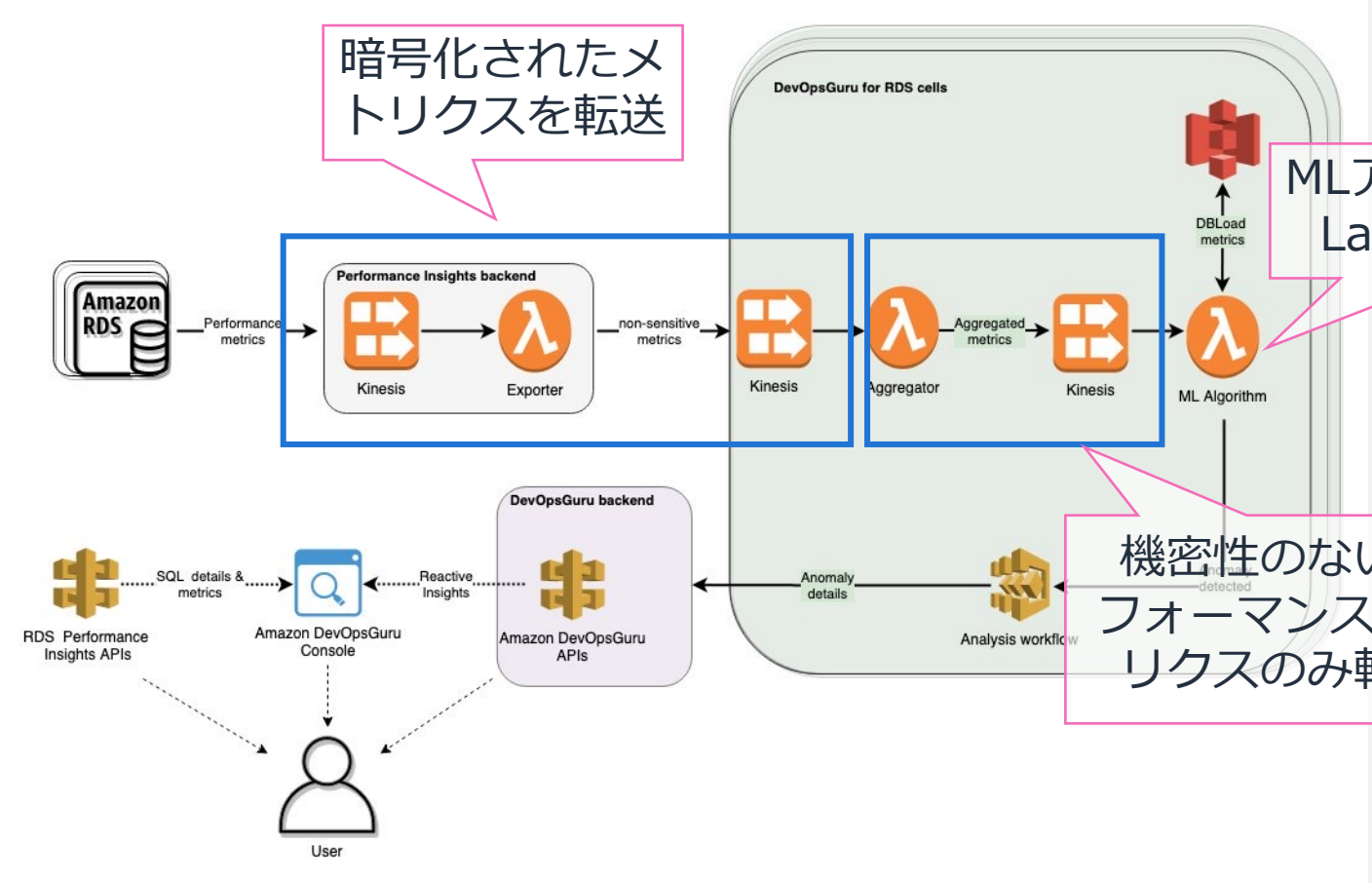
2021年の re:Invent では、機械学習（ML）を利用して、データベースの問題を含むアプリケーションの問題を自動的に検出し、お客様に警告するサービス、DevOps Guruを発表しました。

- 運用上の問題を自動的に検出
- MLを活用したインサイトで問題を迅速に解決
- 可用性を簡単に拡張および維持
- ノイズとアラームの誤検知の軽減

Amazon DevOps Guru for RDS の特徴

- 見つけることが難しいパフォーマンス上のボトルネックや運用上の問題を自動的に検出・診断
- 数分で問題解決のためのインサイトと提案を提供
- Amazon.comの何千ものデータベースを長期間にわたり運用してきた経験に基づくMLモデルを活用
- MLの経験は不要
- 大規模データベースのワークロードを継続的に監視

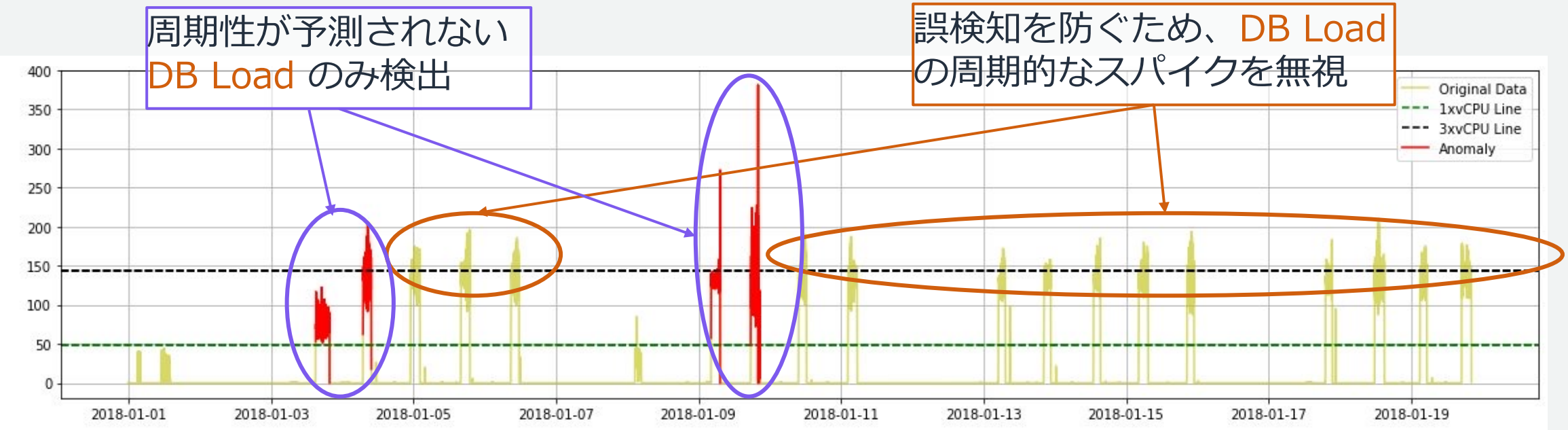
Amazon DevOps Guru for RDS のアーキテクチャ



- DevOps Guruはセルベースアーキテクチャを採用
- 各セルは特定のリージョン内のデータベース・インスタンスの一部のみを処理する孤立したユニットとなり、性能テストが容易な最大サイズのコンポーネントを維持しながら、システムをスケールアウトできます

AWS Database Blog：Amazon DevOps Guru for RDS under the hood
https://aws.amazon.com/blogs/database/amazon-devops-guru-for-rds-under-the-hood/

Amazon DevOps Guru for RDS のアーキテクチャ



AWS Database Blog：Amazon DevOps Guru for RDS under the hood
https://aws.amazon.com/blogs/database/amazon-devops-guru-for-rds-under-the-hood/

誤検出を防ぐため、MLアルゴリズムはバッチジョブなどによる周期的な DB 負荷（DB Load）のような規則的なパターンを識別し、無視するように学習されています

Amazon DevOps Guru for RDS の利用イメージ

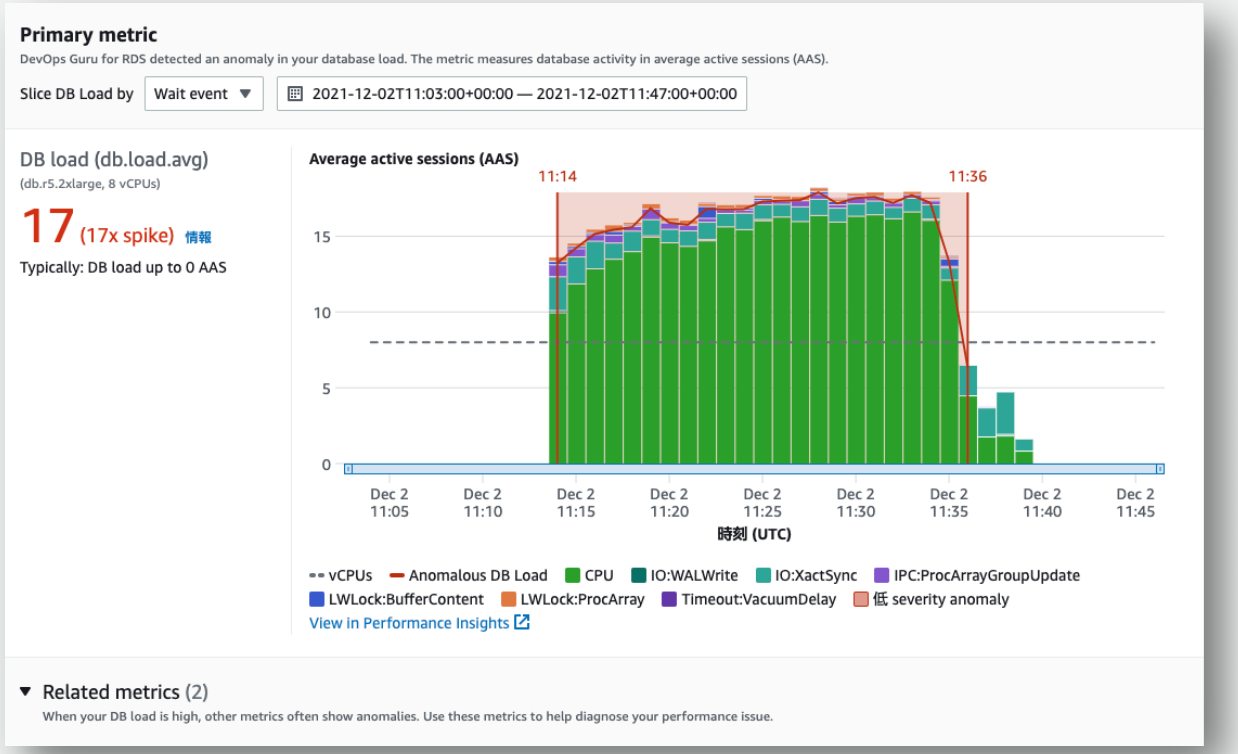
Amazon DevOps Guru for RDS の利用イメージ
ダッシュボード

- 分析されたリソースの数や現在継続中のインサイトなどの情報を集中管理
- インサイト
  - DevOps Guruがリソースを分析し、 Amazon CloudWatchメトリクス、AWS CloudTrail イベント、運用データから異常な動作を検出すると生成され、問題に対処するための推奨事項や関連するメトリクス、イベントに関する情報
- 生成されたインサイトは、Amazon Simple Notification Service (SNS)やAmazon EventBridgeで通知が可能

Amazon DevOps Guru for RDS の利用イメージ
インサイト

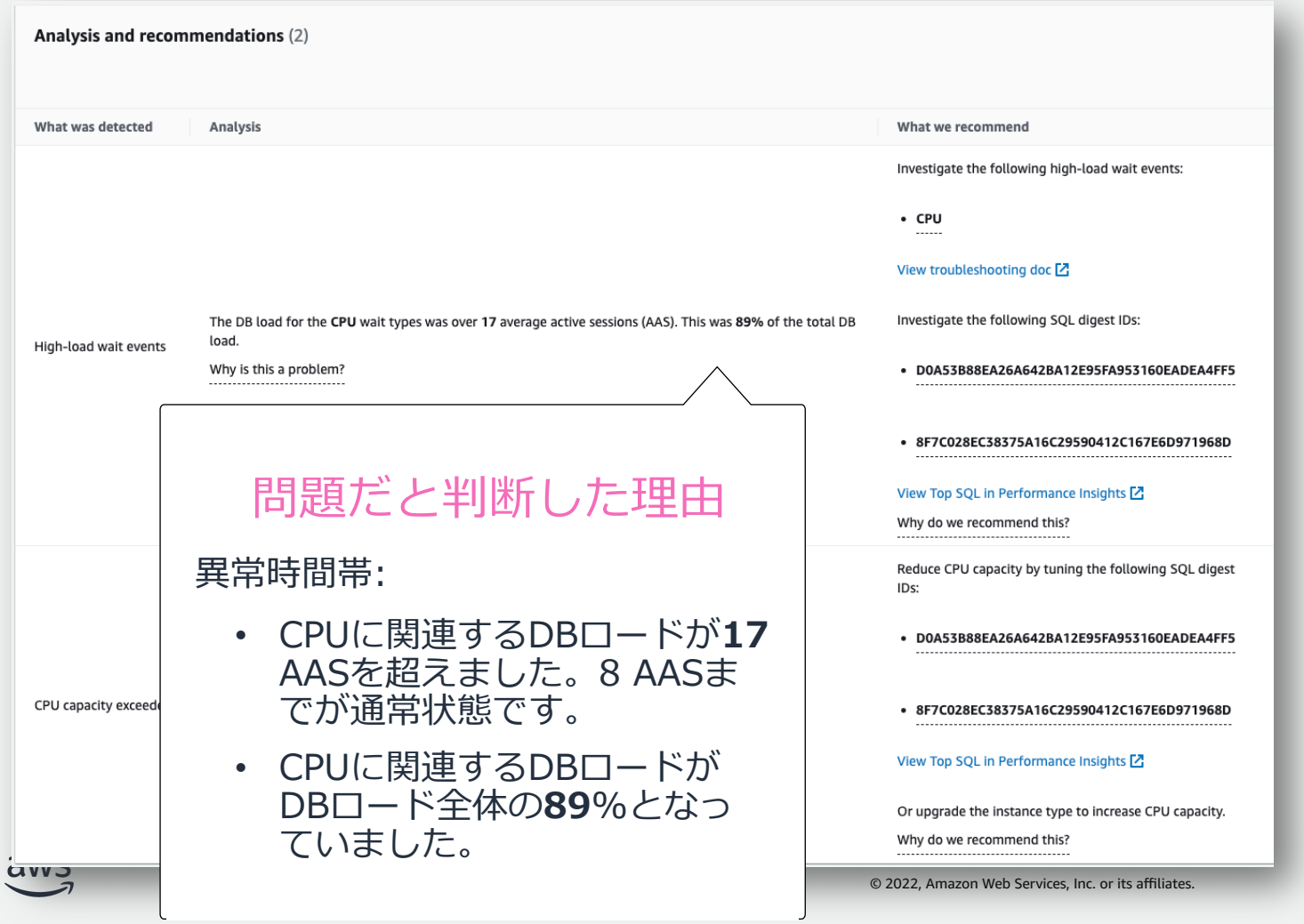
- 報告されたインサイトから、生成されたメトリクスを確認
- DB Load（Performance Insightsのデータ）の場合は、さらに分析を行うことが可能

Amazon DevOps Guru for RDS の利用イメージ
分析とレコメンデーション (1/3)



- DB Loadパフォーマンスメトリクスを使用して問題を検出します。 DB Loadは、平均アクティブセッション (AAS) の単位で測定されます
- DB Loadが仮想 CPU (vCPU) の数よりも大きい場合、問題が発生する可能性があります
- 左の画像は、DevOps Guru for RDS が報告した結果の例です。このグラフは、AASから、ほとんどのユーザーがCPU へのアクセスを待機していたことを示しています

Amazon DevOps Guru for RDS の利用イメージ
分析とレコメンデーション (2/3)



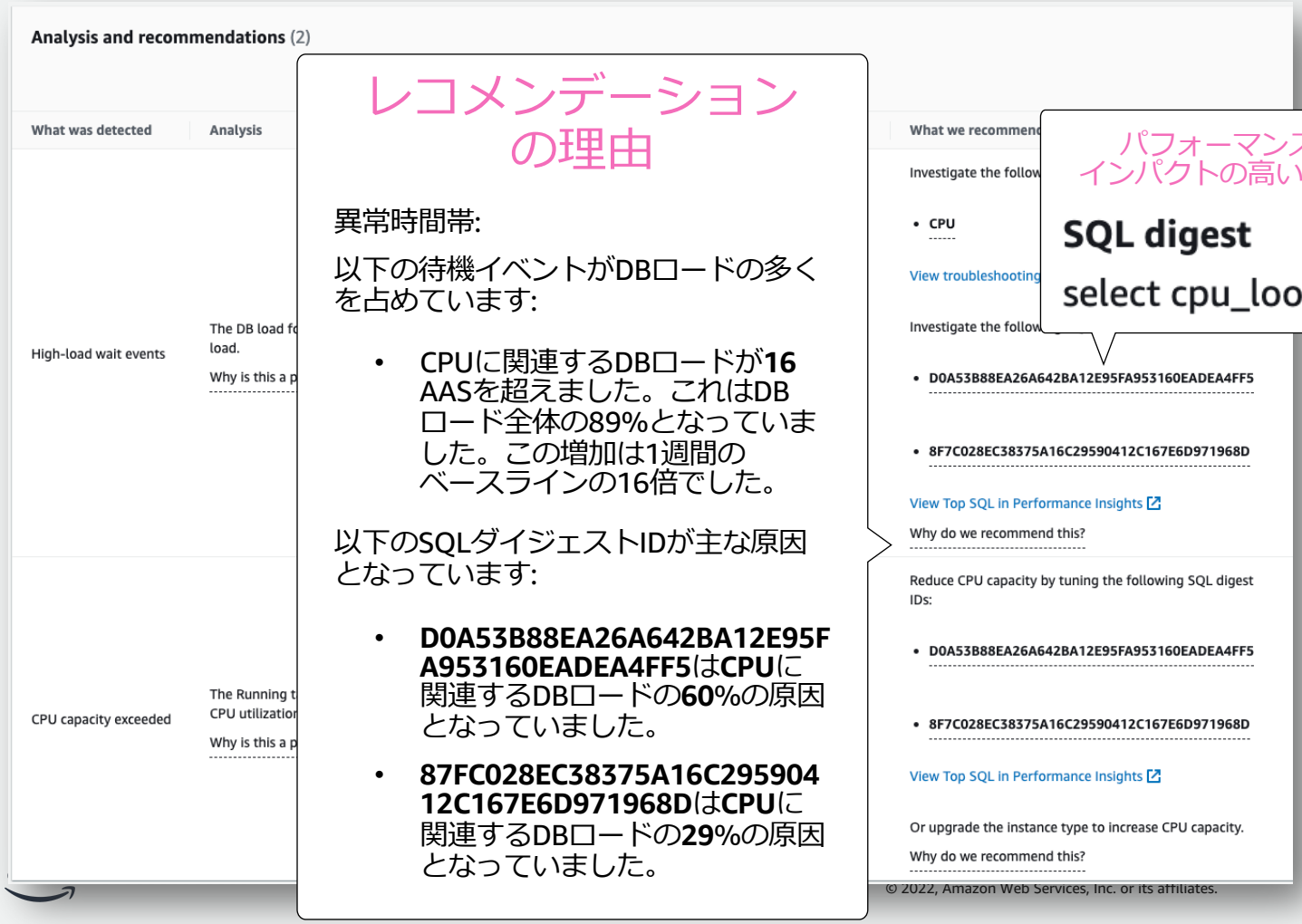
問題だと判断した理由

異常時間帯:

- CPUに関連するDBロードが**17** AASを超えました。8 AASまでが通常状態です。
- CPUに関連するDBロードがDBロード全体の**89**%となっていました。

- CPU の待機タイプに 17 の AAS があり、DB の総負荷の 89% に相当する高負荷の待機イベントを調べています
- データベースには 8 つの vCPU しかなく、実行中のプロセスの推奨数は最大 17（vCPUの約 2 倍）にする必要があります

Amazon DevOps Guru for RDS の利用イメージ
分析とレコメンデーション (3/3)



レコメンデーションの理由

異常時間帯:

以下の待機イベントがDBロードの多くを占めています:

- CPUに関連するDBロードが**16** AASを超えました。これはDBロード全体の89%となっていました。この増加は1週間のベースラインの16倍でした。

以下のSQLダイジェストIDが主な原因となっています:

- **D0A53B88EA26A642BA12E95FA953160EADEA4FF5**は**CPU**に関連するDBロードの**60**%の原因となっていました。
- **87FC028EC38375A16C29590412C167E6D971968D**は**CPU**に関連するDBロードの**29**%の原因となっていました。

パフォーマンスインパクトの高いSQL

SQL digest

select cpu_loops(?)

- DevOps Guru for RDS も、これらの問題を解決するためのレコメンデーションを作成します
- CPUやIO関連の待機イベントからデータベースのボトルネックにインパクトを与えているSQL文を提示

クエリーの実行計画を管理する

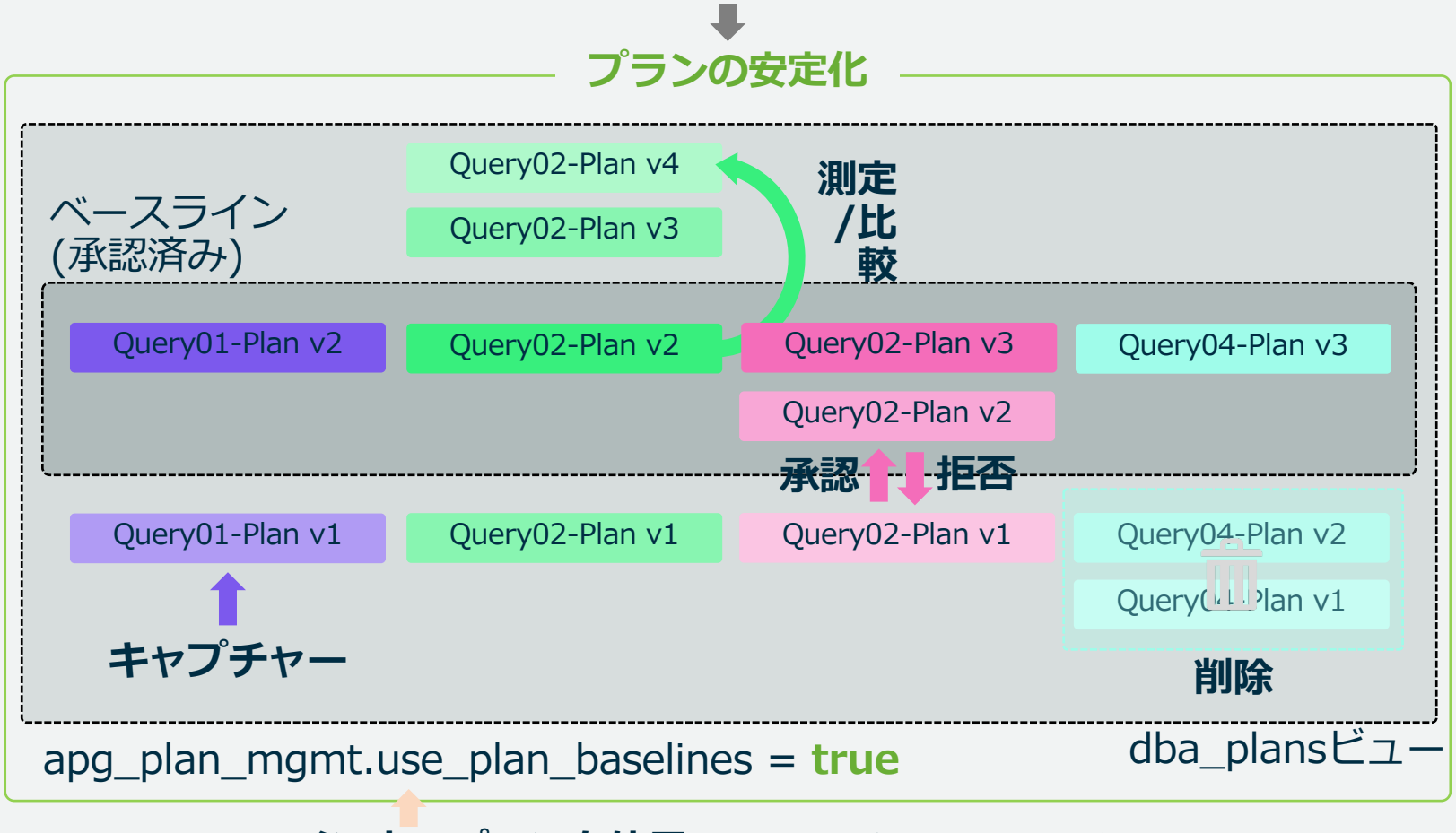
Aurora PostgreSQL: Query Plan Management（QPM）

機能概要

- ✓ 手動/自動でプランのキャプチャー
- ✓ ベースライン内のプランを使用
- ✓ プランの承認/拒否
- ✓ プランの測定/比較
- ✓ pg_hint_planを使ったプランの修正
- ✓ プランの削除
- ✓ プランのエクスポート/インポート

サポートバージョン

- ✓ Aurora PostgreSQL 2.1.0 (PostgreSQL 10.5互換)以降



* デフォルトで32日以上未使用のプランは自動削除 (apg_plan_mgmt.plan_retention_period)

* デフォルトで最大1,000個のプランをキャプチャー (apg_plan_mgmt.max_plans)

Query Plan Management （QPM）のユースケース

- 今まで全くパフォーマンス的に問題がなかったクエリーが、ある日突然パフォーマンスダウンした
- 今回のシステムはとあるベンダーのパッケージ製品のためすぐにSQLの書き換えを行うことができない

Aurora PostgreSQL Query Plan Management (QPM) 機能は、クエリ実行計画不安定の問題を解決します

QPM を使った SQL 実行計画管理

パフォーマンス劣化時の SQL の実行計画と実行時間を確認

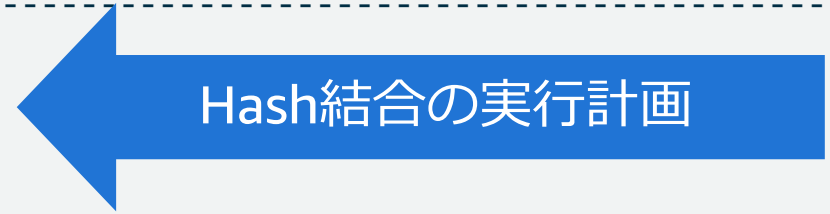
```
EXPLAIN (ANALYZE,HASHES) select * from sample_table_1 a JOIN sample_table_2 b on a.id2=b.id WHERE a.id > 0 and a.id < 210000;

QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
Gather  (cost=10959.17..293715.58 rows=209225 width=126) (actual time=1654.901..2108.682 rows=209999 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Hash Join  (cost=9959.17..271793.08 rows=87177 width=126) (actual time=1654.232..1931.045 rows=70000 loops=3)
         Hash Cond: (b.id = a.id2)
         ->  Parallel Seq Scan on sample_table_2 b  (cost=0.00..155303.67 rows=4166667 width=61) (actual …)
         ->  Parallel Hash  (cost=7847.46..7847.46 rows=87177 width=65) (actual time=27.412..27.412 rows=70000 loops=3)
               Buckets: 65536  Batches: 8  Memory Usage: 3232kB
               ->  Parallel Index Scan using sample_table_1_idx on sample_table_1 a  (cost=0.43..7847.46 rows=87177 width=65) (…)
                     Index Cond: ((id > 0) AND (id < 210000))
 Planning Time: 0.230 ms
 Execution Time: 2116.716 ms
 Plan Hash: -306787140
```

実行計画としてHash結合を選択し、実行時間としては2,116 ms程度

QPM を使った SQL 実行計画管理

QPM ベースライン実行計画確認

```
-[ RECORD 1 ]--------+-------------------------------------------------------------------------------------------
sql_hash             | 699820183
plan_hash            | -306787140
status               | Approved
enabled              | t
sql_text             | select * from sample_table_1 a JOIN sample_table_2 b on a.id2=b.id WHERE a.id > 0 and a.id < 210000;
```



apg_plan_mgmt.dba_plansビューから現在、ベースラインに格納されている実行計画を確認

この時点では、1つの実行計画(Hash結合)のみがベースラインに格納されている

QPM を使った SQL 実行計画管理
チューニング後の SQL をベースラインに取り込む

```
SET apg_plan_mgmt.capture_plan_baselines = manual;

/*+ NestLoop(a b) */ explain select *
                           from sample_table_1 a JOIN sample_table_2 b on a.id2=b.id
                           WHERE a.id > 0 and a.id < 210000;

SET apg_plan_mgmt.capture_plan_baselines = false;
```

apg_plan_mgmt.capture_plan_baselinesを手動に設定(自動も可能)し、

pg_hint_planのヒント句でチューニング対象のSQLに対して別バージョンの実行計画をベースラインに取り込む

QPM を使った SQL 実行計画管理

QPM のベースラインを確認（チューニング承認前）

```
-[ RECORD 1 ]---+-------------------------------------------------------------------------------------------
sql_hash        | 699820183
plan_hash       | -306787140
status          | Approved
enabled         | t
sql_text        | select * from sample_table_1 a JOIN sample_table_2 b on a.id2=b.id WHERE a.id > 0 and a.id < 210000;
-[ RECORD 2 ]---+-------------------------------------------------------------------------------------------
sql_hash        | 699820183
plan_hash       | 1809927363
status          | Unapproved
enabled         | t
sql_text        | select * from sample_table_1 a JOIN sample_table_2 b on a.id2=b.id WHERE a.id > 0 and a.id < 210000;
```

Hash結合の実行計画

Nested Loop結合の実行計画

チューニング後の実行計画は、UNAPPROVED(未承認)としてベースラインに取り込まれていることが分かる

QPM を使った SQL 実行計画管理
チューニング後の SQL を承認

SELECT apg_plan_mgmt.set_plan_status(699820183, 1809927363, '**Approved**');

SELECT apg_plan_mgmt.set_plan_status(699820183, -306787140, '**Unapproved**');

チューニング後のNested Loopの実行計画のステータスをApproved(承認)に変更し、合わせてチューニング前のHash結合の実行計画をUnapproved(未承認)に変更

QPM を使った SQL 実行計画管理

QPM のベースラインを確認(チューニング承認後)

```
-[ RECORD 1 ]--------+-------------------------------------------------------------------------------------
sql_hash             | 699820183
plan_hash            | -306787140
status               | Unapproved
enabled              | t
sql_text             | select * from sample_table_1 a JOIN sample_table_2 b on a.id2=b.id WHERE a.id > 0 and a.id < 210000;
-[ RECORD 2 ]--------+-------------------------------------------------------------------------------------
sql_hash             | 699820183
plan_hash            | 1809927363
status               | Approved
enabled              | t
sql_text             | select * from sample_table_1 a JOIN sample_table_2 b on a.id2=b.id WHERE a.id > 0 and a.id < 210000;
```

Hash結合の実行計画

Nested Loop結合の実行計画

チューニング後の実行計画が、Approved(承認)に変更され、チューニング前の実行計画がUnapproved(未承認)に変更されていることが分かる

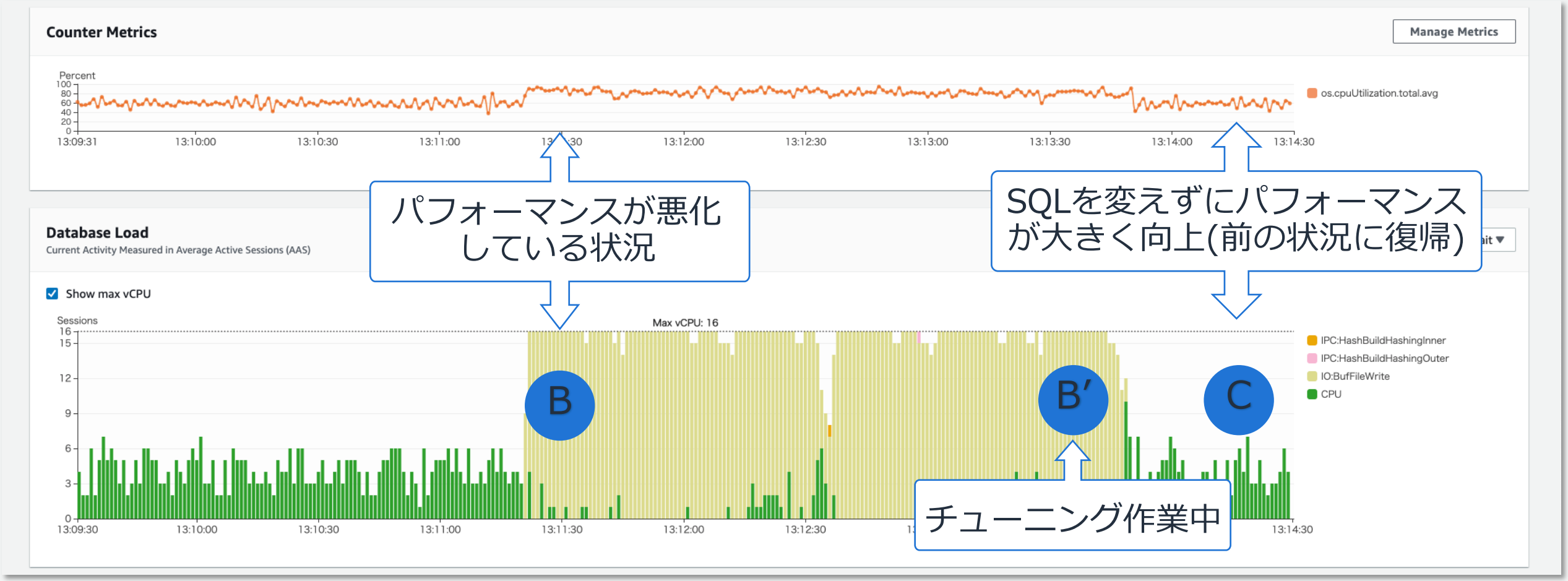
QPM を使った SQL 実行計画管理

QPM のベースライン承認後の SQL の実行計画と実行時間を確認

```
SET apg_plan_mgmt.use_plan_baselines = true;
EXPLAIN ANALYZE select * from sample_table_1 a JOIN sample_table_2 b on a.id2=b.id WHERE a.id > 0 and a.id < 210000;

QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------
 Gather  (cost=1000.87..297357.43 rows=209225 width=126) (actual time=0.263..236.801 rows=209999 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Nested Loop  (cost=0.87..275434.93 rows=87177 width=126) (actual time=0.050..203.592 rows=70000 loops=3)
         ->  Parallel Index Scan using sample_table_1_idx on sample_table_1 a  (cost=0.43..7847.46 rows=87177 width=65) (…)
               Index Cond: ((id > 0) AND (id < 210000))
         ->  Index Scan using sample_table_2_idx on sample_table_2 b  (cost=0.43..3.06 rows=1 width=61) (…)
               Index Cond: (id = a.id2)
 Planning Time: 0.502 ms
 Execution Time: 245.751 ms
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 699820183, Plan Hash: 1809927363, Minimum Cost Plan Hash: -306787140
```

実行計画としてNested Loop結合を選択し、実行時間としては245 ms
(チューニング前は2,116 ms)に改善している

チューニング前後の状況の確認



まとめ

まとめ

- Performance Insights の特徴
  - パフォーマンス問題を適切に分析するのに必要な様々なメトリクスを必要な粒度で取得可能、
    ユーザー側の管理も不要
  - シンプルかつドリルダウン可能なパフォーマンス指標を使って簡単に分析が可能
- DevOps Guru for RDS
  - パフォーマンス問題の発生状況をタイムリーに監視。通常とは異なるアクティビティを検知できます
- Aurora PostgreSQL Query Plan Management
  - SQL文の修正を伴わず、SQLの実行計画を管理、チューニングを行うことが可能

Thank you!