aws DEV DAY
ONLINE JAPAN

A - 7

ECS×Fargateで実現する運用コストほぼ0なコンテナ運用の仕組み

コネヒト株式会社　永井勝一郎(shnagai)

In Partnership with intel

自己紹介

永井 勝一郎

コネヒト株式会社　テクノロジー推進G

インフラ/機械学習

@shnagai

主な活動:

- AWS DevDayは昨年に引き続き2度目
- コネヒトエンジニアブログ: https://tech.connehito.com/archive/author/nagais

mamari

About コネヒト

人の生活になくてはならないものをつくる

私たちは「人の生活になくてはならないものをつくる」をミッションに
家族のライフイベントにおける意思決定をITの力でサポートする会社です

About ママリ

新ママの「3人に1人」にご利用いただいているアプリ、情報メディア「ママリ」

Q&Aコミュニティ

ユーザーが悩みを投稿、相談しあうQ&A機能
専門家による回答も期間限定で提供

メディア

妊娠・育児などの記事をwebとアプリで配信

新ママの

3人に1人 が利用中! ※2

アプリ会員数
250万人

ママ向け**No.1**アプリに選出 ※1

「第８回健康寿命をのばそう！アワード」にて

厚生労働省子ども家庭局長賞 受賞

※1 1,084人のママが選ぶ「現在使っているアプリ」 にて、5項目 (他のママにオススメしたい、認知度、利用率、利便性、好感度) で
１位を獲得しました。「ママ向けNo.1アプリ」は2019年3月インテージ調べ　調査対象：妊娠中～2歳0ヶ月の子供を持つ女性（n=1,084）を抽出
※2 2019年の「ママリ」内の出産予定日を設定したユーザー数と、厚生労働省発表「人口動態統計」の出生数から算出

今日お話すること

- 4年に渡るECS運用を運用負荷/費用削減の観点から振り返ります
  - Fargateの登場で変わったECS運用
  - ECS ターゲット追跡ServiceAutoScallingを活用した動的キャパプラ

Fargateを使ってECSでコンテナ運用する際に
役立つ気付きを一つでも
持ち帰っていただければと思っています。

サービス運用の理想像

- 運用コストを極限まで少なくして安定稼働してほしい
  - クラウド、各種SaaSの利用
  - 自己回復型アーキテクチャの採用(コンテナ、マネージドサービス活用)
- それにかかる費用は少なければ少ないほど良い
  - 同じことをやるのであれば低コストで出来る方が価値は高い

ECSとは?

- AWSマネージドなコンテナのオーケストレーションツール
  - コンテナのライフサイクル管理(スケジューリング、セルフヒーリング)
  - AWSマネージドなのでオーケストレーションツール自体の運用が不要

ECSの気に入っているところ

- メンテナンスコストが低い
  - コントロールプレーンに関してはほぼ意識する必要はない
  - コンテナをどう動かすかに注力出来る
- AWSのリソースとフレンドリー
  - ALB,IAMロール,CloudWatchEvents,StepFunctions
  - コネヒトはAWSでほぼ全てのシステムを組んでいるので連携が楽なのは大事
- 学習コストも低め
    - サービス、タスク定義、クラスタを最低限抑えればコンテナ運用出来る
    - 自由度は低いが、コンテナ運用にあたり設定項目や考慮事項が少ない

コネヒトでのECS運用の歴史

2017年7月	ECS本番運用開始	Dockerを使ったコンテナの本番運用開始 EC2バックエンド
2017年12月	全サービスをECS化	既存サービスの移行が完了し全サービスをECSで運用 EC2バックエンド
2018年10月	Fargate本番導入	Fargateを本番運用開始 運用コスト激減だが費用を抑えるために様々な工夫
2020年5月	Fargate × ターゲット追跡 ServiceAutoScalling	Fargateの柔軟性を駆使した動的キャパプラを実現 運用コストほぼ0で費用も削減

mamari

ECS×EC2構成はよしなにEC2の運用コストが発生

- EC2自体の運用
  - よしなにAMI更新
  - よしなにEC2メンテナンス対応
  - よしなにEC2を監視
- ECS関連の運用
  - **タスク数を事前に見越したキャパシティプランニングが必要**
    - どのEC2で動かすか
  - タスクの配置確認(無駄なく配置されているか)
  - 手動スケールアウト/イン
  - クラスタをどう組むか

ECS×EC2構成はよしなにEC2の運用コストが発生

- EC2自体の運用
  - よしなにAMI更新
  - よしなにEC2メンテナンス対応
  - よしなにEC2を監視
- ECS関連の運用
  - **タスク数を事前に見越したキャパシティプランニングが必要**
  - タスクの配置確認(無駄なく配置されているか)
  - 手動スケールアウト/イン
  - クラスタをどう組むか

一つ一つは軽い..
が積み重なると大きな運用コストに

ECS×EC2構成はオートスケールが複雑

- タスクを動かす基盤のEC2とタスク自体のスケールを考慮する必要がある
  - 出来なくはないが複雑な仕組みを自前で運用しなければならない
  - ECS Cluster Auto Scalingが解決してくれるかも??

    https://aws.amazon.com/jp/about-aws/whats-new/2019/12/amazon-ecs-cluster-auto-scaling-now-available/

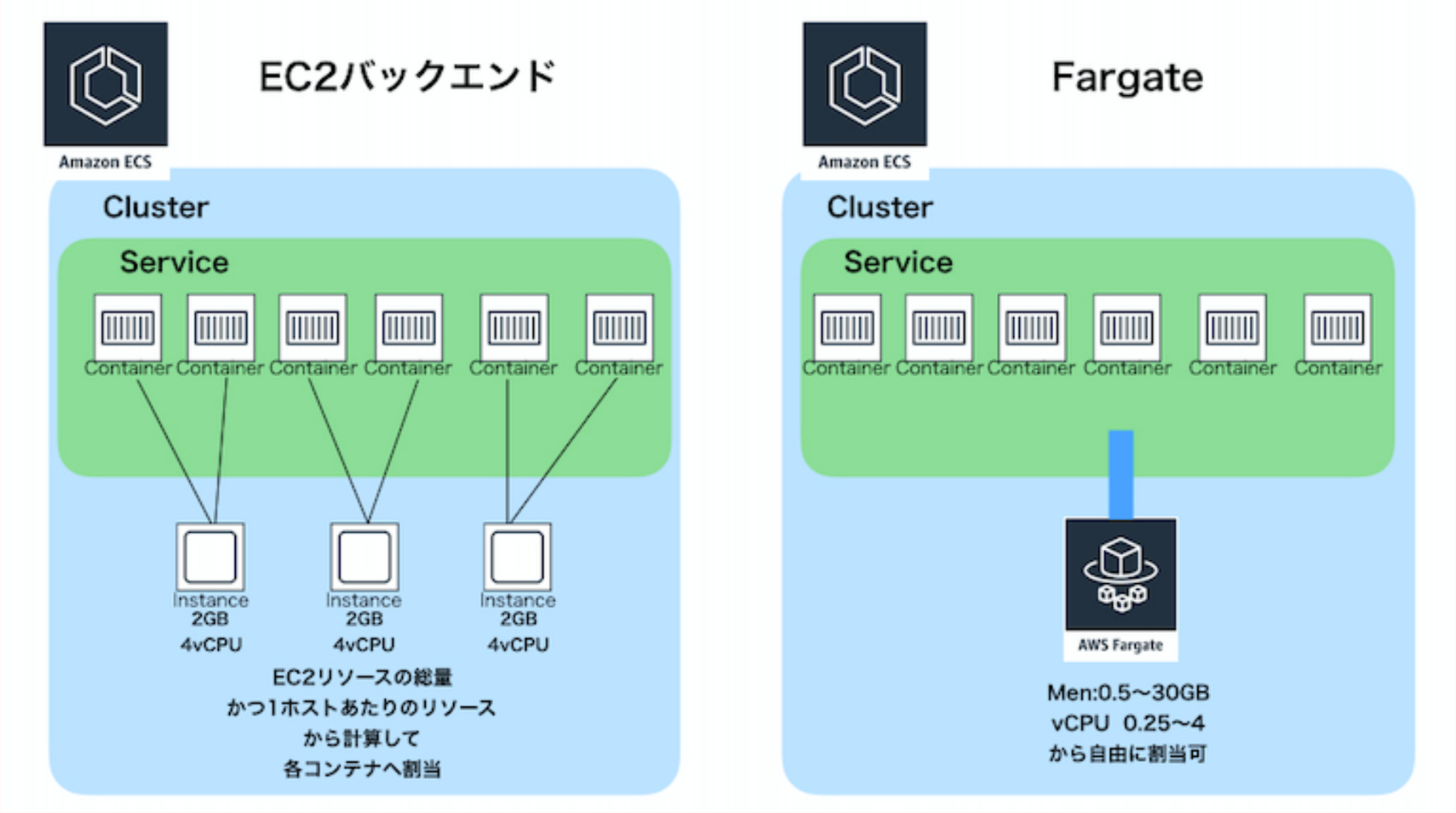
- **運用コスト>費用** ピークトラフィックを捌けるリソースを事前に用意
  - 少数のインフラとWebエンジニアでサービス運用
  - 複雑な仕組みを導入した際の運用コストは大きい

Fargateの登場

Fargateとは

- AWSマネージドなコンテナ向けコンピューティングエンジン
  - EC2不要
- 柔軟なリソースと従量課金
  - Mem:0.5〜30,vCPU:0.25〜4の範囲で柔軟に割り当て
  - 使った分だけ従量課金

Fargateの登場でリソース割当の考え方が変わった



Fargateの登場でECSの運用が大幅に楽になった

そもそもEC2が不要に

- EC2自体の運用
  - よしなにAMI更新
  - よしなにEC2メンテナンス対応
  - よしなにEC2を監視

すべてリソース割り当ての問題

- ECS関連の運用
  - **タスク数を事前に見越したキャパシティプランニングが必要**
  - タスクの配置確認(無駄なく配置されているか)
  - 手動スケールアウト/イン
  - クラスタをどう組むか

EC2から解放されたが費用が高い

- RIでEC2→Fargate(SavingsPlansなし)だと約2倍の費用
- EC2バックエンド時代と同じタスク数でFargateに移行
  - 東京リージョン対応後数ヶ月でまだ事例もなかったので安全に
    https://tech.connehito.com/entry/2018/11/21/163534
- 費用を下げるためにタスク数を減らしていった
  - EC2バックエンド時代デプロイ用に確保されたリソース分を徐々に減らす
  - サービス運用に問題のないタスク数を見極める

Fargate×Compute Savings Plansの登場

- Savings Plansの登場でFargateも少し安く使えるようになった
  - 全額前払いでオンデマンドの22%OFFで利用可能
- 事前に利用想定額をコミットして前払いするスタイル
  - マネージドコンソールの請求→推奨事項で利用状況による見積り可
  - Fargate利用の7,80%をコミットして前払いするような使い方をしている

Fargate Spotの登場

- オンデマンド価格から最大70%OFFで利用可能
- Spotタイプなので強制中断あり
  - 空きキャパがないと2分前に通知し中断
- ステージング環境や中断してもいいバッチ処理に利用
  - 安定稼働が一番大事なので本番のウェブには使っていない

EC2とFargateの料金比較

EC2RI→Fargate
約$45/month * 台数分高い
※SVなし

1時間あたりの費用比較 ($/hour)

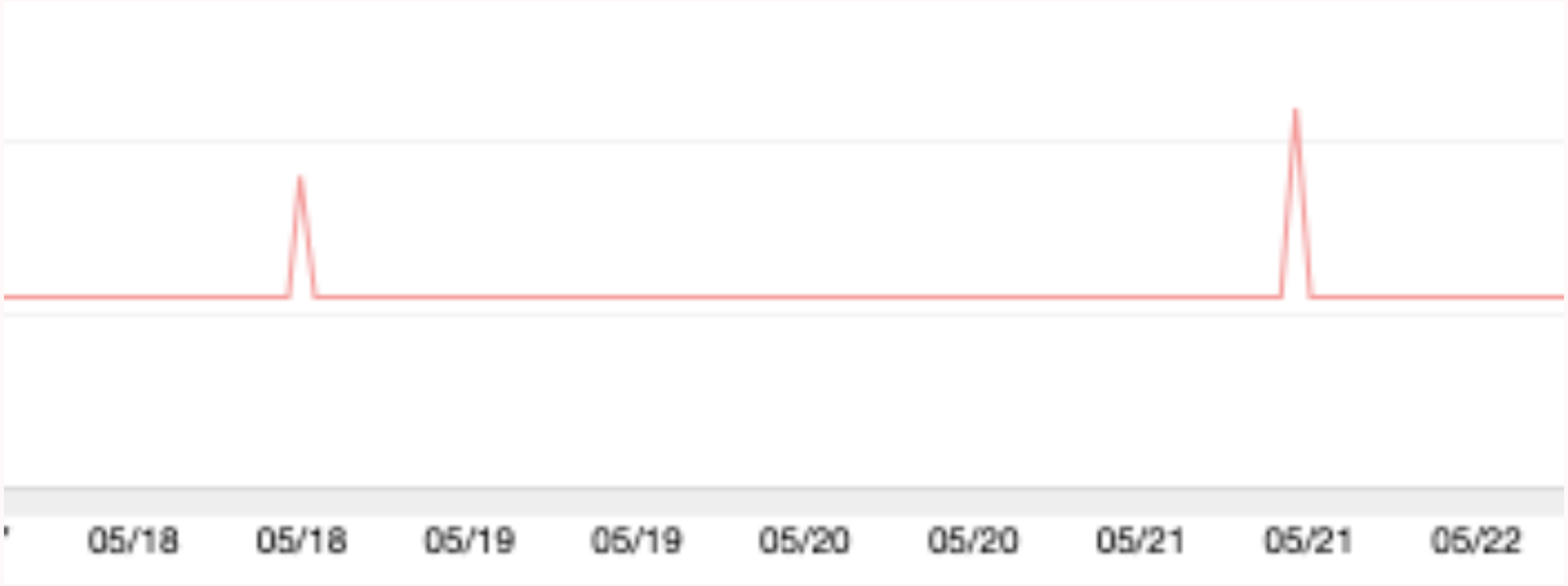
ap-northeast-1	オンデマンド	RI / Compute Savings Plans	割引率
EC2 C5.large	0.107	0.063	全額前払い 41%
Fargate 2vCPU mem4G	0.12324	0.0961272	全額前払い 22%
Fargate - EC2	0.01624	0.0331272	

EC2RI→FargateSV
約$24/month * 台数分高い

mamari

定常的なタスク数減らした分のスパイク対策はオートスケールで

- CPU使用率のしきい値60%で発動するオートスケール
  - スケールアウト/イン時の他タスクへの考慮不要になったので採用
  - 起動に50sくらいのバッファが必要なので瞬間的なスパイクに弱い…



mamari

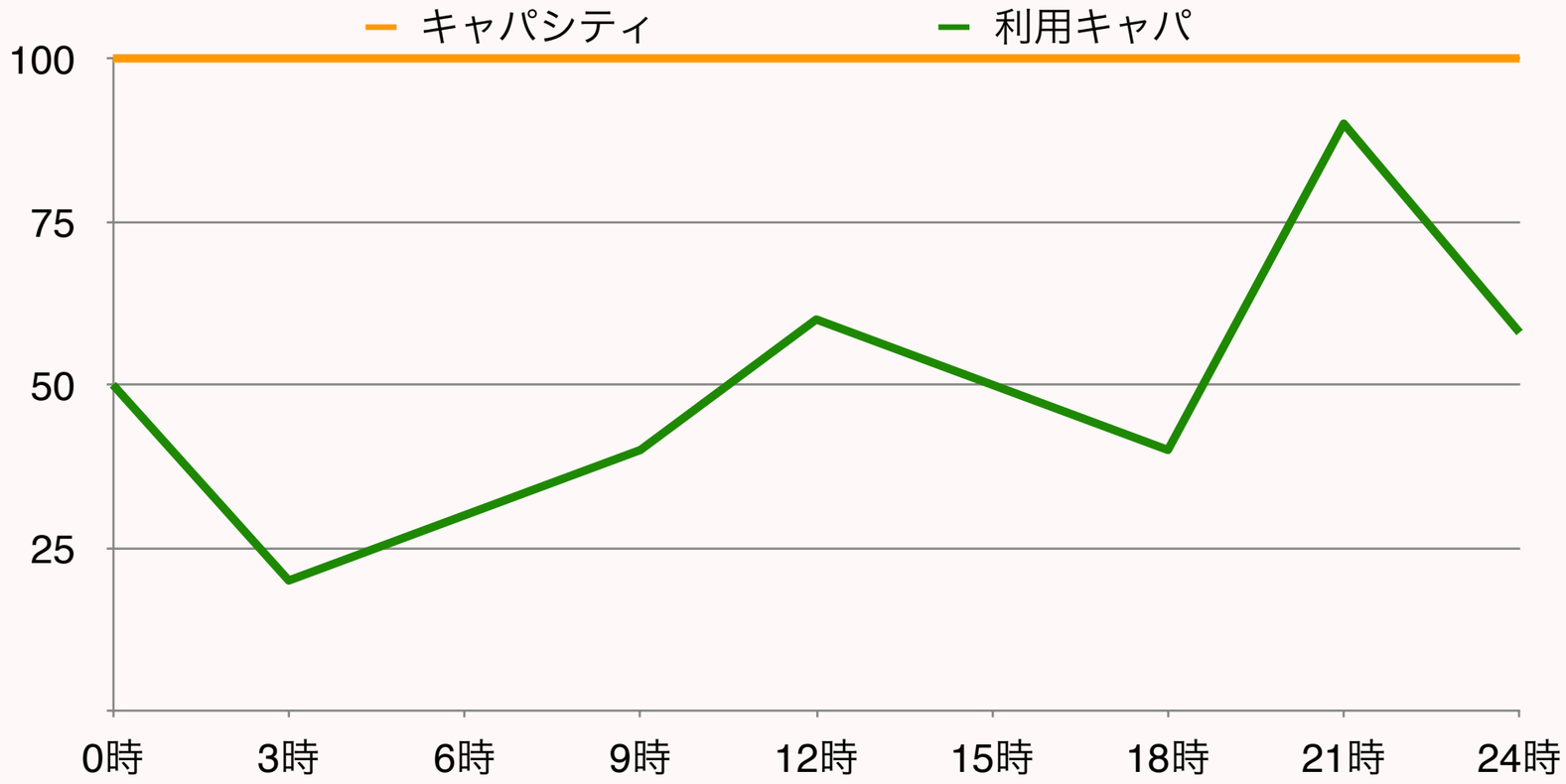
Fargate運用で感じていた課題

- オートスケーリングが追いつかない
  - せっかくFargate化してオートスケール出来るようになった
  - ピーク時間にレイテンシ違反アラートが出始めた(毎晩ヒヤヒヤ)
  - 見るとオートスケール発動してるが初動に追いついていない..
  - CPU60%トリガの50s程の起動バッファだとちょっと遅い..
- 費用がEC2時代と比べると高止まりしていた
  - タスク数を減らして、SavingsPlansを使ってもまだ高い
  - 同じサービスを運用するなら費用は少ないのが正義ですよね

せっかくFargateを使っているのだから
もっとうまくスケーリングが出来ないものか

キャパシティプランニングに対する考え方の変化

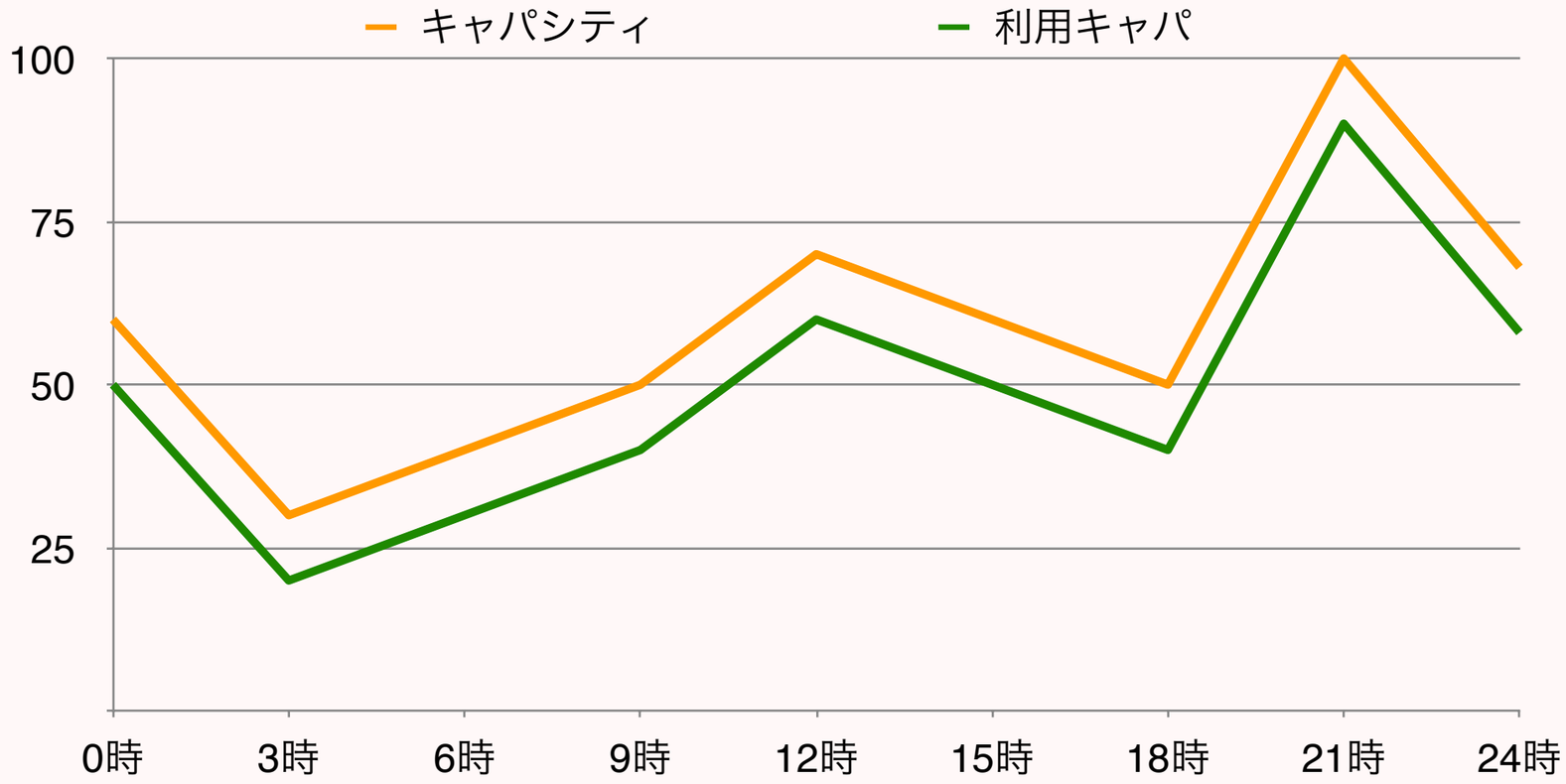
- ピークトラフィックを安全に捌けるキャパシティを用意
  - 安全なサービス運用のために余分な費用を支払う
  - EC2バックエンドでもFargate使ってもこの考えでサービス運用していた



mamari

キャパシティプランニングに対する考え方の変化

- 従量課金を活かした動的なキャパシティコントロールへ
  - ユーザが多い時には多くのリソースを構え少なくなったらリソースを最小化
  - 必要なリソースを必要な時だけ用意しその対価を支払うのがクラウドの原点



mamari

キャパシティプランニングに対する考え方の変化

- 従量課金を活かした動的なキャパシティコントロールへ
  - ユーザが多い時には多くのリソースを構え少なくなったらリソースを最小化
  - 必要なリソースを必要な時だけ用意しその対価を支払うのがクラウドの原点

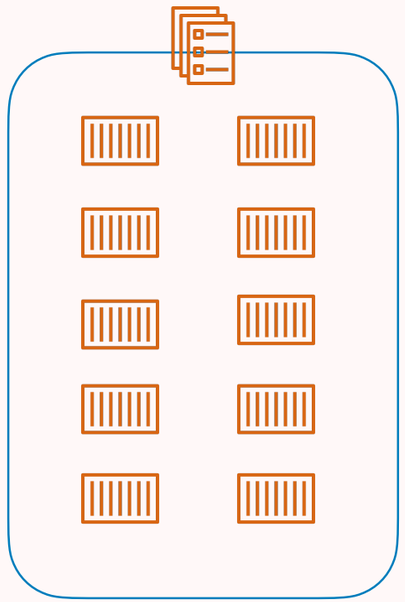
ECSで実現したい

mamari

ECS ターゲット追跡ServiceAutoScalling

- ECS側でタスク必要数を動的にコントロールしてくれる機能
  - ECS Serviceの平均CPU使用率
  - ECS Serviceの平均メモリ使用率
  - ECS Serviceに紐づくALBのタスクあたりのリクエスト数
- これまでもスパイク対策に使っていた
  - スケールアウト用に平均CPU使用率60%トリガで発動

平均CPU使用率40%を追跡するServiceAutoScalling挙動

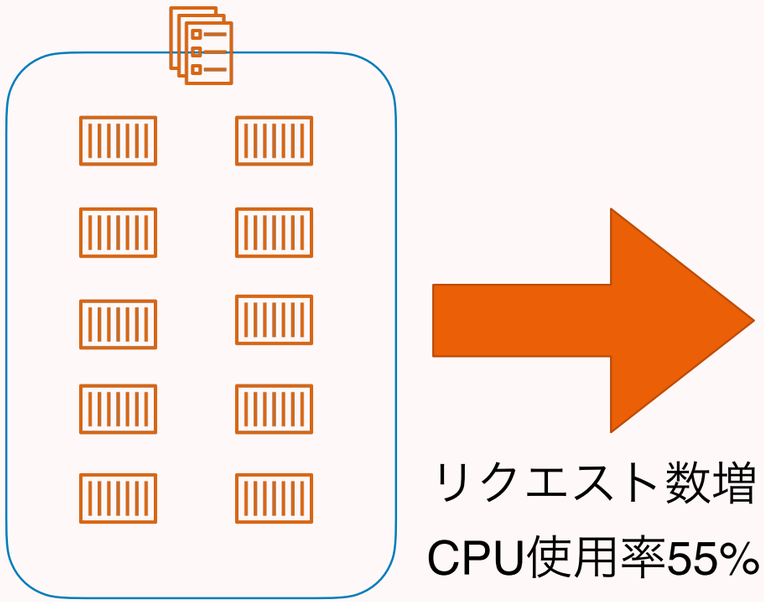


タスク必要数: 10

CPU使用率39%

mamari

平均CPU使用率40%を追跡するServiceAutoScalling挙動



リクエスト数増

CPU使用率55%

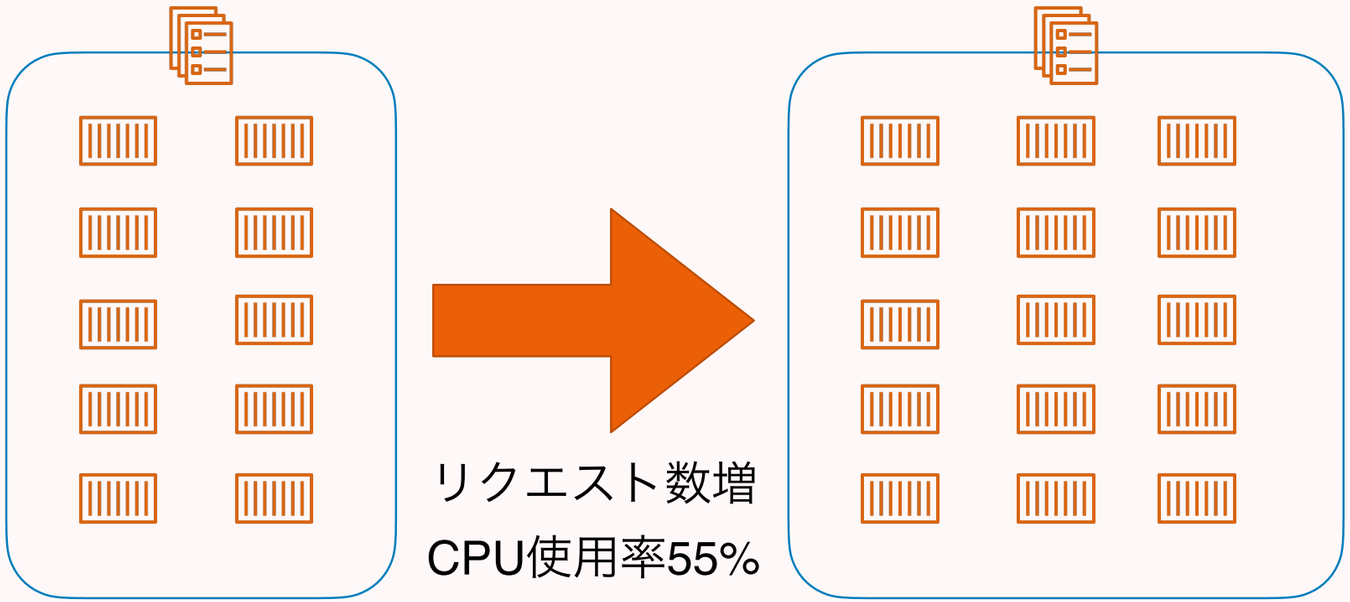
🔥🔥🔥🔥

タスク必要数: 10

CPU使用率39%

mamari

平均CPU使用率40%を追跡するServiceAutoScalling挙動



リクエスト数増

CPU使用率55%

🔥🔥🔥🔥

タスク必要数: 10

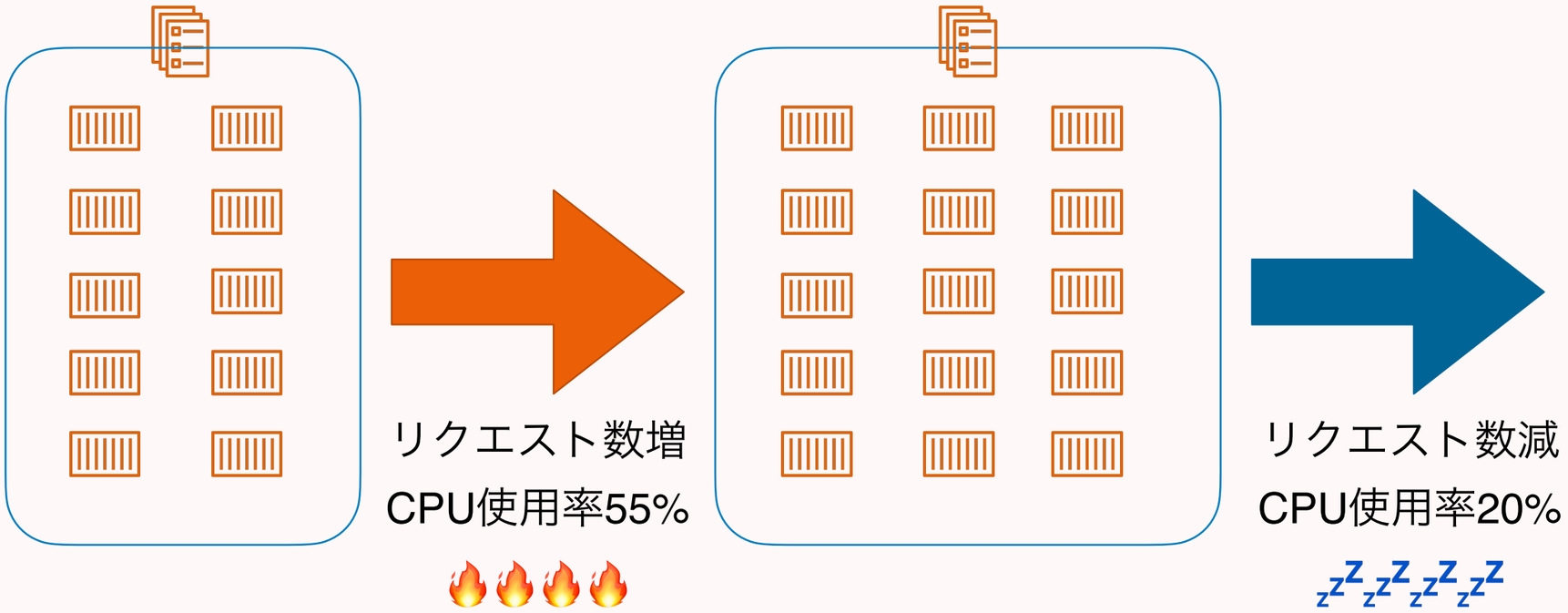
CPU使用率39%

タスク必要数: 15

CPU使用率38.5%

mamari

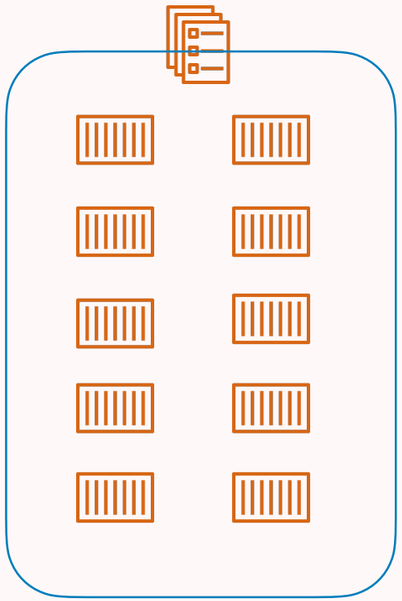
平均CPU使用率40%を追跡するServiceAutoScalling挙動



mamari

平均CPU使用率40%を追跡するServiceAutoScalling挙動

CPU使用率を指定値に収束するように
タスク数をコントロール



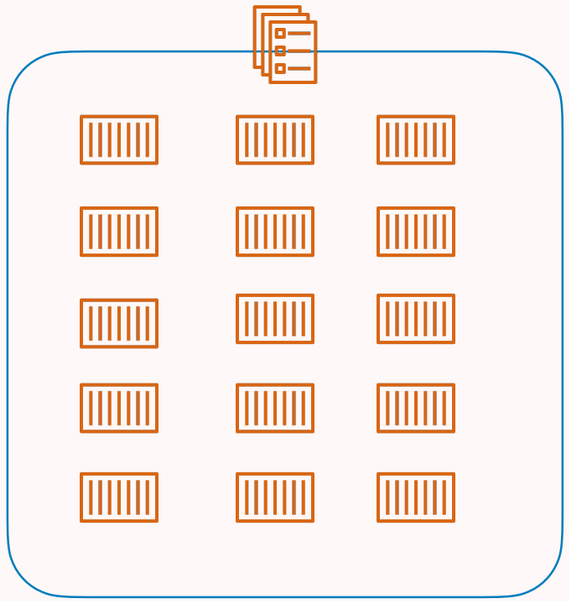
タスク必要数: 10

CPU使用率39%

リクエスト数増

CPU使用率55%

🔥🔥🔥🔥



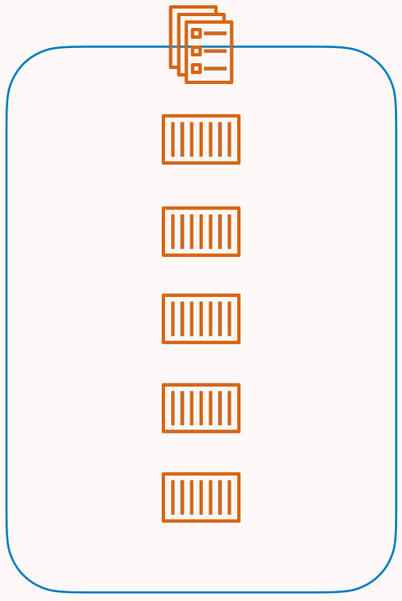
タスク必要数: 15

CPU使用率38.5%

リクエスト数減

CPU使用率20%

zzzzzzz



タスク必要数: 5

CPU使用率30%

※最低数までスケールイン

mamari

ECS ターゲット追跡ServiceAutoScallingの挙動

- しきい値を超えた時にスケールアウトする機能と捉えていたが

- **実は、しきい値に収束するためにスケールイン/アウトする機能だった**

ECS ターゲット追跡ServiceAutoScallingの挙動

- しきい値を超えた時にスケールアウトする機能と捉えていたが

- **実は、しきい値に収束するためにスケールイン/アウトする機能だった**
  →動的なキャパプラを実現出来る機能だという気づき

ECS ターゲット追跡ServiceAutoScallingの裏側の仕組み

- CloudWatchAlarmがセットされてそれを元にスケール判定
  - 【スケールアウト判定】 3分間連続しきい値違反
  - 【スケールイン判定】15分間連続しきい値-3%に収束
- スケールインまでしてくれれば費用の課題を解決出来る
  - 従量課金を最大に活かすために夜間は最低限のリソースでカバー

ECS ターゲット追跡ServiceAutoScallingの設定の勘所

- スケールインまで考える時はしきい値は少しゆるめにする
  - サービスの性質によるが、CPU50%以上だとスパイクに耐えれないかも
  - コネヒトでいうとだいぶゆるめでも費用削減にインパクトが出ている
- 導入時は負荷試験もしくはゆるいしきい値から挙動を確認

ECS ターゲット追跡ServiceAutoScallingの設定値

項目	内容	コネヒトでの設定値
ターゲット値	追跡したメトリクスでのしきい値 (平均CPU/平均メモリ/タスクあたりのreq数)	サービスの性質により 検証しながら適切な閾値を入れる
スケールアウト クールダウン期間	スケールアウトした後の待機時間 (この間は連続でスケールアウトは発動しない)	60s (負荷時は毎分スケールアウト)
スケールイン クールダウン期間	スケールインした後の待機時間 (この間は連続でスケールインは発動しない)	900s (ゆっくり縮退してほしいので)
スケールインの無効化	On/Offでスケールインを行うかどうか	今回の目的だとOff

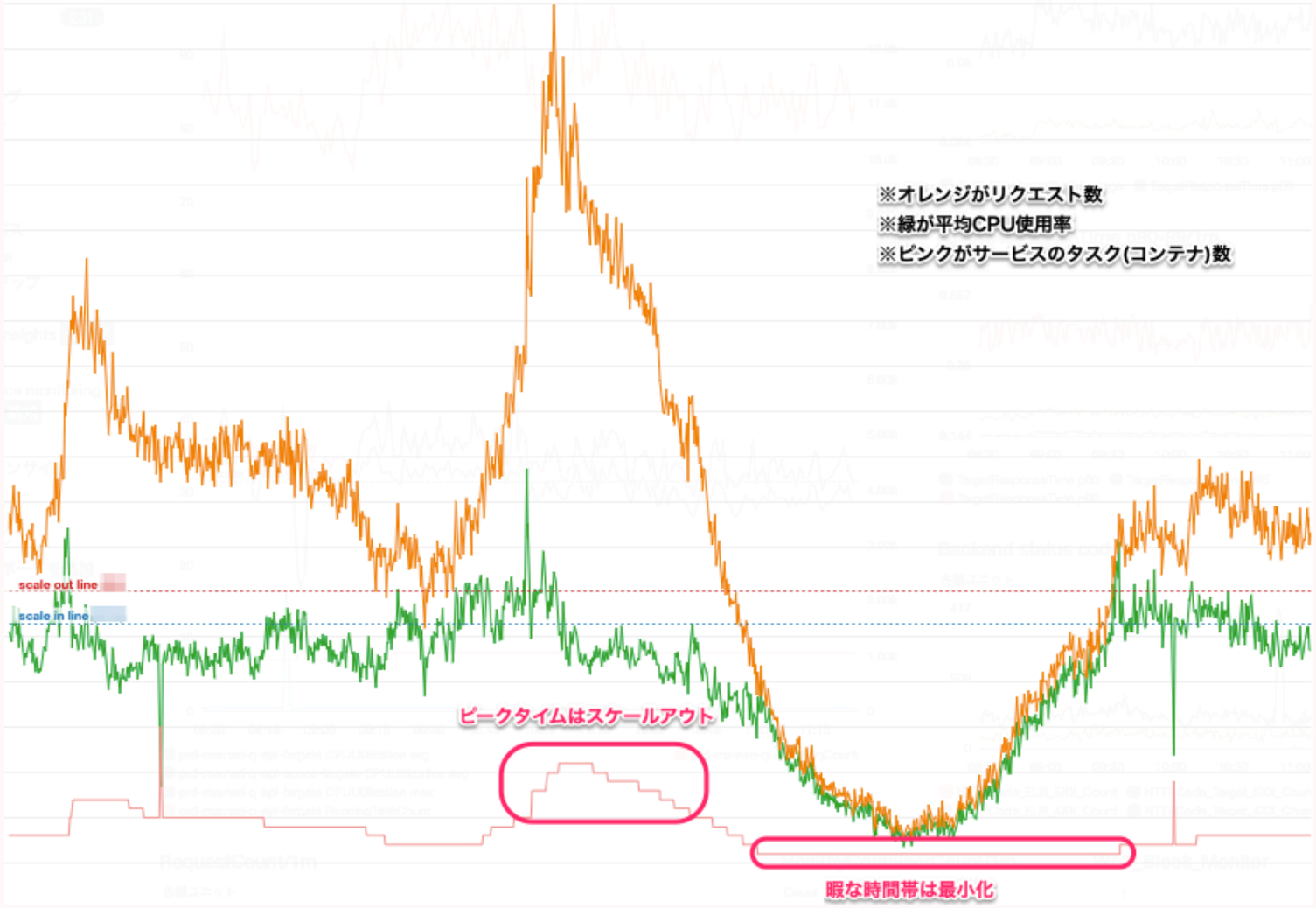
課題に対してどんな結果になったか?

(再掲)Fargate運用で感じていた課題

- **オートスケーリングが追いつかない**
  - ピーク時にレイテンシ違反アラートが出始めた(毎晩ヒヤヒヤ)
  - 見るとオートスケール発動してるが初動に追いついていない..
- **費用がEC2時代と比べると高止まりしていた**
  - タスク数を減らして、SavingsPlansを使ってもまだ高い
  - 同じサービスを運用するなら費用は少ないのが正義ですよね

ピーク前の余裕を持ったスケールアウトを実現

- ピーク時にたまに出ていたレイテンシアラートがなくなった
  - 事前に十分なタスクを構えるのでFargate起動が遅い影響を受けない



mamari

Fargateの費用が25%減

- 負荷に応じた動的なスケーリングで平均タスク起動数が1/4
  - ピーク時間に比べて暇な時間の方が多いのでタスク数の最適化に成功
  - 固定値時代よりも安定稼働かつコスト削減を実現

mamari

現在のECS×Fargate運用

- **Fargate化しているサービスはほぼメンテ不要で運用**
  - Fargateエージェントのv1.4対応はあるので0ではない
  - ピーク時間帯のスケールもアラートなく勝手にやってくれてる感じ
- **費用もFargate導入時に比べて大幅に削減**
  - 平均タスク稼働数が25%下がったので従量課金の恩恵をそのまま受ける
  - + SavingsPlansで22%割引の恩恵を受ける
  - + ステージングや一部のバッチではFargate Spotで大幅なコストカット

ECS×EC2のコンビも用途によって使っている

- **cron用途のコンテナはEC2バックエンドで動かしている**
  - cron的なスケジュール起動が必要なものは起動速度に利があるのでEC2
  - Dockerレイヤキャッシュを使えるのでpull速い=起動速い

まとめ

アーキテクチャ/項目	運用コスト	費用	ポイント
EC2バックエンド	△ 前述のEC2運用	◎ Reserved,SpotInstanceを利用	オートスケールが複雑なので、ピークトラフィック*数倍に耐えうるクラスタ(EC2)を用意
Fargate	◎	△ SavingsPlansを利用 タスク数減らしEC2バックエンドと同じくらいの費用感	RI,Spotと比較すると高いので、ピークトラフィックに耐えうるようタスク減らす+オートスケール(遅いという課題感)
Fargate × ターゲット追跡 ServiceAutoScalling	◎	◎	タスク数制御をECSに委任。 平均的なタスク数の削減に成功し費用圧縮

mamari

まとめ

アーキテクチャ/項目	運用コスト	費用	ポイント
EC2バックエンド	△ 前述のEC2運用	◎ Reserved,SpotInstanceを利用	オートスケールが複雑なので、ピークトラフィック*数倍に耐えうるクラスタ(EC2)を用意
Fargate	◎ タスク数はEC2バックエンドと同じくらいの費用感	△ SavingsPlansを利用	RI,Spotと比較すると高いので、ピークトラフィックに耐えうるようタスク減らすオートスケール(遅いという課題感)
Fargate × ターゲット追跡 ServiceAutoScalling	◎	◎	タスク数制御をECSに委任。 平均的なタスク数の削減に成功し費用圧縮

Fargateのメリットを活用して
運用負荷を極力低くコンテナ運用することが
ECSを選択する強みだと考えている

さいごに宣伝

コネヒトではエンジニア積極採用中です！

興味がある方、もっと話を聞いてみたいと思う方がいましたら

Wantedly経由もしくは

 @shnagai までご連絡ください

mamari

aws DEV DAY
ONLINE JAPAN

Thank you!

コネヒト株式会社
永井勝一郎(shnagai)

In Partnership with intel