スパイクアクセス対策としての pitchfork 導入

Ruby World Conference 2024

Sim Sangyong@STORES

Self introduction

- Sangyong Sim @ STORES. Inc
- shia @ Internet
- riseshia @ {X, GitHub}

STORES ネットショップ

STORES ネットショップ

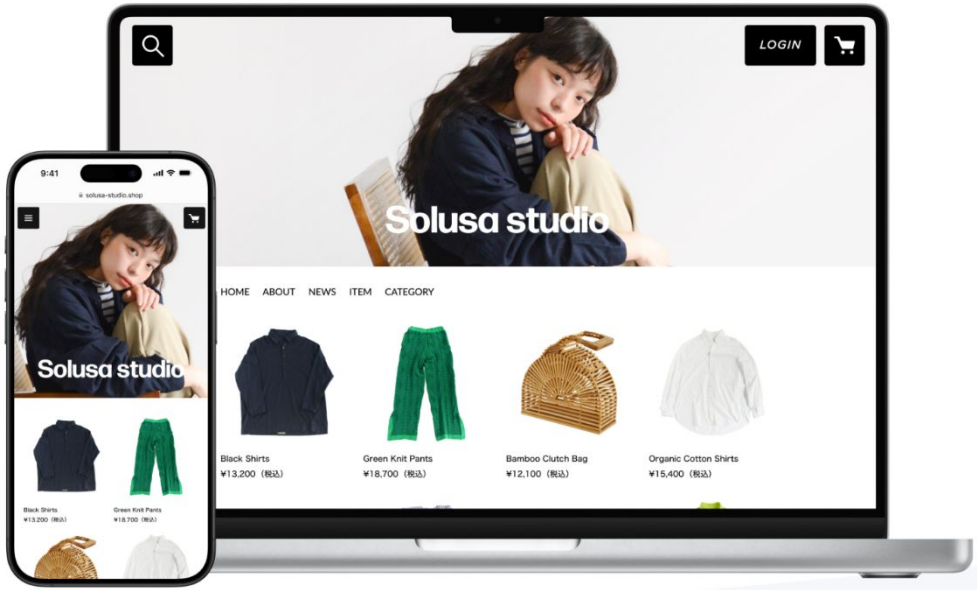
概要　料金　デザイン　機能　事例　よくある質問　ログイン　アカウント作成

はじめての人も
いますぐ、かんたんに

アカウントを作成してはじめる

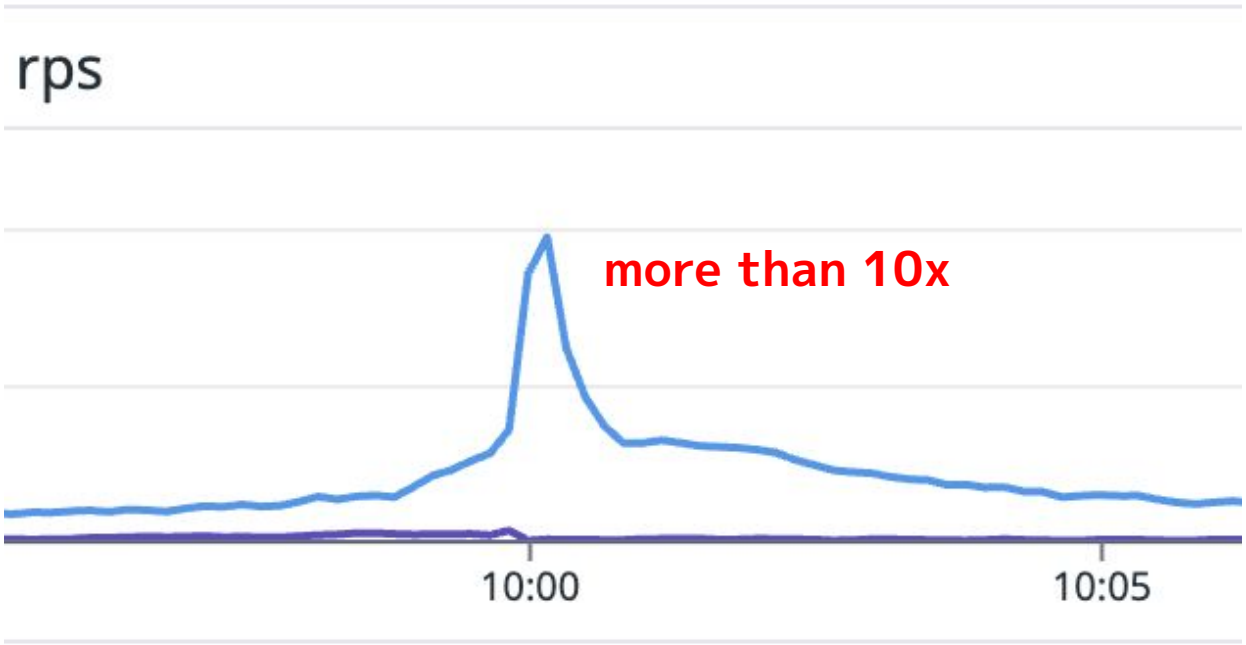
レジやアプリもご検討の方

STORES で
オムニチャネルの
CRMを実現 >

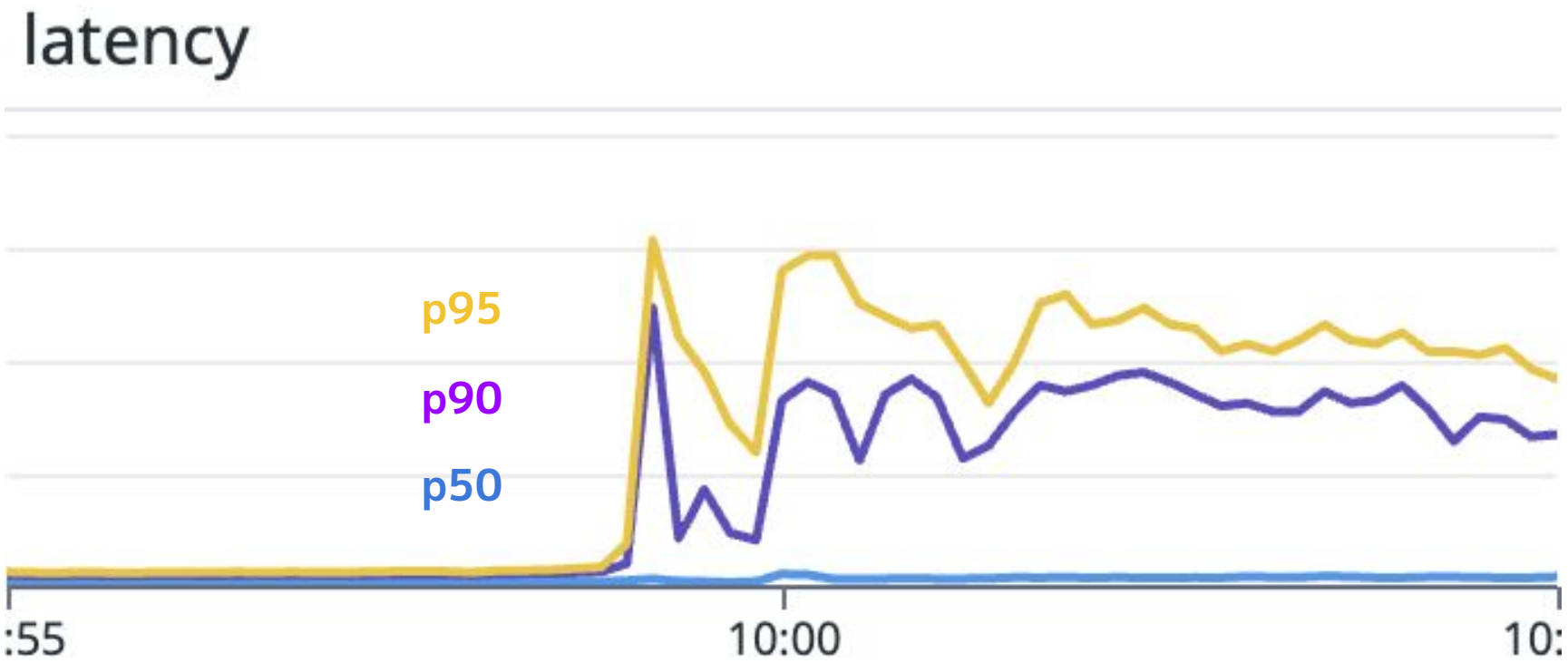


- 多様な規模の事業者
- 特定時刻から販売を開始することができる
- 規模を正しく予測するのは難しい

xx/xx 10時から数量限定グッズ販売開始します！！！



レイテンシが劣化する



目標

(できれば何もせずに) スパイクアクセス時にも安定した購入体験ができるようにしたい！！！

課題

- リクエストをできるだけ待たせない ~= 十分な数のWebサーバのワーカーを用意する
- p90 あたりから観測されるレイテンシ劣化を改善する

注: このセッションではアプリケーション高速化およびキャッシングによる負荷軽減はスコープ外なので話しません

環境

- ECS Fargate 上で動く
- ASG(Auto Scaling Group) でキャパシティ管理する
- Ruby on Rails / unicorn で動く

課題 - 十分な数のWebサーバのワーカーを用意する

- 正確にトラフィックを予測することはできないので過去の実績ベースで戦略を考える
- 予想を超えてしまった場合はしょうがないので待ちを許す(しかない)
  - ほとんどのスパイクのピークは 1分以下で 5分以内でほぼ捌き終わるので、 ASG では間に合わないため

課題 - 十分な数のWebサーバのワーカーを用意する

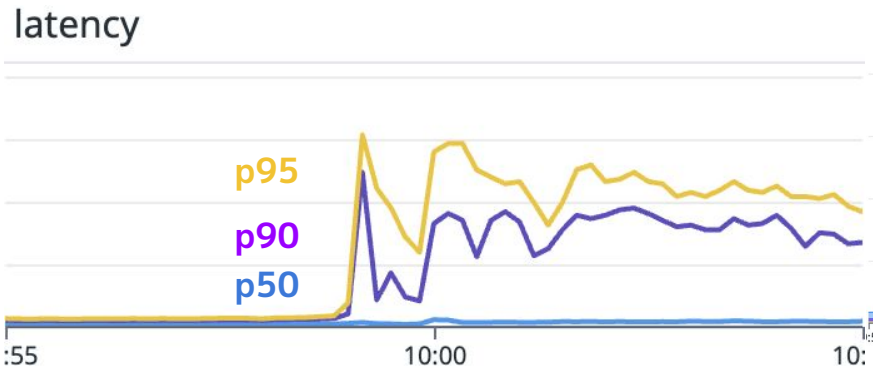
小規模のもの

- 常に過剰キャパシティを持ってスパイクが発生したらそれで吸収する
- ECS Fargate Spot で格安で運用できている

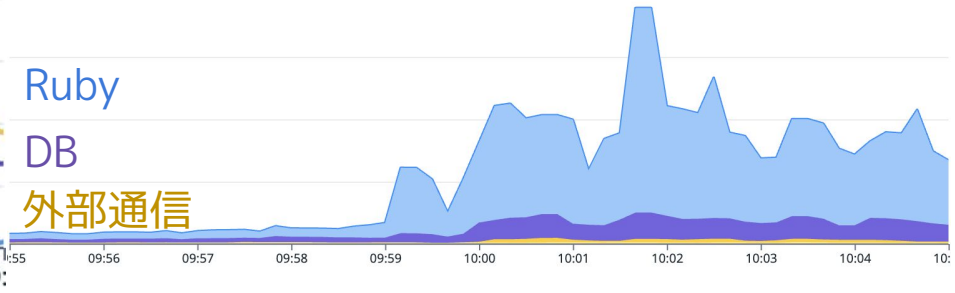
大規模なもの

- まれに来るそれ以上のスパイク、規模感から事前に把握してることが多く、販売直前でサービスをスケールアウトする

課題 - レイテンシ劣化を改善する



リクエスト平均処理時間を
内部の処理時間で分類



課題 - レイテンシ劣化を改善する

もしかして Webサーバのワーカー、温まってない...?

Webサーバのワーカー、温まってないとは

Webサーバ（Rails アプリケーション）は起動して実際リクエストが処理することで初めて走る処理が色々あり、それらによって起動直後は遅いことがある

- 各種の TCP コネクション生成
- インメモリーキャッシュ生成
- (YJIT を有効にしている場合) JIT コンパイル
- method_missing から始まるメタプロ
- Action View のコンパイル
- …

なぜ一部だけ？ - 実験

unicorn でリクエストを処理する時、どのワーカーが仕事していたのかの確認をしてみる

- 処理に 0.1s かかるエンドポイント
- ワーカー数 8
- 低負荷の再現するため 2並列
- 10s 負荷

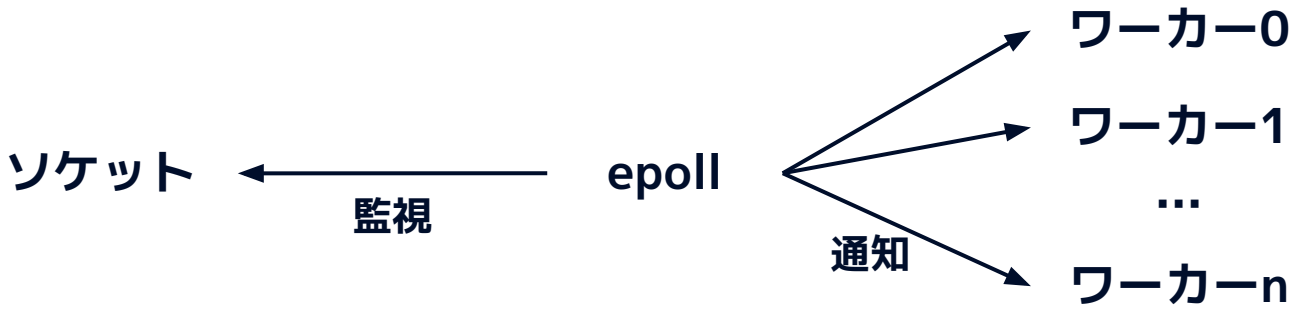
各ワーカーが処理したリクエストの数を調べてみる

なぜ一部だけ？ - 実験

- worker 0: 85
- worker 1: 86
- worker 2: 2
- worker 3: 0
- worker 4: 0
- worker 5: 0
- worker 6: 0
- worker 7: 0

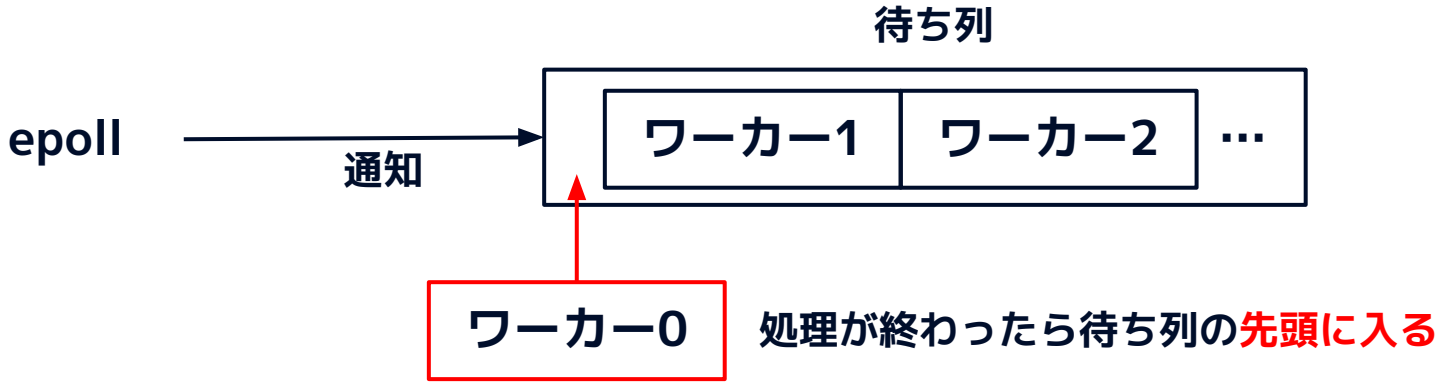
注：Linux 環境のみ再現します

なぜ偏る？



- unicorn は prefork 型 web サーバ
- 起動して要求された数のワーカーを fork し新しいプロセスを生成
- １つの TCP ソケットが共有される
  - unicorn では epoll(or kqueue) というのが使われる
- この通知順番はどうなっているか

なぜ偏る？



- リクエストが来た時、それを処理するワーカーが順番に並んでる
  キューを想像すると、そのキューは LIFO
  - 処理が終わったワーカーがキューに入ったら、次のリクエスト時にも同じワーカーが選ばれるので偏る

Ref: https://blog.cloudflare.com/the-sad-state-of-linux-socket-balancing/

つまり起きてるのはおそらく

- スパイクに備えて過剰キャパシティを確保する
- 過剰に確保されたワーカーは起動してから仕事していない
- 販売開始時刻の大量のリクエストにより遊んでいたワーカーが仕事を始める
- 温まってないので処理に時間がかかる…？

どうやって全ワーカーを温める？

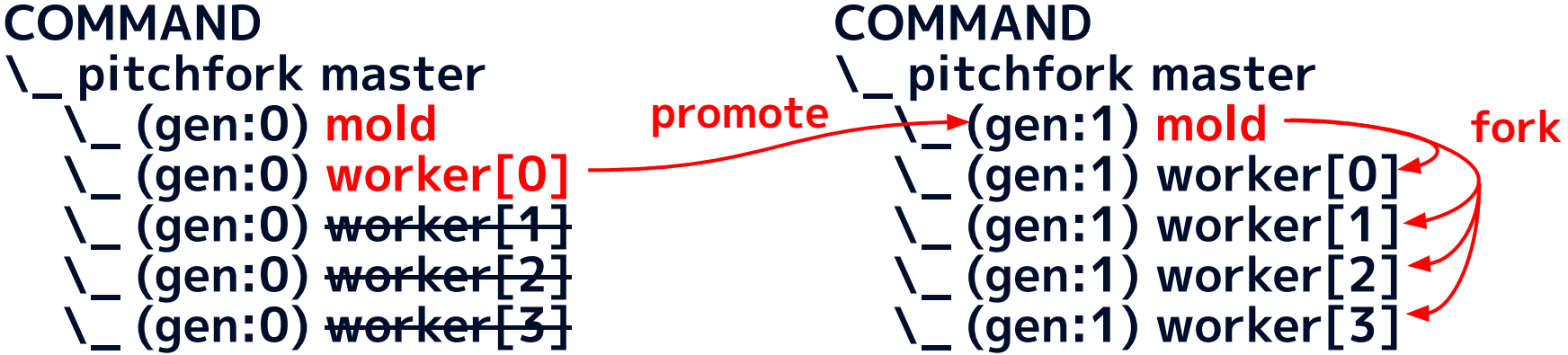
- 実際トラフィックを作って温める
- 温まった状態でサービスインする
- puma にする
- ??

pitchfork

- Shopify による unicorn の fork
- refork という機能がある

pitchfork - refork

- 一定数(adjustable)のリクエストを処理したワーカーをテンプレートとして全ワーカーを再度 forkする
- Copy on Write(CoW) による共有メモリーを増やしてメモリー使用量を減らす戦略



温まったワーカーを refork すると

全ワーカーが温まった状態になるのでは？

導入

- pitchfork が問題ないか確認するために開発環境でしばらく運用
- 本番を徐々にロールアウト

導入の注意点

fork safety 確認が必要

- コネクションが継承されるとか
- バックグラウンドで動くスレッドの扱いとか

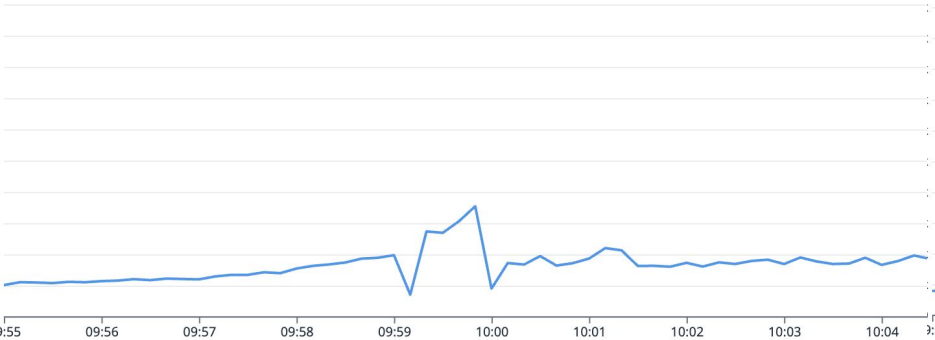
相性が悪い事例もあるので気をつける

Ref: https://github.com/Shopify/pitchfork/blob/master/docs/FORK_SAFETY.md

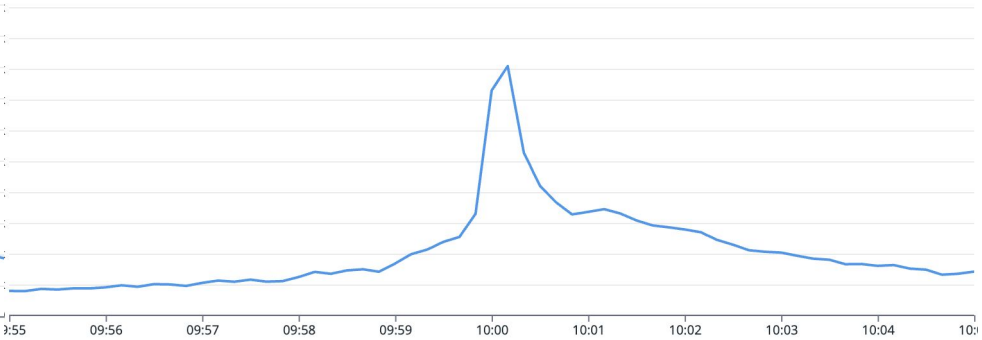
導入結果

毎年定期的に開催されている大きい販売の比較。

グラフの高さは同じスケールに調整されてます。



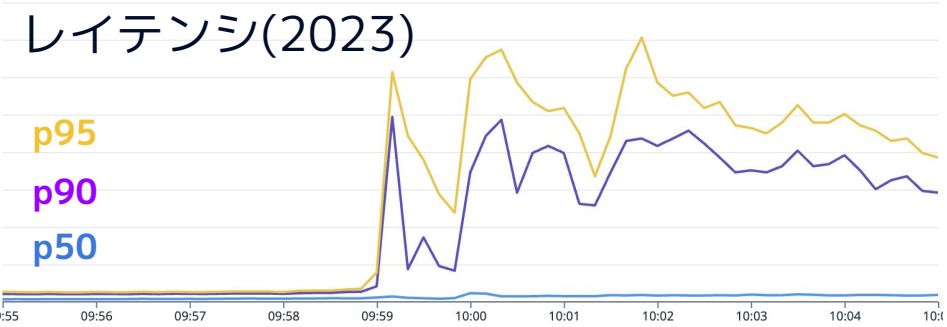
rps(2023)



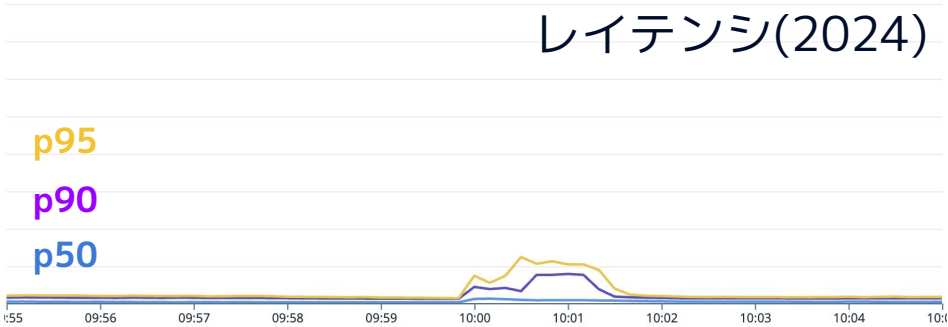
rps(2024)

導入結果

レイテンシ(2023)

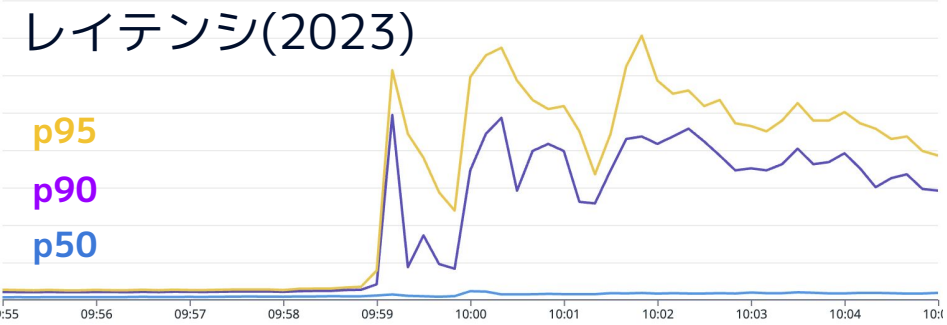


レイテンシ(2024)

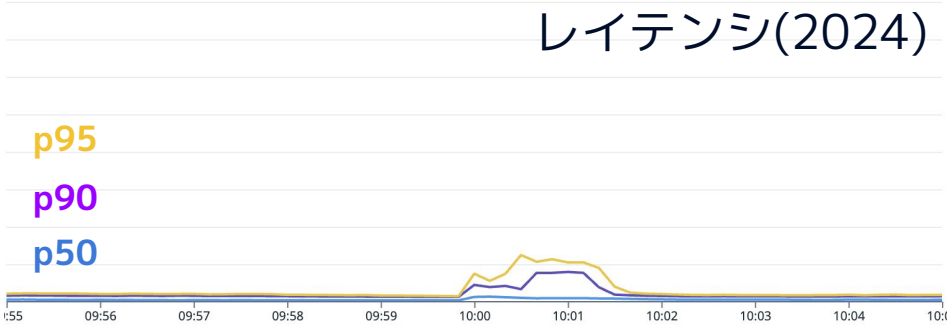


導入結果

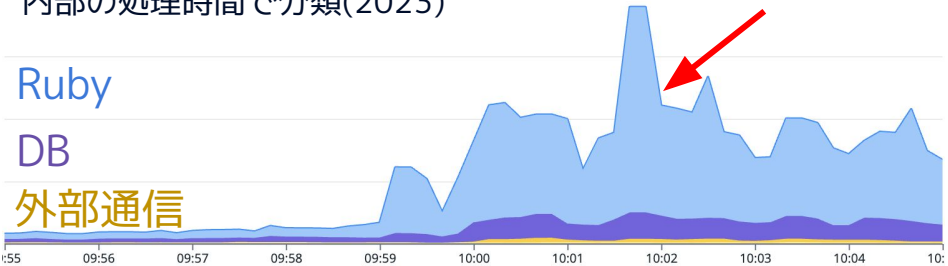
レイテンシ(2023)



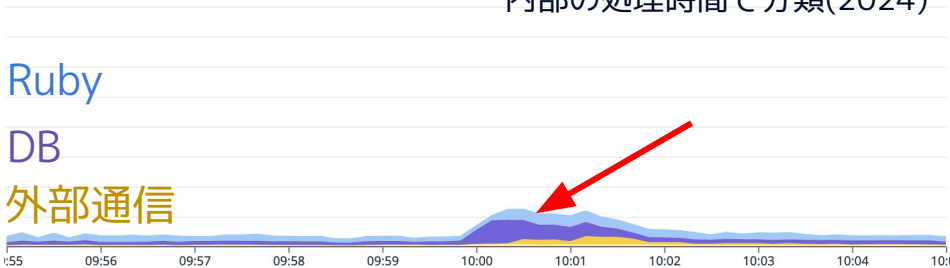
レイテンシ(2024)



リクエスト平均処理時間を
内部の処理時間で分類(2023)



リクエスト平均処理時間を
内部の処理時間で分類(2024)



まとめ

不規則なスパイクアクセスの処理のため、低コストの効率的な暖気手段として pitchfork を試して一定の成果がありました

まとめ

ご清聴ありがとうございました