『あんさんぶるスターズ！！Music』を支えるAmazon ECS
～人気ゲームの新作でのコンテナ化～
AWS Summit Online Japan

Happy Elements株式会社
インフラグループ グループリーダー
鷲見　啓志

Happy Elements
Make The World Happy!

用語統一

Amazon EC2 : Amazon Elastic Compute Cloud

Amazon ECS : Amazon Elastic Container Service

Amazon ECR : Amazon Elastic Container Registry

Amazon EKS : Amazon Elastic Kubernetes Service

Amazon RDS : Amazon Relational Database Service

Amazon S3 : Amazon Simple Storage Service

Aurora : Amazon Aurora

AWS : アマゾン ウェブ サービス

CloudFront : Amazon CloudFront

CloudWatch : Amazon CloudWatch

CloudWatch Logs : Amazon CloudWatch Logs

CodeBuild : AWS CodeBuild

CodeDeploy : AWS CodeDeploy

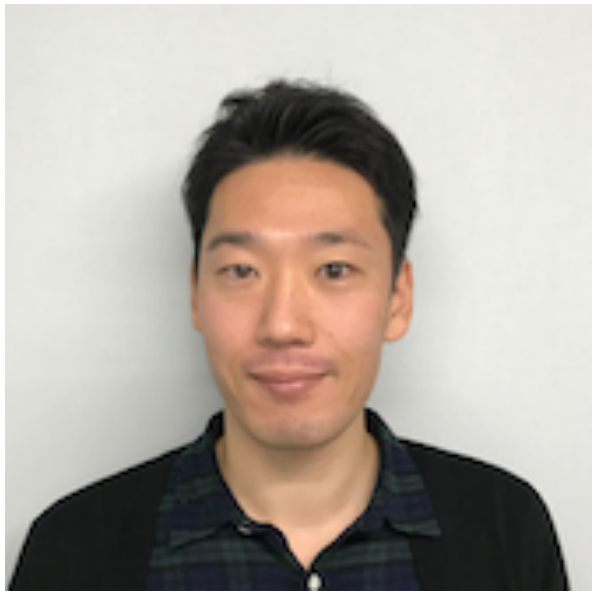
Fargate : AWS Fargate

Kinesis Data Firehose : Amazon Kinesis Data Firehose

自己紹介

自己紹介

- 鷲見啓志（すみひろし）
- 2019年1月より Happy Elements株式会社 に勤務
  - 京都のカカリアスタジオ所属
  - インフラグループのリーダー・インフラエンジニア
  - 複数のゲームタイトルのインフラの運用と導入支援
- 前職ではゲームディレクター・ゲームプロデューサー
- 趣味は自転車
  - 休日は北摂の山の中でもがいています
- AWS利用歴は10年(2010~)
- 好きなAWSサービス：Aurora



会社紹介

Happy Elements?

Happy Elements株式会社

北京を拠点にアジア圏でゲームサービスを展開する「Happy Elementsグループ」の日本子会社。日本市場に向けたオリジナルゲームの開発・運営を行う。

設立　　：2010年 4月

代表取締役：新井元基

所在地　　：京都府京都市下京区立売西町82 京都恒和ビル6F

従業員　　：社員数：228名 学生アルバイト12名（2020年6月現在）

平均年齢　：30歳(2020年6月現在)

資本金　　：5,000万円

Happy Elements
Make The World Happy!

提供タイトル

あんさんぶるスターズ！！Basic

あんさんぶるスターズ！！Music

メルクストーリア – 癒術士と鐘の音色 –

ラストピリオド – 巡りあう螺旋の物語 –

本日お話すること

本日お話すること

2020年3月15日にリリースした『あんさんぶるスターズ！！Music』。
本作のAmazon ECSでのコンテナ化にフォーカスを当て、経緯・技術選定・直面した課題・リリース時の様子についてご紹介いたします。

本日お話すること

1. コンテナの導入以前に抱えていた課題について
2. コンテナの導入と技術選定について
3. コンテナの導入で直面した課題と解決方法について
4. リリース時の様子とその後

本日お話すること

想定する聴講者

- コンテナやAmazon ECSの導入を…
  - これから検討したいと考えている技術選定の担当者・責任者
  - これから始めるインフラエンジニア
  - 現在進めているインフラエンジニア

『あんさんぶるスターズ！！』について

既存タイトルと新タイトルについて

既存タイトル
「あんさんぶるスターズ！」
(2015/04~2020/03)

大型アップデート

新章「あんさんぶるスターズ！！」
(2020/03~)

「あんさんぶるスターズ！！Basic」
既存タイトルのリニューアル
従来のプレイスタイル

「あんさんぶるスターズ！！Music」
新作ゲーム
新しいリズムゲームスタイル

共通のストーリー／イラストで遊べる！

あんさんぶるスターズ！！Music

3Dキャラクターのライブシーンをバックにリズムゲームをプレイ！自分の好きなキャラクターへの入れ替え、衣装の着せ替えも可能！豊富な楽曲とハイクオリティな3DCGを駆使したMVをお楽しみいただけます！

あんさんぶるスターズ！！Music

2020年3月15日リリース

事前登録者数：70万人

リリース当日のDL数：140万件

2020年4月10日にTwitter社のブログで公開された記事では、世界中で今年もっともツイートされたゲームのTop10にいれていただきました。

出典：Twitter上のゲームに関する会話の盛り上がり (Twitter社ブログ)
https://blog.twitter.com/ja_jp/topics/company/2020/April_gaming_on_Twitter.html

『あんさんぶるスターズ！！Music』の技術構成

あんさんぶるスターズ！！Musicの技術構成

クライアント

unity

C#

アセット配信/Web

CloudFront Amazon S3

サーバーサイド(API)

RAILS

PUMA NGINX

CloudFront Amazon ECS Amazon EC2 Aurora Amazon ElastiCache

docker

Amazon Linux 2
(ECS Optimized)

Application
Load Balancer

その他ツール

Jenkins

GitHub

New Relic.

DATADOG

Terraform

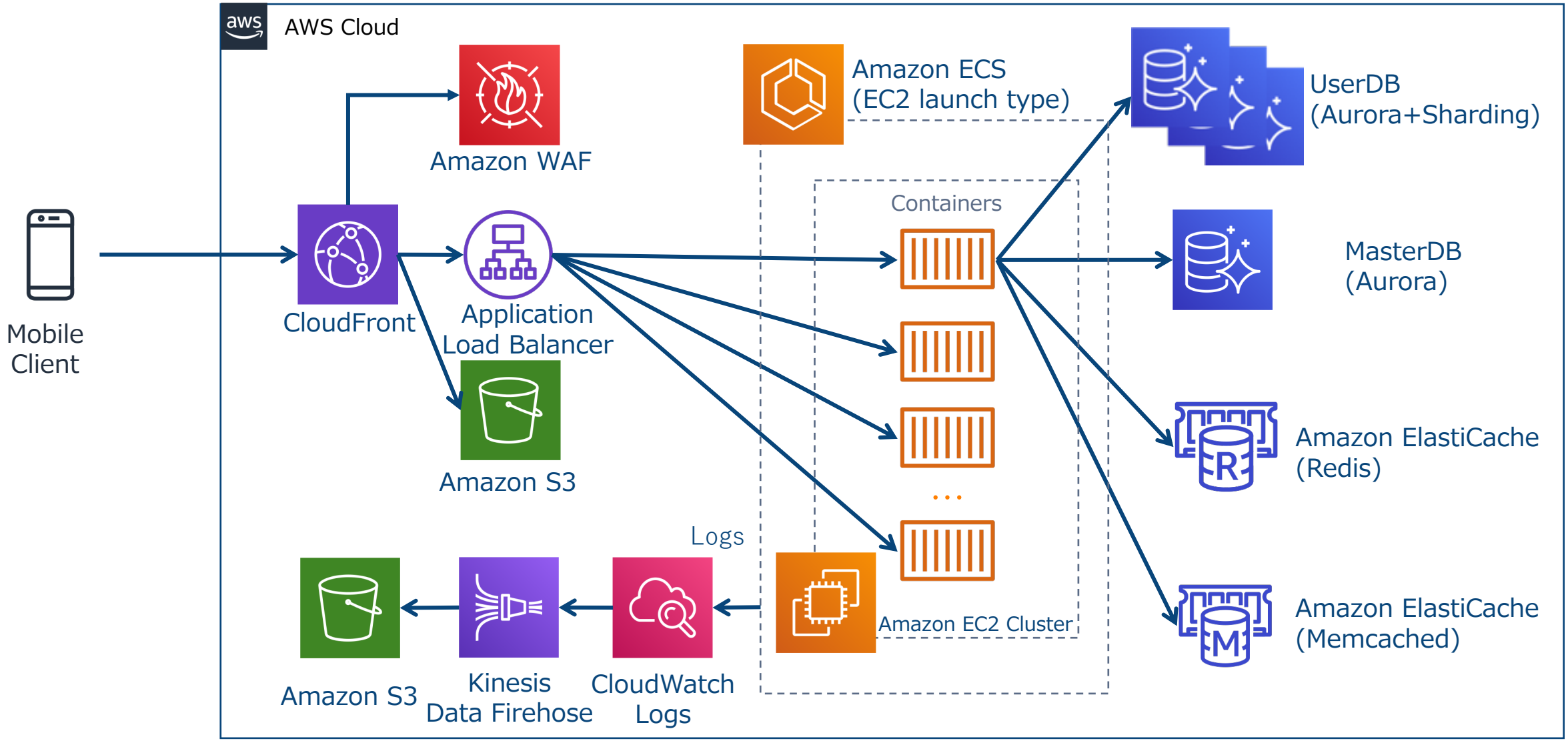
CodeBuild

CodeDeploy

Amazon ECR

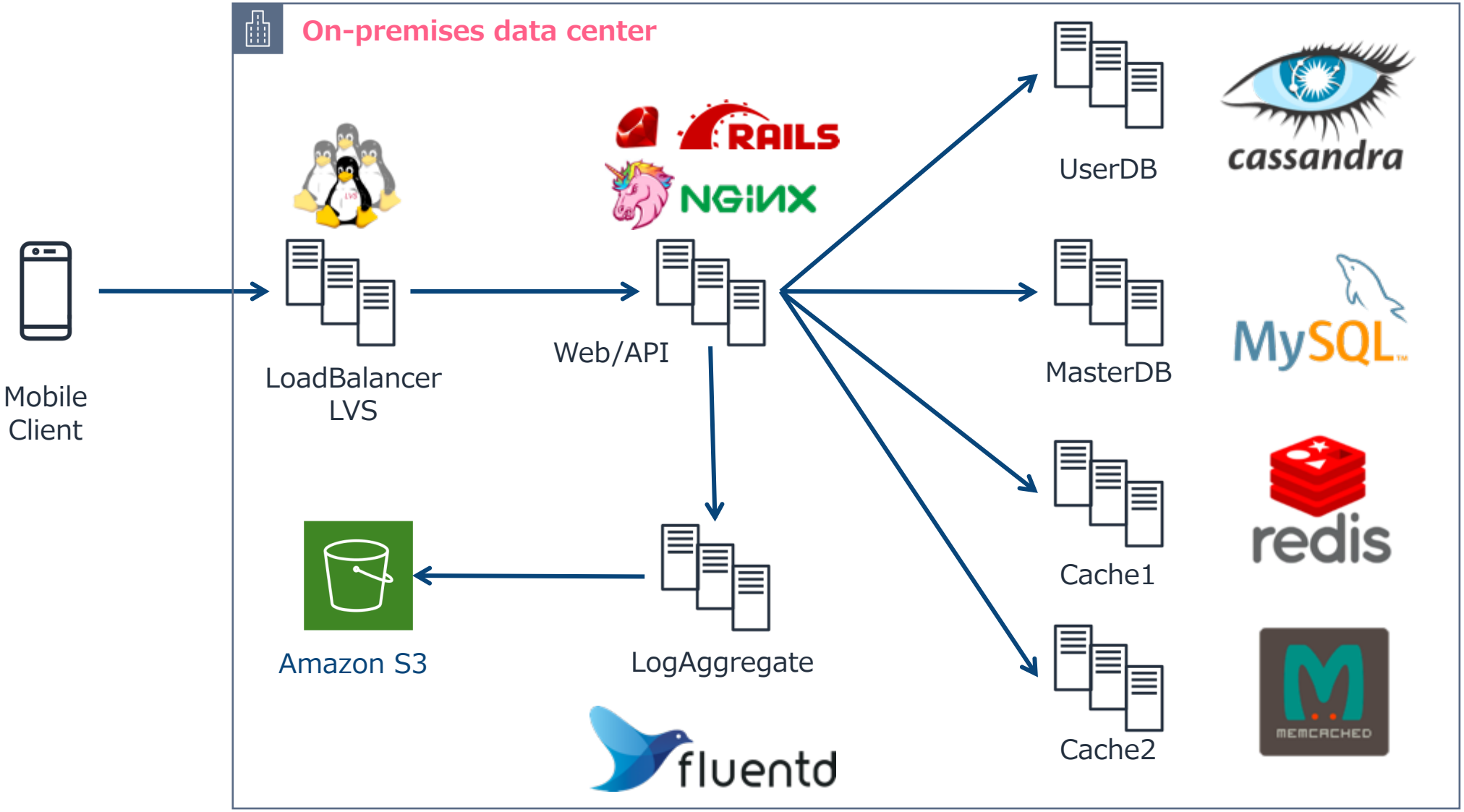
CloudWatch Logs

あんさんぶるスターズ！！Musicのアーキテクチャ



『あんさんぶるスターズ！』時代の状況と課題

『あんさんぶるスターズ！』時代のアーキテクチャ



『あんさんぶるスターズ！』時代の課題

2019年時点でサービスを提供しているゲームタイトルのAPIサーバーについては、オンプレミスのデータセンター上で稼働していました。

課題は大きく３つ

1. スケールしない環境・リソースの無駄使い
2. 技術的負債
3. 時間のかかる環境構築

1. スケールしない環境・リソースの無駄遣い

オンプレミスのデータセンターで物理サーバーが動いている環境

サーバーの増減をすぐには行うことができない

→想定外の突発的なスパイクに対応できない

アクセス集中時に合わせてサーバーを調達

→アクセスが集中するのはゲーム内イベントの開始時と終了時のみ

→集中時以外のサーバーリソースが無駄遣いになる

2. 技術的負債

ユーザーファーストを重視する社風による技術的負債の蓄積

- ユーザーに楽しんでもらうことを重視し、ゲームへのコンテンツ・機能の追加を最優先にした結果、インフラ・ミドルウェアのメンテナンス・新しい要素技術の導入などは後回しに。
- 後回しになったはいいが、いつまでたっても返済に着手できない
- 技術的負債を返す時間を確保できないまま負債が蓄積していった

3. 時間のかかる環境構築

開発環境の構築に時間がかかるうえに、他のタイトルのヘルプを行う場合はローカル環境に複数のミドルウェアのバージョンが同居することになり、環境の切り替えを意識しないといけなくなる。

- 現状、開発を開始できるまで半日～1日かかる
  - GitHubからのソースコードのDL
  - Ruby、Railsのバージョン合わせ
  - Unityのバージョン合わせ
  - Cassandra,MySQL,Redis,Memcachedのインストール、バージョン合わせ
  - 開発環境用のデータのロード
  - etc.

なぜコンテナ化したのか？

コンテナ化を選択した理由

1. 環境構築と管理の容易さ
2. 可搬性の確保と環境差異の排除
3. 技術的負債の返済

環境構築と管理の容易さ

- 環境構築の容易さ
  - Dockerをインストールして数コマンドで構築完了
- 環境管理の容易さ
  - 人やPCによる環境差異がなくなる
  - ローカル環境でミドルウェアが複数バージョン必要となるケースでも、必要なタイミングのみコンテナを起動するだけとなり、ローカル環境の管理が容易に
- Infrastructure as Codeの実現
  - プロビジョンからアプリケーションまで1ファイルで設定できる

可搬性・環境差異の排除

- 可搬性
  - Dockerコンテナが動く環境ならどこでも動く（Linuxコンテナ）
  - もちろん他のクラウドベンダーでも、オンプレミスでも
- 環境間の差異の排除
  - ローカル/開発/ステージング/本番といった環境毎のインフラ設定の差異がなくなり、環境依存の不具合が発生しづらくなる
  - どの環境でも動く一貫性が保証される

技術的負債の返済

- 新技術採用のチャンス
  - 今までの技術的負債を返済して一歩前に進みたい
  - 今からAmazon EC2での従来のオンプレミス環境と同様の使い方では、またすぐに負債がたまっていく未来がみえてしまう

フルマネージドサービスの活用と一歩進んでコンテナの採用へ

あんさんぶるスターズ！！Music におけるコンテナ技術

Amazon ECSとAmazon EKS

Amazon Elastic Container Service(Amazon ECS)

- AWSが開発・運用しているAWSネイティブなコンテナオーケストレーションサービス

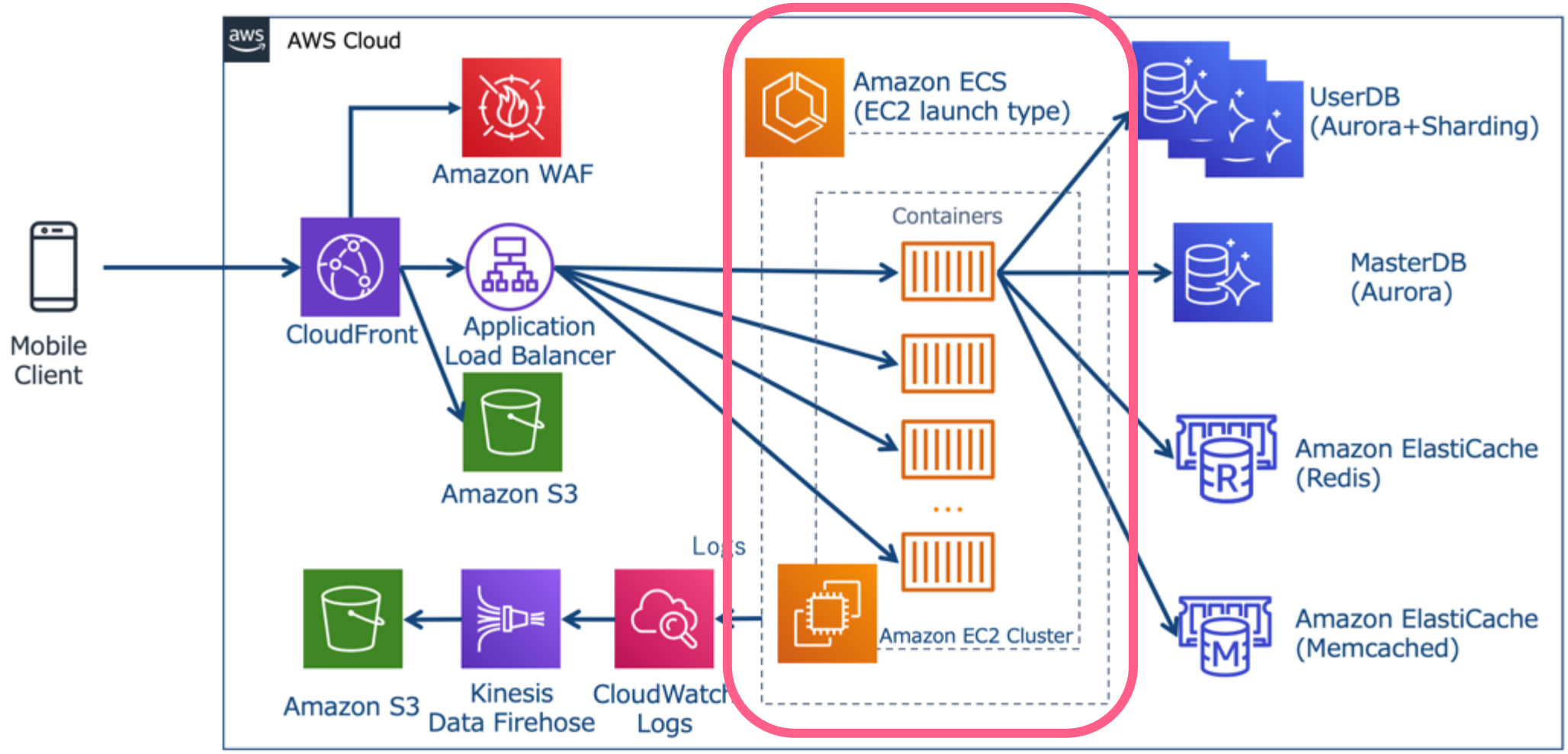
Amazon Elastic Kubernetes Service(Amazon EKS)

- OSSのコンテナオーケストレーションツールのデファクトと言われているKubernetesをAWSでマネージドサービスとして提供しているサービス

コントロールプレーンはどちらもAWSが管理

※コンテナオーケストレーションツールについてはSA濵さんのセッションで詳しく説明されます！

あんさんぶるスターズ！！Musicのアーキテクチャ



「あんさんぶるスターズ！！Music」ではAmazon ECSを採用

Amazon ECSを選んだ理由

Amazon EKS(Kubernetes)とくらべて

運用管理・学習教育コストが抑えられる

Amazon ECSを選んだ理由

Amazon ECSもやや複雑ではあるが、

Amazon EKSはさらに複雑

そもそもベースとなるKubernetesがかなり複雑

運用にはKubernetesに対する深い理解が必要

Amazon ECSはAmazon EKSと比べるとかなりシンプル

Amazon ECSを選んだ理由

Amazon EKSを採用するのであれば

Kubernetesに詳しいエンジニアが複数人必要

Amazon ECSでも学習することは必要だが、

今いるエンジニアが学ぶことで対応可能と判断

運用管理・学習教育コストが抑えられる

よりシンプルなAmazon ECSを採用

EC2起動タイプ
と
Fargate起動タイプ

Amazon ECSの起動タイプ

EC2起動タイプ

- EC2起動タイプでは、管理しているEC2インスタンスのクラスターで、コンテナ化されたアプリケーションを実行できます。

Fargate起動タイプ

- バックエンドインフラストラクチャをプロビジョニングおよび管理することなく、コンテナ化されたアプリケーションを実行できます。

EC2 起動タイプ

スマートフォンゲームの運用では

瞬時にスケールできるスケール性能が求められる

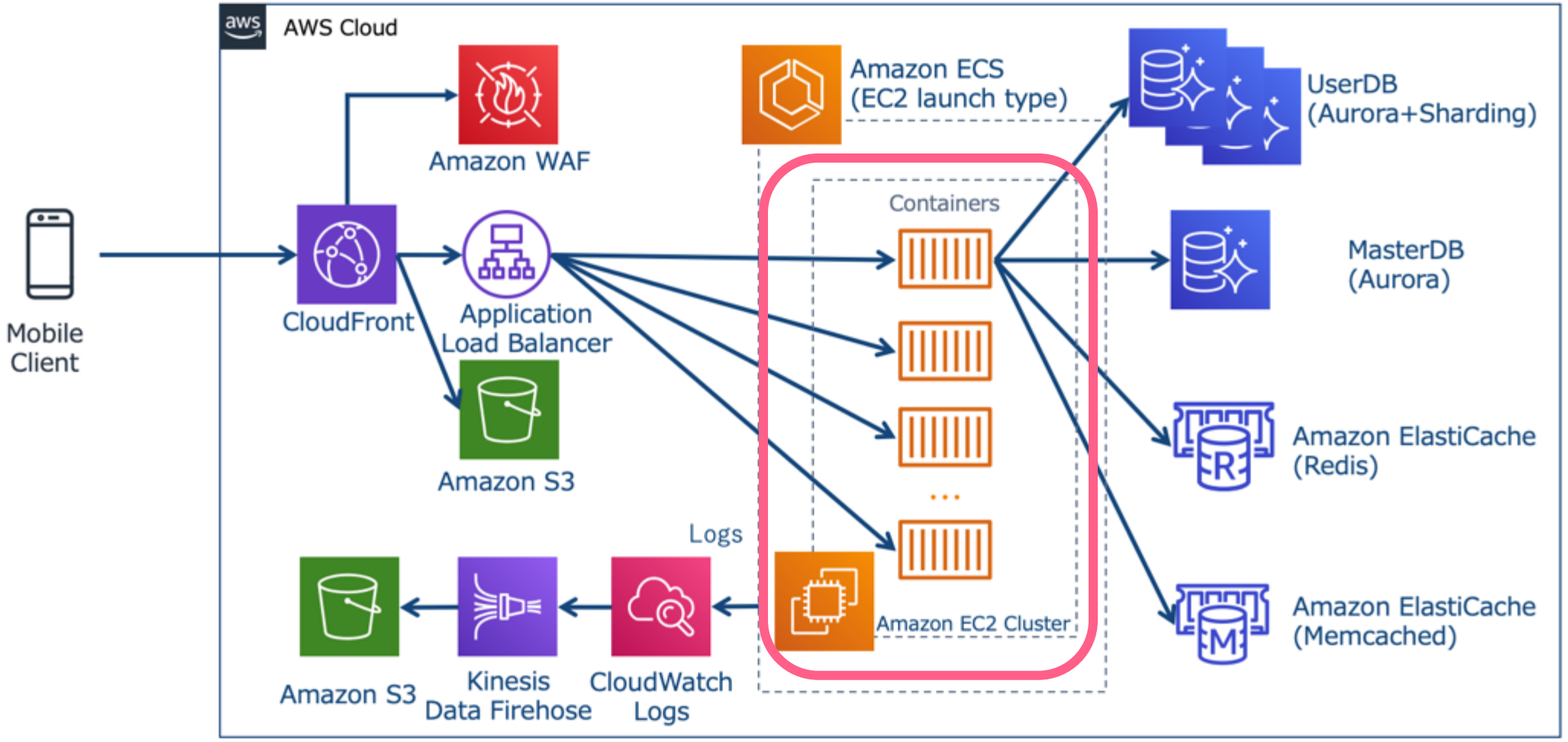
イメージキャシュを有効活用でき、

スケール速度を出すことができる

EC2起動タイプの方が特性に合致すると判断

※イメージキャッシュなどの詳細についてはCotainer Nightの登壇資料を御覧ください
https://speakerdeck.com/hiroshisumi/sumatohuongemudefalsekontenahuo-yong-container-night-guan-xi-kontenashi-li-deng-tan

あんさんぶるスターズ！！Musicのアーキテクチャ



「あんさんぶるスターズ！！Music」ではEC2起動タイプを採用

コンテナ導入での課題

コンテナ導入で出た課題

1. コンテナ・Docker・Amazon ECSの教育
2. ビルド＆デプロイ
3. キャパシティ設計

1. コンテナ・Docker・Amazon ECSの教育

開発メンバーにコンテナ・Docker・Amazon ECSの概念や使い方を理解してもらわなければならない。

とはいえ、資料を準備する時間がない。。。

1. コンテナ・Docker・Amazon ECSの教育

AWSジャパンのSAの方を講師にお招きし「コンテナDojo」というタイトルで、座学とハンズオンの社内セミナーを実施しました。

※SAの濵さん、藤原さんありがとうございました！

2. ビルド＆デプロイ

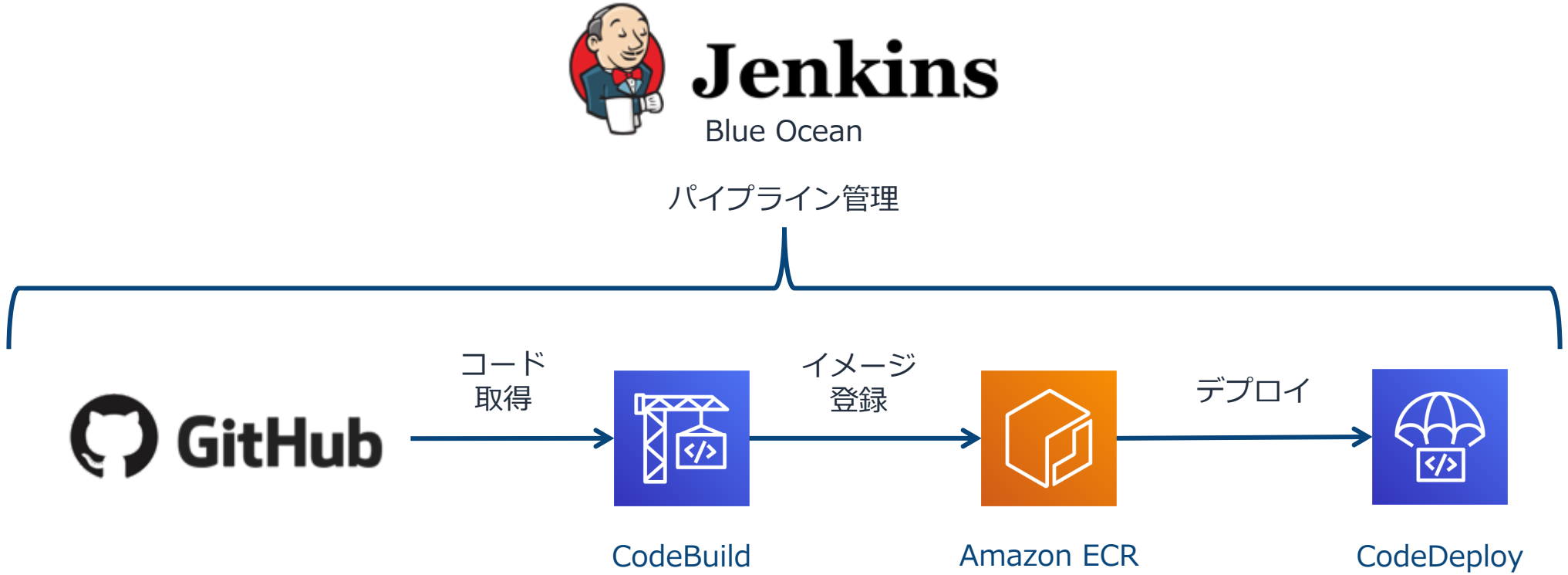
今まではコードのデプロイにCapistranoを利用していました

今後はコンテナイメージのデプロイになるので、

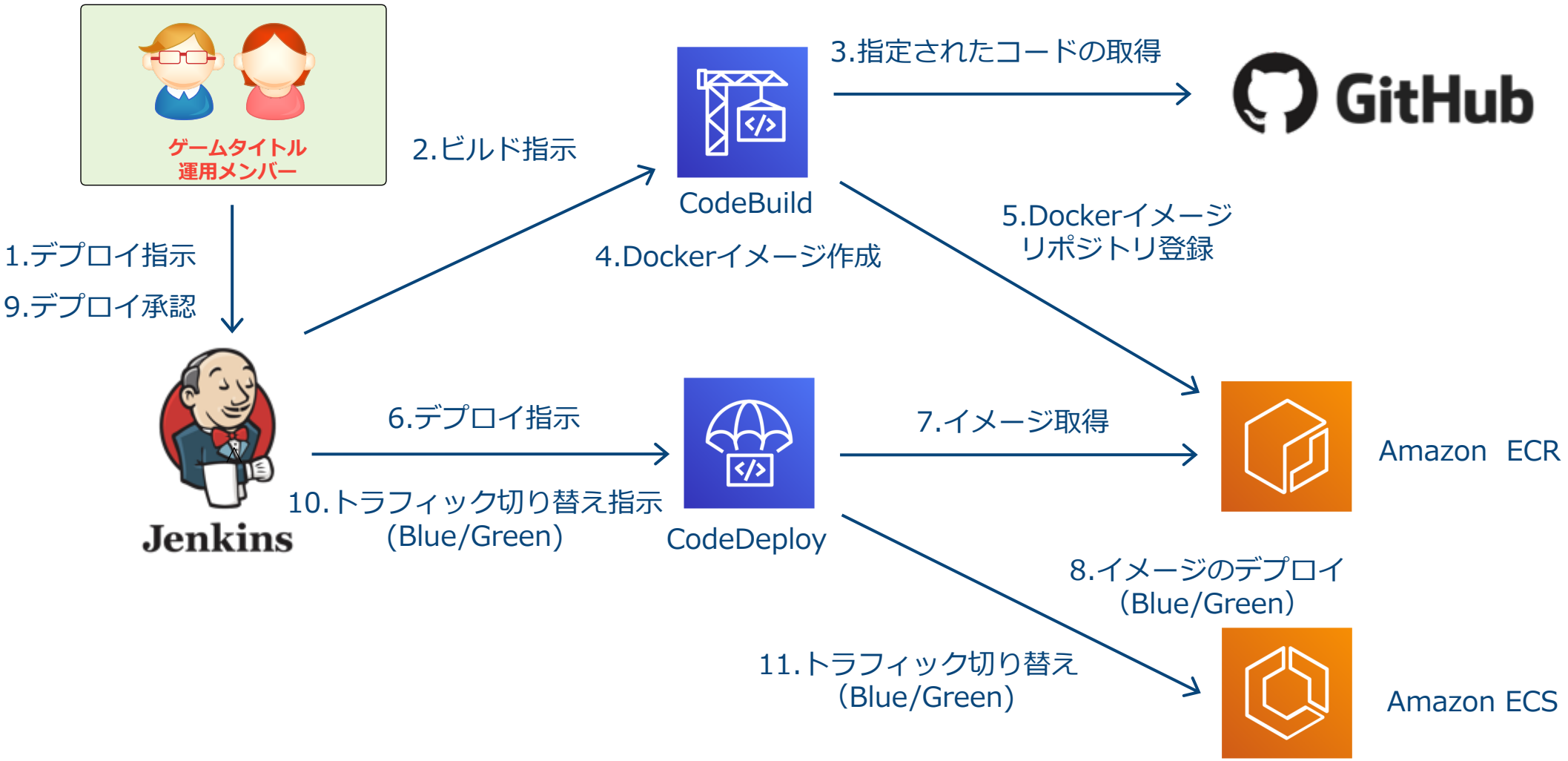
Capistranoに代わるデプロイ方法を検討しなければならない

2. ビルド＆デプロイ

いろいろ試した結果、この構成で運用しています



2. ビルド＆デプロイ

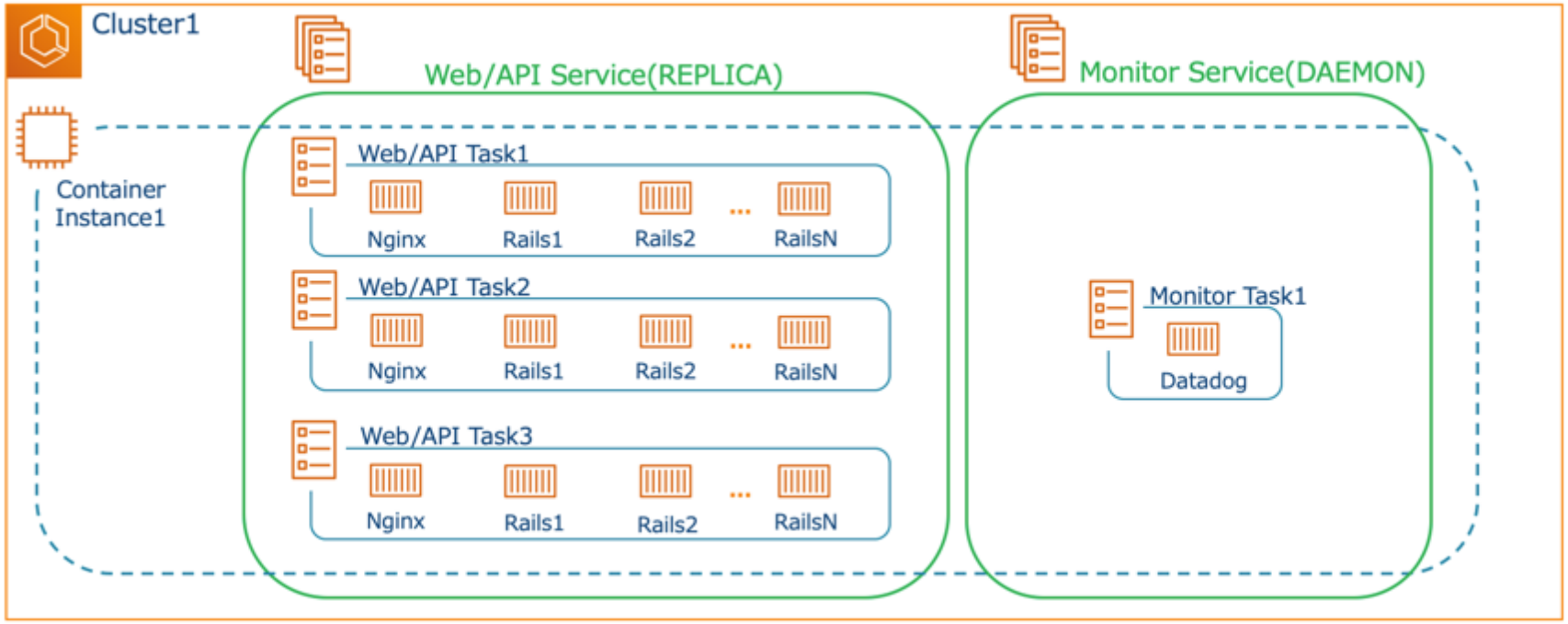


3. キャパシティ設計

Amazon ECSとEC2起動タイプを選択した場合、Amazon EC2を含め各種リソースのキャパシティを検討する必要がある

インスタンスのインスタンスファミリー・サイズ・台数をどうするか？

インスタンスあたりのタスク数をいくつにするか？



タスクあたりのコンテナ数をいくつにするか？

コンテナあたりのCPU・メモリの割当をいくつにするか？

3. キャパシティ設計

ワークロードによってもリソースの使用状況が異なるため、

たったひとつの正解というものは存在しない。

インタンスファミリー・サイズ・タスク数・コンテナ数・コンテナへのCPUとメモリの割当などいろいろ組み合わせて単体の負荷試験を実施し、パフォーマンスを測定。

結果として出てきたパフォーマンス数値を比較し最適解を導き出す。

負荷試験とキャパシティ設計

負荷試験とキャパシティ設計

負荷試験ツール

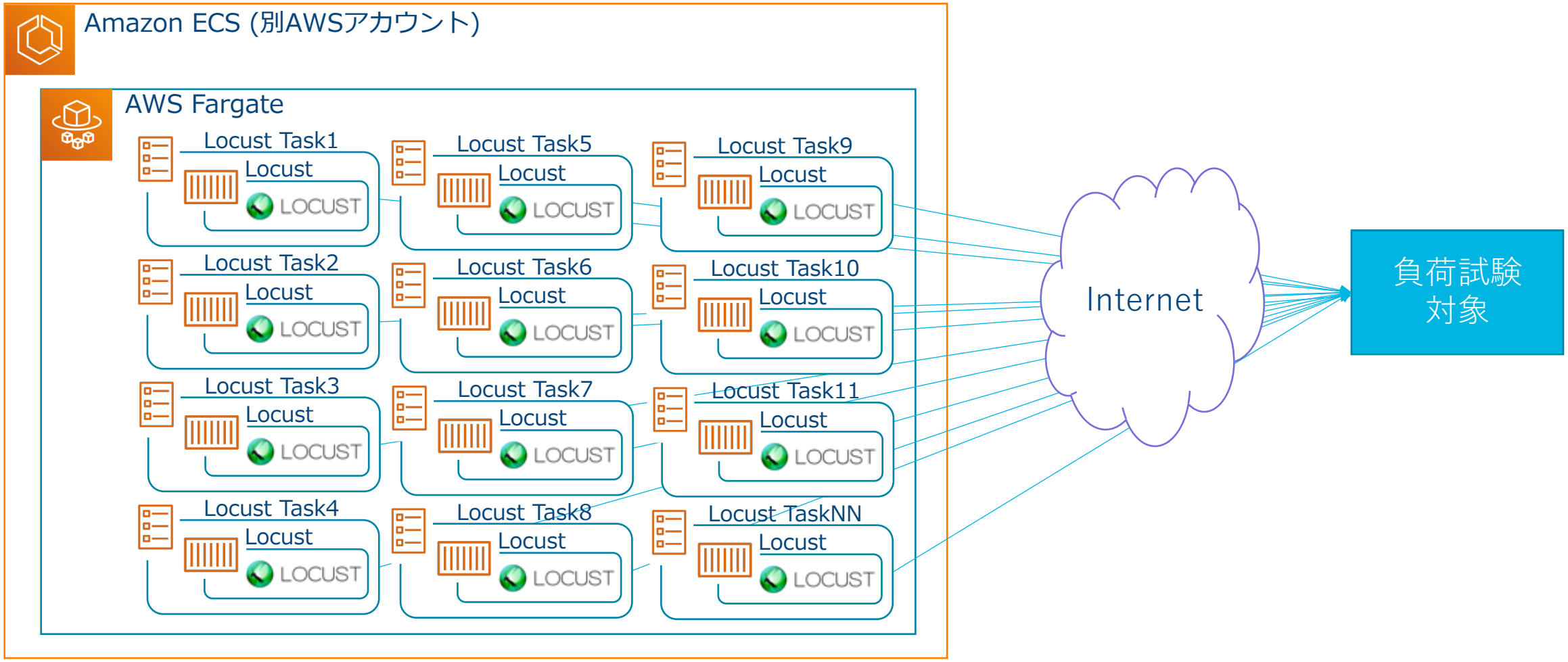
Locustを利用

- Pythonで作られた負荷試験ツール
- Pythonのコードでテストシナリオを書ける

負荷試験をかける側の環境

- Amazon ECS(Fargate起動タイプ)
  - コンテナイメージを事前に作成しておく
  - 多くのタスクを同時並列実行して負荷をかける
  - 別AWSアカウントからインターネット経由で負荷をかける

負荷試験とキャパシティ設計

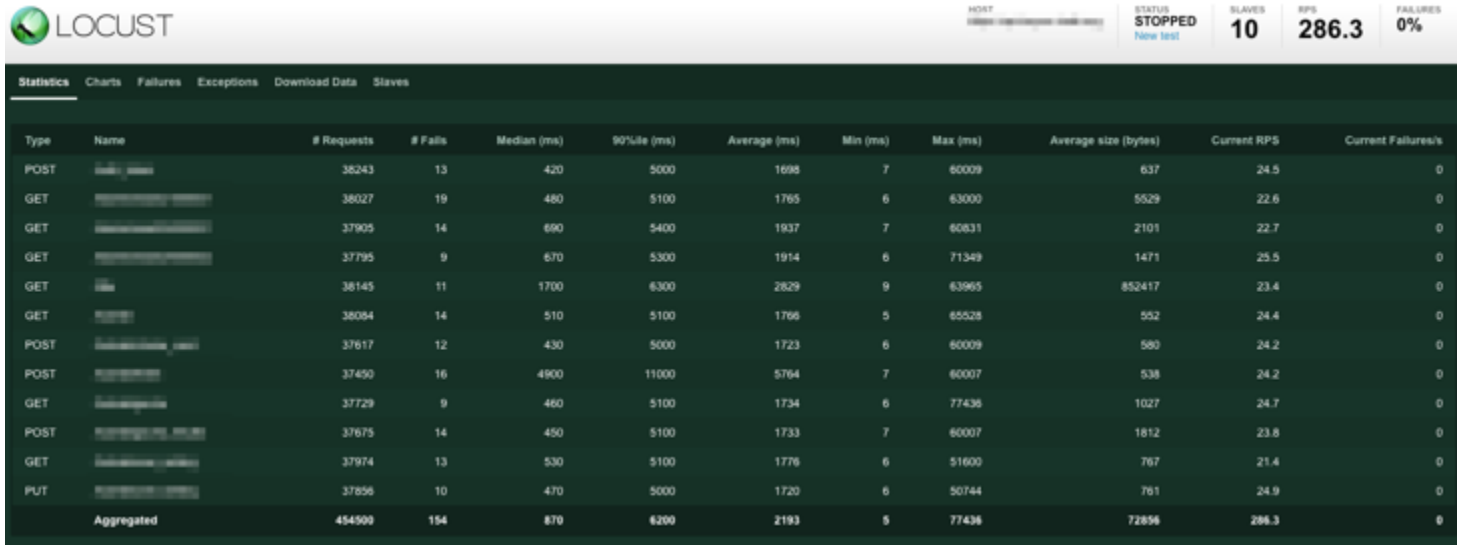


負荷試験とキャパシティ設計

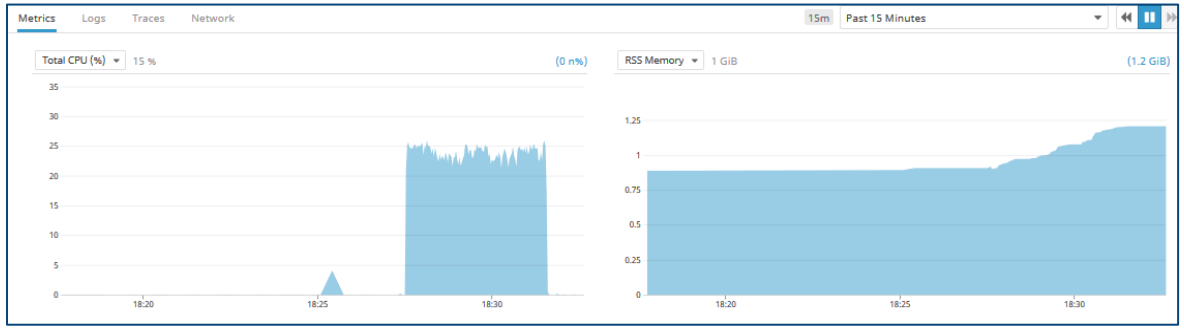
単体負荷試験

- 最小構成での機能単位の負荷試験
- パフォーマンスの測定
  - キャパシティ設計のための測定
    - インスタンスタイプ・サイズ・タスク数・コンテナ数・CPUとメモリの割当など組み合わせを変えて、組み合わせごとのパフォーマンスを測定。最適な組み合わせを探る

1. 負荷試験とキャパシティ設計



Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
POST	[redacted]	38243	13	420	5000	1698	7	60009	637	24.5	0
GET	[redacted]	38027	19	480	5100	1765	6	63000	5529	22.6	0
GET	[redacted]	37905	14	690	5400	1937	7	60831	2101	22.7	0
GET	[redacted]	37795	9	670	5300	1914	6	71349	1471	25.5	0
GET	[redacted]	38145	11	1700	6300	2829	9	63965	852417	23.4	0
GET	[redacted]	38084	14	510	5100	1766	5	65528	552	24.4	0
POST	[redacted]	37617	12	430	5000	1723	6	60009	580	24.2	0
POST	[redacted]	37450	16	4900	11000	5764	7	60007	538	24.2	0
GET	[redacted]	37729	9	460	5100	1734	6	77436	1027	24.7	0
POST	[redacted]	37675	14	450	5100	1733	7	60007	1812	23.8	0
GET	[redacted]	37974	13	530	5100	1776	6	51600	767	21.4	0
PUT	[redacted]	37856	10	470	5000	1720	6	50744	761	24.9	0
	Aggregated	454500	154	870	6200	2193	5	77436	72856	286.3	0



Locustで負荷をかけ、メトリクスをCloudWatchやDatadog、New Relicで確認

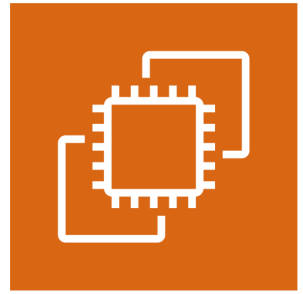
複数の組み合わせで実施し、パフォーマンスを測定、比較検討する。

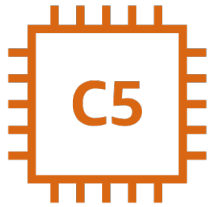
リリース時の構成

リリース時の構成

APIサーバー:Amazon ECS(EC2起動タイプ)

- EC2インスタンス
  - c5.4xlarge × 81
    - インスタンス1台あたり5タスク
      - APIサービスのタスク × 4
      - Datadogエージェントのタスク × 1
    - APIサービス1タスクあたり6コンテナ
      - Rails × 5
      - Nginx × 1
  - Railsのコンテナは合計1620個稼働





リリース時の構成

DBサーバー(ユーザーデータ)

- Amazon RDS
  - Aurora(MySQL)
    - db.r5.4xlarge × 20
    - 10シャード(マスター+レプリカ)

DBサーバー(マスタデータ)

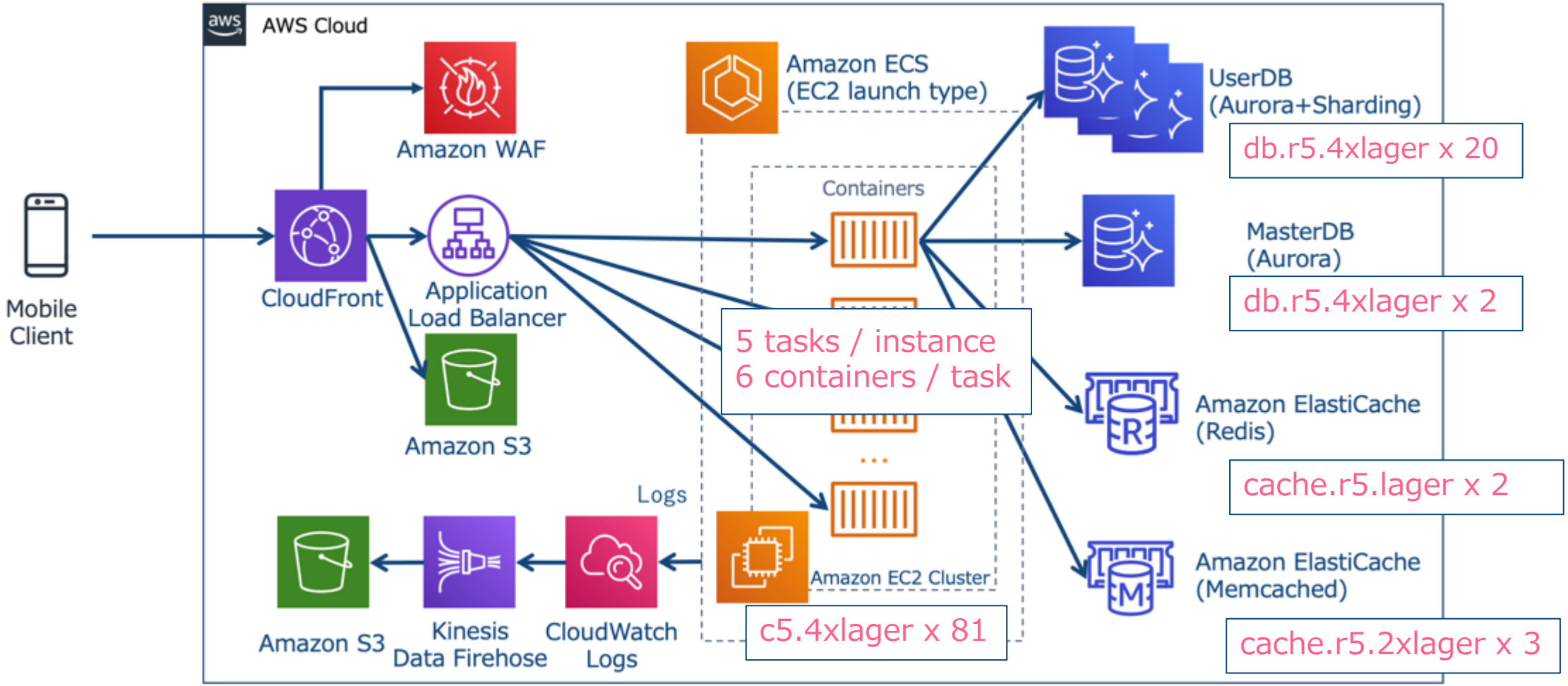
- Amazon RDS
  - Aurora(MySQL)
    - db.r5.4xlarge × 2
    - マスター+レプリカ

リリース時の構成

キャッシュサーバー

- Amazon ElastiCache
  - Memcached
    - cache.r5.2xlarge × 3
  - Redis
    - cache.r5.large × 2

あんさんぶるスターズ！！Musicのアーキテクチャ



※リリース後はアクセス数の状況に合わせて台数・タイプの調整を実施しています。

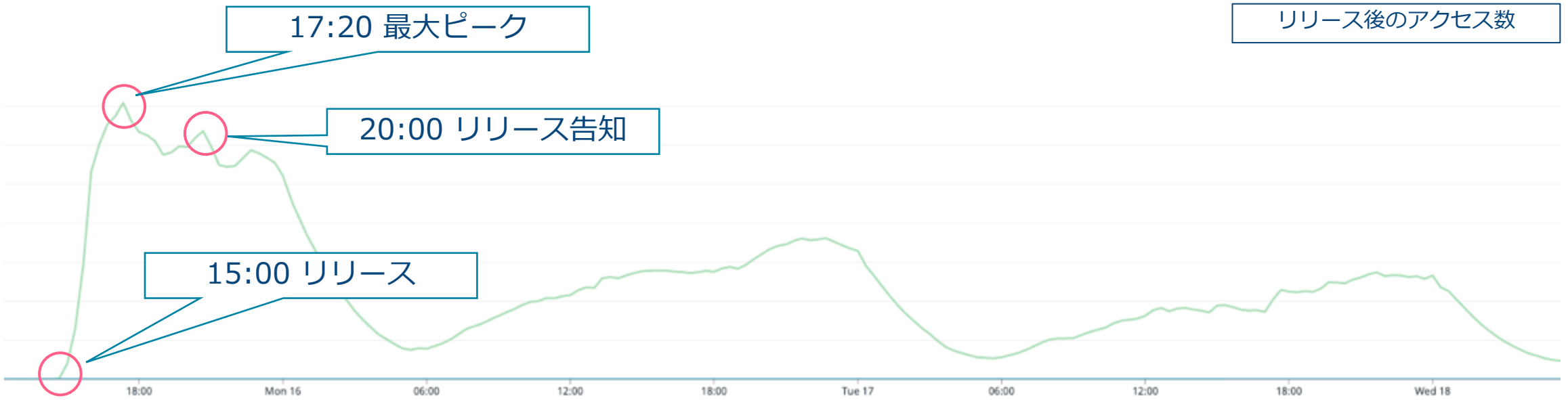
リリース時とその後

リリース時の様子

2020年3月15日 15:00 にリリース

大きな不具合が出ていないことを確認できてから告知したかったため、
ユーザーへの告知は20:00に実施

リリース時の様子

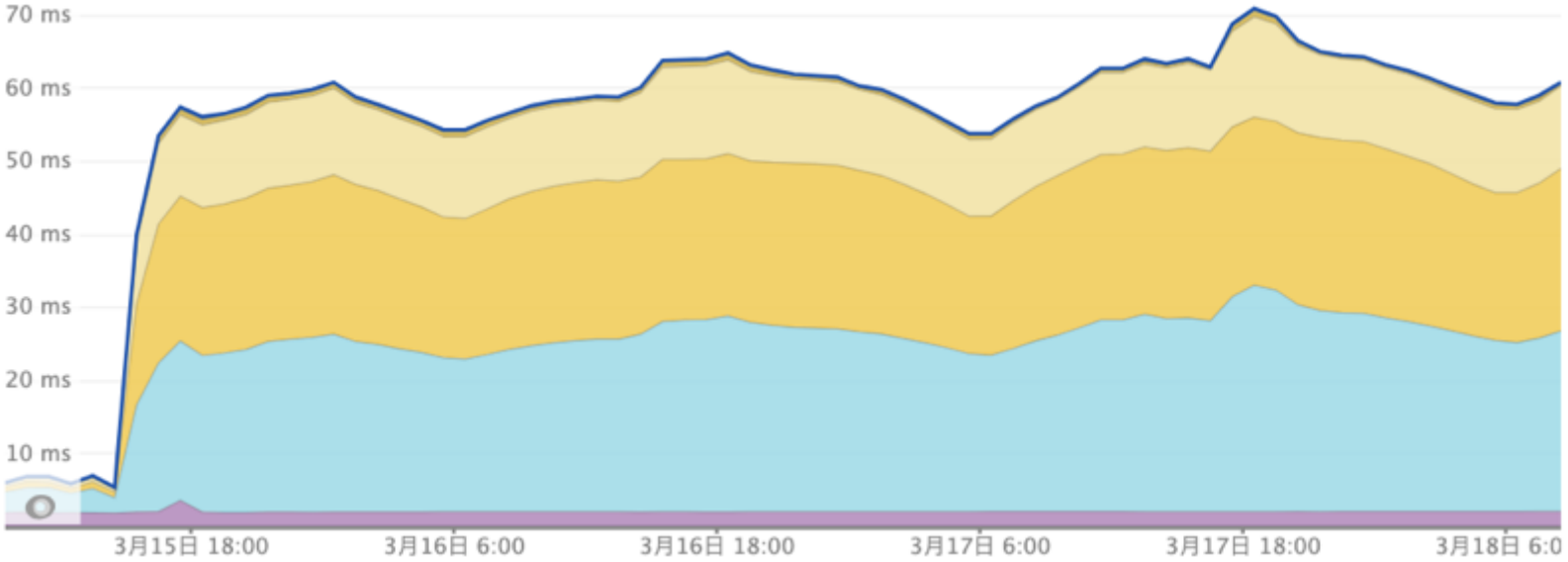


リリース日を事前に告知していたため、待ち構えていてくださったユーザーが多く、プレイできるようになったことがSNSで拡散していき、告知前にアクセスの最大ピークを迎えました。

リリース後は日を追うごとに徐々に落ち着いていきました。

リリース時の様子

リリース後のNew Relicの画面



リリース直後は概ね70ms以内にサーバー側の処理を完了

コンテナに移行して特に遅いと感じることは今のところない

リリース時の様子

リリース後、サービスに影響を与える大きな不具合はなし

「あんさんぶるスターズ！！Music」のリリース作業は、

ほぼノートラブルで終えることができました。

コンテナ化の効果
今後の要望

コンテナ化の効果

効果

- 環境構築が容易になった
- 環境間の差異がないこと
- オートヒーリング（オーケストレーションによる）

コンテナ化の効果

コンテナ導入時の課題

- 運用方法をコンテナに合わせる変更する必要がある
  - migrationなどのタスクの実行方法の変更
  - 定時バッチ処理の実行方法
  - ログの運用方法
  - デプロイ方法の変更

コンテナ化の効能

今後を見据えたAWSへの改善要望

- ログ(CloudWatch Logs)の改善
  - UI改善
    - 対象が見つけづらい
    - 見づらい
  - コスト改善
    - 値段高い
- Fargateの改善
  - Fargateでもイメージキャッシュが使えるようになってほしい

まとめ

まとめ

「あんさんぶるスターズ！！Music」でのコンテナ利用

- 環境構築の容易さ・可搬性と技術的負債の返済を目的
- サービスはAmazon ECSを利用
- Fargateの利用は見送りEC2起動タイプを選択
- 導入時の課題は教育・ビルド＆デプロイ・キャパシティ設計
- 導入後、パフォーマンスの問題や大きな障害は発生していない
- コンテナ化の効果
  - 環境構築の容易さ
  - 環境間の差異がなくなる
  - オートヒーリング（オーケストレーション）

まとめ

シンプルで学習コストの低いAmazon ECSはコンテナ導入の第一歩として最適なサービスの一つだと思います。

導入時に検討が必要な点・変更しなければならない点も少なくはありませんが、得られるメリットも大きいです。

今回の「あんさんぶるスターズ！！Music」の導入事例が
みなさんのAmazon ECS導入の一助となりますと幸いです！

最後に

Happy Elements株式会社では、
一緒に働いてくれるメンバーを募集中です！

積極採用中の職種

- インフラエンジニア
- データエンジニア
- プログラマー（アプリ）
- プログラマー（サーバーサイド）
- etc...

熱狂的に愛されるコンテンツを、
私達と一緒につくりたい**熱意**のある方、
ぜひご応募お待ちしております！
（詳しい内容については、採用サイトをご確認ください）
https://www.happyelements.co.jp/recruit/