I2-05

コンテナ移行ってこんなに大変？ ～「家族アルバム みてね」を支えるインフラの裏側～

清水 勲
SRE
株式会社ミクシィ/Vantageスタジオ みてね事業部 開発グループ

aws SUMMIT

自己紹介

清水 勲 （しみず いさお）

所属
株式会社ミクシィ Vantageスタジオ
みてね事業部 開発グループ SREチーム
経歴

- SIerで受託開発、プロダクト開発を経験後、ミクシィに入社
- SNSの運用、モンスターストライクのSREを経て、現在は「家族アルバム みてね」のSRE
- AWSとの最初の出会いは2009年頃
- 2014年 AWS Summit Tokyo 登壇
  - 「徹底討論！ゲームプラットフォーマーのクラウド活用法」
- 2018年 AWS Dev Day Tokyo LT大会 ベストスピーカー賞
- 2018年 AWS re:Invent 初参加（今年も参加します！）

aws CERTIFIED
Solutions Architect - Associate

aws SUMMIT

本日お持ち帰りいただきたいこと

- ✓ コンテナ移行によってシステムを一新する際に気をつけたいこと
- ✓ コンテナ移行によって解決されること
- ✓ コンテナ移行の難しいところ、工夫するポイント
- ✓ Amazon EKS(Kubernetes)を選ぶメリット

アジェンダ

① 「家族アルバム みてね」の紹介
② みてねのアーキテクチャ
③ コンテナ移行によって解決すること
④ コンテナサービスの選択
⑤ コンテナ環境のデザインと構築
⑥ Amazon EKS移行の状況
⑦ まとめ

アジェンダ

① 「家族アルバム みてね」の紹介
② みてねのアーキテクチャと課題
③ コンテナ移行によって解決すること
④ コンテナサービスの選択
⑤ コンテナ環境のデザインと構築
⑥ Amazon EKS移行の状況
⑦ まとめ

家族アルバム みてね

- 子どもの写真・動画を、無料・無制限に共有できるアプリ
- 子どもが、生まれると…
  - こんなに写真って撮るの？
  - 家族が撮った写真・動画も全部みたい
  - 親・親戚と、どうやって共有しよう？
- その子の人生をまるごと残せる家族アルバムを！
- 世界中の家族の “こころのインフラ” をつくる
- 2015年サービスリリース
- **現在、国内外の利用者数 500万人以上**
- **App Store レビュー　4.7**
- **Google Play ストア レビュー　4.7**

aws SUMMIT

海外での受賞歴

- **2019 THE WEBBY AWARDS 受賞**
  - “国際デジタル芸術科学アカデミー (IADAS) によって毎年主催され、優れたインターネットに贈られる賞”
  - プレスリリース https://mixi.co.jp/press/2019/0405/3792/index.html
  - https://www.webbyawards.com/winners/2019/apps-mobile-and-voice/apps-mobile-features/best-user-experience/familyalbum/

- **2019 NATIONAL Parenting Product(NAPPA) AWARDS 受賞**
  - 家族にフォーカスした賞。おもちゃや家族向けの商品に特化している。
  - https://www.nappaawards.com/product/familyalbum/

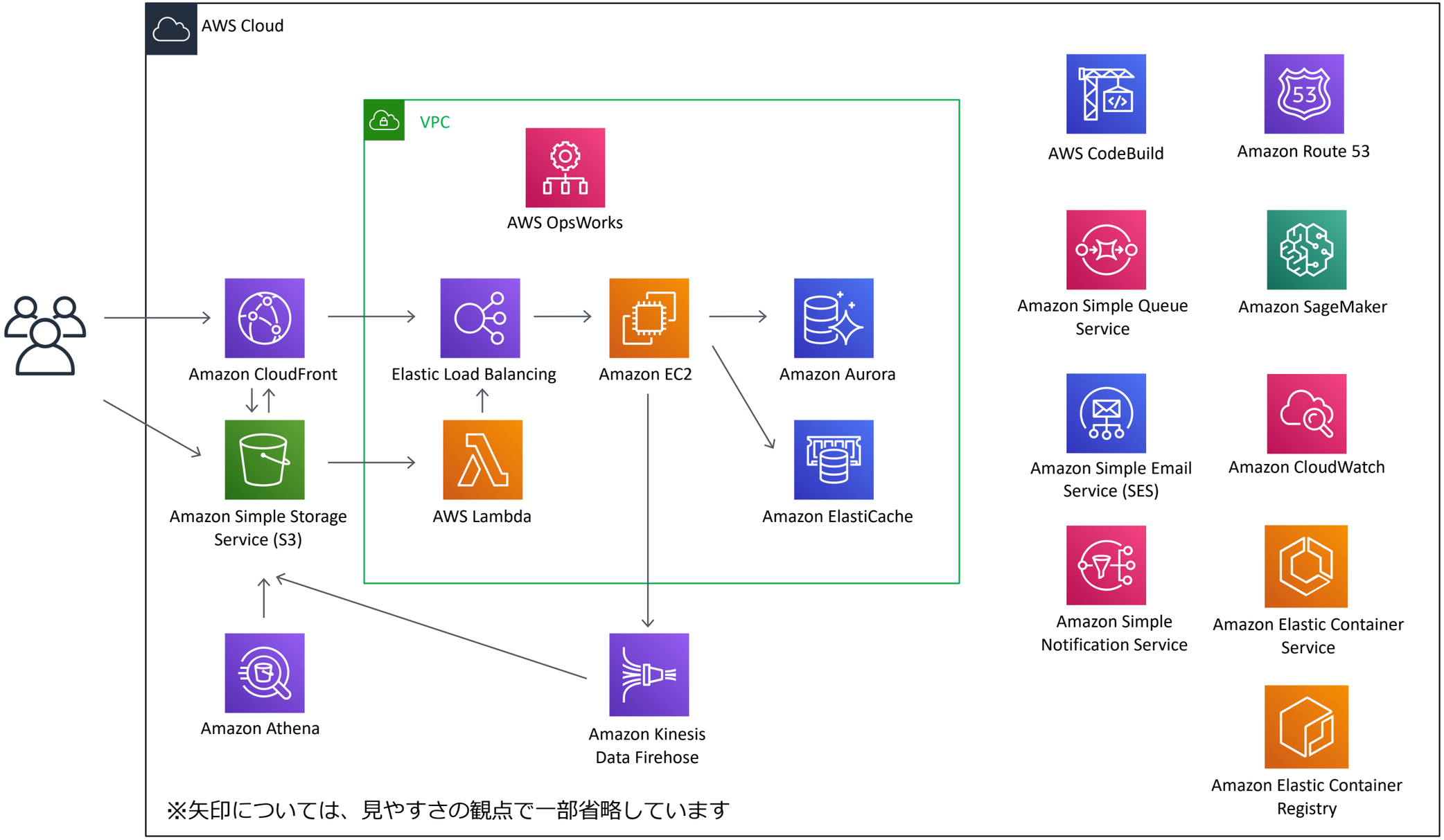
みてねプレミアム

- 2019年4月リリース 🎉
- 月額課金型の有料プラン
  - 月額 480円
- ブラウザから写真や動画のアップロードが可能に
- 1秒動画の毎月配信
  - 通常は3ヶ月毎
- 動画アップロードの時間制限を延長
  - 3分から10分へ

みてねをお使いの方、ぜひお試しください！

アジェンダ

① 「家族アルバム みてね」の紹介
② みてねのアーキテクチャと課題
③ コンテナ移行によって解決すること
④ コンテナサービスの選択
⑤ コンテナ環境のデザインと構築
⑥ Amazon EKS移行の状況
⑦ まとめ



みてねのシステム構成

クライアント

- iOS
- Android
- Webブラウザ

サーバー

- Ruby on Rails
  - API
  - Web UI
  - Sidekiq（ジョブキュー）
  - Whenever（Cron）
- インフラはすべてAWS

利用しているサービス

- 構成管理、オーケストレーション
  - AWS OpsWorks
- 画像、動画配信
  - Amazon CloudFront、Amazon S3
- データストア
  - Amazon Aurora / Amazon ElastiCache(Redis, Memcached)
- その他
  - Amazon SES / Amazon SNS / Amazon SQS / Amazon Route 53 ほか
- ログ転送、ログ解析
  - Amazon Kinesis Data Firehose / Amazon Athena / Amazon EMR / AWS Batch ほか
- モニタリング
  - NewRelic / Amazon CloudWatch / Grafana ほか

いままでの環境における課題

- プロビジョニング、デプロイに時間がかかりがち
  - ChefのCookbook、レシピの数が多く、オートスケールするまでに時間がかかる
  - 問題が起きたときにデバッグしづらい
- Chefのバージョンアップが難しい
  - OpsWorksのスタックごと入れ替えが必要（仕様上）
  - ChefのCookbook、レシピの記述が古いままに
- EC2リソースを使い切るための調整が難しい
- EC2スポットインスタンスの活用ができていない（コスト削減したい）
- SSHを前提としたオペレーションになりがち
  - オペレーションコストが高い、セキュリティ面で考慮することが多い

コンテナ環境へ移行することで解決できる？

アジェンダ

① 「家族アルバム みてね」の紹介
② みてねのアーキテクチャと課題
③ コンテナ移行によって解決すること
④ コンテナサービスの選択
⑤ コンテナ環境のデザインと構築
⑥ Amazon EKS移行の状況
⑦ まとめ

コンテナ移行によって解決すること（技術面）

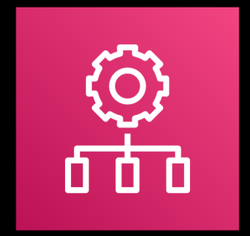
- OSに依存せず、イメージ化によるポータビリティ性
  - どこの環境でも簡単にイメージを配置できる（ただしカーネルに依存する場合がある）
- ディスポーザブルなインフラの実現
  - 高速なデプロイの実現につながる
  - SSHする機会を減らす
- 従来のVMよりも効率的なハードウェアリソースの活用
  - ボトルネックが少ない、空きリソースを活用できる
- さまざまなサービスがコンテナを前提としてきている
  - 例: AWS CodeBuild、AWS Batch、Amazon SageMakerなど
- 新技術の習得、トレンドへの追従、エンジニア採用
  - 2020年、コンテナ市場は26億8800万ドルの市場規模といわれる
    https://japan.zdnet.com/article/35095461/

コンテナ移行によって解決すること（ビジネス面）

- ユーザーにより快適に使っていただく
  - オートスケール速度向上
  - 耐障害性向上
- コスト削減
  - コンピュートリソースの効率化
  - スポットインスタンスの活用
- 新機能開発、不具合修正速度の向上
  - より事業にフォーカスできる
  - デプロイが速い
  - 環境構築しやすい

AWS OpsWorksからコンテナ環境へ

- AWS OpsWorksはECSと連携できるらしい
  - しかし、あえてAWS OpsWorksを使う必要もない
- Chef（Cookbook）からの脱却
- デプロイ速度向上への期待
- コスト削減（スポットインスタンスの活用）
- 新しい技術の導入がより進むように



AWS OpsWorks

or

Amazon Elastic Container Service

Amazon Elastic Container Service for Kubernetes

アジェンダ

① 「家族アルバム みてね」の紹介
② みてねのアーキテクチャと課題
③ コンテナ移行によって解決すること
④ コンテナサービスの選択
⑤ コンテナ環境のデザインと構築
⑥ Amazon EKS移行の状況
⑦ まとめ

Amazon ECS vs Amazon EKS

どちらも優れたコンテナサービス

- **Amazon ECS**
  - AWS Fargateが使える
  - シンプル
  - タスク単位のIAMロール
  - AWS独自サービス
- **Amazon EKS**
  - AWS Fargateはまだ使えない（資料作成時点）
    - https://github.com/aws/containers-roadmap/issues/32
  - 機能が豊富な反面、学習コストは高い
  - 世の中にあふれるKubernetesのノウハウやツールをそのまま導入できる
  - Kubernetesへの準拠性が認証されている

Kubernetesという選択肢

- 今最も注目を浴びるコンテナ技術の1つ
- 2014年、GoogleがOSSとして公開
  - https://github.com/kubernetes/kubernetes
  - GitHubスター数: 53,081（資料作成時点）
- CNCF(Cloud Native Computing Foundation) プロジェクトの1つ
- Kubernetes開発者、利用者が集まるイベント「KubeCon」参加者が年々増加。
  - 2019年5月にスペインで開催されたKubeCon Europe 2019では約7,700人の参加

Amazon EKSの利点 ①

- マネージドなコントロールプレーン
  - 自前でKubernetesを構築して運用するは大変なことが多い
  - EKSは、3つのAZにまたがって動作する2つ以上のAPIサーバーノードと3つのetcdノードで構成される
  - これらをAWSが管理してくれる
- Kubernetesのエコシステム
  - Kubernetes ユーザーは世界中でどんどん増えている
  - ノウハウやツールが豊富にある
  - Kubernetesコミュニティの既存のプラグインやツールが使える
  - プラグインやツールの独自開発で課題解決も可能
- IAMとの連携
  - 安全にAWSのリソースを扱うことができる

Amazon EKSの利点 ②

- Amazon CloudWatch Container Insights
  - 2019年５月プレビュー開始
  - EKSにおけるモニタリングがより簡単に
- EC2 Spot Instanceとの組み合わせ
  - コスト最適化
- SIG(Special Interest Group) AWS
  - https://github.com/kubernetes/community/blob/master/sig-aws/
  - “Covers maintaining, supporting, and using Kubernetes hosted on AWS Cloud.”
  - 多くのサブプロジェクトで、Kubernetes向けのドライバやコントローラーが開発されている

以上をふまえて、Amazon EKSを使うことを決定!

アジェンダ

① 「家族アルバム みてね」の紹介
② みてねのアーキテクチャと課題
③ コンテナ移行によって解決すること
④ コンテナサービスの選択
⑤ コンテナ環境のデザインと構築
⑥ Amazon EKS移行の状況
⑦ まとめ

コンテナ環境全体のデザイン

- **AWSアカウント**
  - いままでは1つのAWSアカウントで運用してきた
  - いままでのアカウントを本番専用に、ステージング、開発とAWSアカウントを分けることに
  - 「AWSにおけるマルチアカウント管理の手法とベストプラクティス」
    https://d0.awsstatic.com/events/jp/2017/summit/slide/D4T2-2.pdf
    - 完全なセキュリティとリソースの分離
    - アカウント毎に行える課金管理
    - 問題発生時の影響範囲の縮小化
  - **マルチアカウントは必須ではない**が、複雑なIAMポリシーからの脱却、Terraform利用による構築の効率性向上を前提としていたため採用
- **EKSクラスター**
  - 機能分類（サービス）ごとに分ける

コンテナ移行の道筋

① Dockerfile化

② AWSアカウントをマルチアカウント構成に

③ Terraformによる適用環境の構築

④ 必要なAWSリソースの構築

⑤ Amazon EKSでクラスターを作る

⑥ マニフェストファイルの記述、適用

Chef Cookbookの資産からDockerfileへ

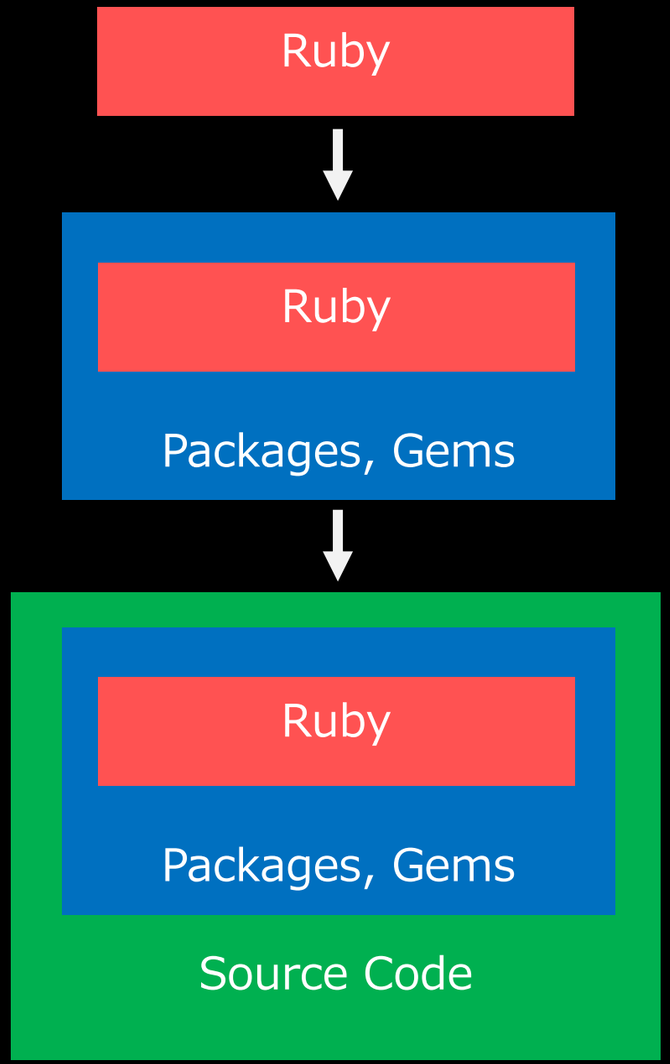
1. Dockerfileを書く 
2. AWS CodeBuildでイメージをビルド
   - buildspec.ymlを書く
   - GitHubと連携して自動でビルド
3. Amazon ECRへプッシュ
   - 同じように自動でプッシュ

Ruby on Railsプロジェクトのコンテナ化

1. Ruby（rbenv/ruby-build）のイメージ作成
2. Rubyイメージをベースに必要パッケージや、RubyGemsをインストールしたイメージの作成
3. ソースコードを入れたイメージの作成

Point

- ビルドに時間がかかる部分を分離
  - ソースコードなど差分の頻度が高い箇所のビルド時間を削減
- BuildKitを利用したビルドの高速化
  - 並列化、キャッシュ利用などによる高速化
  - AWS CodeBuildでDocker 18.09以降の環境を使う
  - 環境変数に DOCKER_BUILDKIT: “1” を指定する
- デプロイをより早くするために、コンテナイメージを小さくしていく工夫が必要（依存ライブラリが多いと大変）



ローカル開発環境でイメージの動作検証

- ローカルでの開発環境の構築
  - macOSやLinux環境でコンテナイメージを動作させるため
  - すべての開発者に使ってもらう
    - 構成や手順などの理解をチーム内で深める必要がある

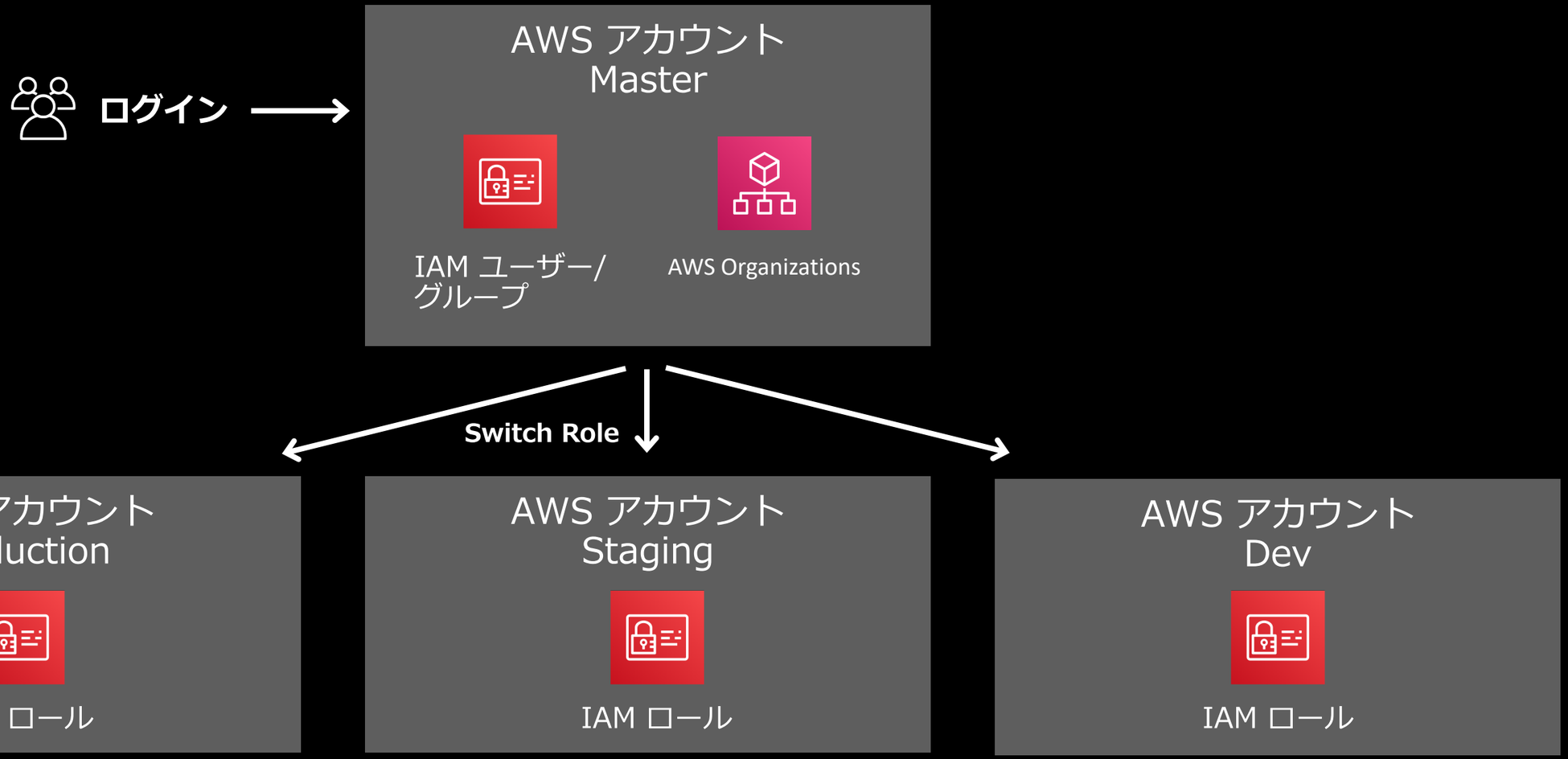
- コンテナイメージの動作確認
  - Docker Composeを使って複数のイメージ（アプリとDB、KVSなど）との連携
  - 環境変数の検証
  - ユニットテスト（RSpec）の実行と結果の確認
  - macOSでのDocker利用はIO処理が遅い問題があるので注意
    - 多少は改善できる
      - rawフォーマットのディスク利用
      - docker-syncで高速化を試みるも、不安定な現象に遭遇し不使用となった

AWSアカウントをマルチアカウントに

AWS Organizationsを使う

1. まずはAWSアカウントを新規作成（これをマスターアカウントとする）
2. マスターアカウントでAWS Organizationsを使い、子アカウントを管理
   - 従来のアカウントを招待（本番環境）
   - ステージング、開発環境用のアカウントをAWS Organizationsを使って新規作成
3. サポートプランの適用、RIの共有
   - エンタープライズサポートの場合、すべてのアカウントにサポートプランを適用できる
   - RI（Reserved Instances）はすべてのアカウントで共有できる
4. 必要に応じて上限緩和申請
   - 新規に作られたAWSをアカウントにおけるリソースの制限値に注意

複数のAWSアカウントとIAMのイメージ



複数のAWSのアカウントでのIAM設計はそれなりに大変。IAMの理解を深める必要あり。

Terraformによる適用環境の構築

- Infrastructure as a Codeの実現
- Terraformを使う
  - 利用実績が豊富、OSSで開発が活発
  - その他のプロジェクトでの利用経験があった
  - 他の選択肢: CloudFormation、AWS CDK、Pulumi、 Codenize.tools
- GitHub + AWS CodeBuildの連携
  - TerraformのCLIを手動実行しない環境を作る
  - AWS CodeBuildプロジェクトの作成
  - GitHubと連携（Webhook）
    - Pull Request作成時に terraform plan （実行計画）
    - レビューの実施
    - Pull Requestマージで terraform apply （適用）

各AWSアカウントで必要なリソースの作成

例えば

- AWS IAM
- Amazon VPC
  - ルートテーブル
  - サブネット
  - インターネットゲートウェイ
  - NATゲートウェイ
- セキュリティグループ
- Amazon Route 53
- AWS Certificate Manager(ACM)
- AWS CloudTrail、Amazon GuardDuty

など、すべてをTerraform on AWS CodeBuildで構築

Amazon VPC

AWS Certificate Manager

Amazon Route 53

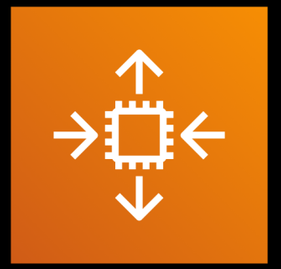
AWS Identity and Access Management (IAM)

EKSクラスターの作成

- AWS CLIで作成
  - aws eks create-cluster --kubernetes-version 1.12 ...
- eksctlでも作ることができる
  - https://eksctl.io/
  - eksctlはCloudFormationとの連携を前提としていて、Terraformとの相性が悪そうだったため使っていない（実際はそうじゃないかもしれない）

- EKSクラスターを作る上でのハマりポイント
  - 作成するときに使われた IAM エンティティ (ユーザーまたはロール) が管理者として Kubernetes RBAC認証テーブルに追加される
  - そのIAMエンティティだけがkubectlを使用してKubernetes APIサーバーにアクセスできる

EKSワーカーノードの作成



- Auto Scaling Groupを使う
  - これもTerraformで作る
  - AWS EKS Introduction https://learn.hashicorp.com/terraform/aws/eks-intro を参考にするとすんなりできる
  - EKSのワーカーノード（EC2）の作成に使う
  - EKSクラスタのバージョンに合ったEKS最適化AMIを使う
  - Podのスケジューリングの制約（Taints、Tolerations）の設定が可能
    - EKS最適化AMIのbootstrap.shのオプション指定がわかりづらいので、https://github.com/awslabs/amazon-eks-ami/blob/master/files/bootstrap.sh を読むのがオススメ

AWSアカウントとEKSクラスターのイメージ

AWS アカウント
Master

AWS Identity and Access Management (IAM)

AWS アカウント
Production

Service A VPC

Service B VPC

AWS アカウント
Staging

Service A VPC

Service B VPC

AWS アカウント
Dev

Service A VPC

Service B VPC

aws SUMMIT

マニフェストの適用

- kubectlコマンドの実行環境をどうするか
  - 手元のPCではやりたくない（セキュリティ上）
- 最初はAWS Cloud9を試した
  - ブラウザ上でのIDE、ターミナル環境
  - IAMを使った権限の制御ができる
  - 無操作によるサスペンド機能でのコスト削減
  - ブラウザのショートカットと競合して操作しにくい問題
  - 当時、東京リージョンではまだCloud9が使えなかったため、シンガポールリージョンで利用していたが、ターミナルのレイテンシの問題があった
- セッションマネージャーの利用に変更
  - EC2にSSHレスでCLIコンソールが利用できる
  - IAMを使った権限の制御ができる
  - AWS CLIから簡単にアクセス可能

適用したマニフェストの例

- ConfigMap、Secret
  - DBやMemcachedなどの情報を格納
- Deployment
  - 必要な環境変数の定義（Secret利用も含む）、起動コマンドの定義
  - Docker Composeに慣れていると設定内容をつかみやすい
- Job
  - DBの初期化（データベース作成、権限設定など）
- ALB Ingress Controller
  - ALB（Application Load Balancer）の連携機能
  - ACMのARN指定でHTTPS対応
- external-dns
  - Route 53との連携

アジェンダ

① 「家族アルバム みてね」の紹介
② みてねのアーキテクチャと課題
③ コンテナ移行によって解決すること
④ コンテナサービスの選択
⑤ コンテナ環境のデザインと構築
⑥ Amazon EKS移行の状況
⑦ まとめ

いまできていること

- 複数のAWSアカウントによる環境の分離
- Terraformによる構築のコード化、自動化
- セッションマネージャーを使ったkubectl実行環境
- Amazon EKSを使った開発環境の構築
  - EKSクラスター上でRailsアプリケーションの起動
  - ALB Ingress Controller、externa-dns
  - クライアントアプリケーションからのAPIアクセス
  - Taints、TolerationsによるPod配置のコントロール

現在進行中

- 本番環境の構築
  - 開発環境のTerraformのコードをほぼそのまま利用できる

- 本番環境の移行
  - デプロイ手順や運用
  - チーム内のノウハウ共有、ドキュメント化
  - 障害時のオペレーション
  - EKSクラスタのアップデート手順
  - できる限り属人化しないようにノウハウの共有、手順化が大事

アジェンダ

① 「家族アルバム みてね」の紹介
② みてねのアーキテクチャと課題
③ コンテナ移行によって解決すること
④ コンテナサービスの選択
⑤ コンテナ環境のデザインと構築
⑥ Amazon EKS移行の状況
⑦ まとめ

まとめ

- なぜコンテナが必要か、正しい理解が必要
  - コンテナ移行で課題解決ができるかどうかをよく考える
- コンテナに関する技術領域は広く、習得は簡単ではない
  - Kubernetesの分野は学ぶことがとても多い
  - ノウハウの横展開が大事
- コンテナ移行によって得られたもの
  - ポータビリティ性の高いディスポーザブルなインフラ
  - 新たな技術スキル、ノウハウ
  - イチから構築する手順の再確認ができた、インフラに関する情報の整理ができた
- コンテナ移行によってこれから期待できること
  - 高速にスケーラブルなインフラ
  - コスト削減
  - 開発速度の向上（事業の成長のスピードアップにつながる）

まとめ

- Dockerfileの書き方の工夫
  - 世の中にあるDockerfileを参考にするのがオススメ
  - ビルド時間の短縮につながる
- 複数のAWSアカウント運用
  - AWS Organizationsは便利
  - ただし管理対象が増えるのでInfrastructure as a Codeが大前提
- Amazon EKSの進化に追従すること
  - クラスターはアップデートし続ける必要がある（それなりの覚悟が必要）
    - 最新から3つ古いバージョンでのクラスタ作成ができなくなる
      - https://aws.amazon.com/jp/blogs/compute/updates-to-amazon-eks-version-lifecycle/
  - 検証環境を用意してアップデートの検証を十分に行う

Join Our Team

「家族アルバム みてね」を世界中の家族に広めるためのメンバーを探しています。

https://mitene.us/recruit

アプリ開発エンジニア / SRE / コンテンツ開発エンジニア、それぞれ募集中!!

aws SUMMIT

Thank you!

株式会社ミクシィ 清水 勲
@isaoshimizu

aws SUMMIT