約6年間運用したシステムを Kubernetesに完全移行するまで

Kubernetes Novice Tokyo #20
2022.6.28
Isao Shimizu

About Me

清水 勲 @isaoshimizu

- 2011年～ 株式会社ミクシィ
- 2011年～2014年 SNS「mixi」運用エンジニア
- 2014年～2018年 モンスターストライク SRE
- 2018年～現在 家族アルバム みてね SRE
- 2022年1月～ SREグループ マネージャー

週末は社会人吹奏楽団での活動（楽団長、トロンボーン約30年、たまに指揮者）。
キャンプとクラフトビールが好き。

本日お話ししたいこと

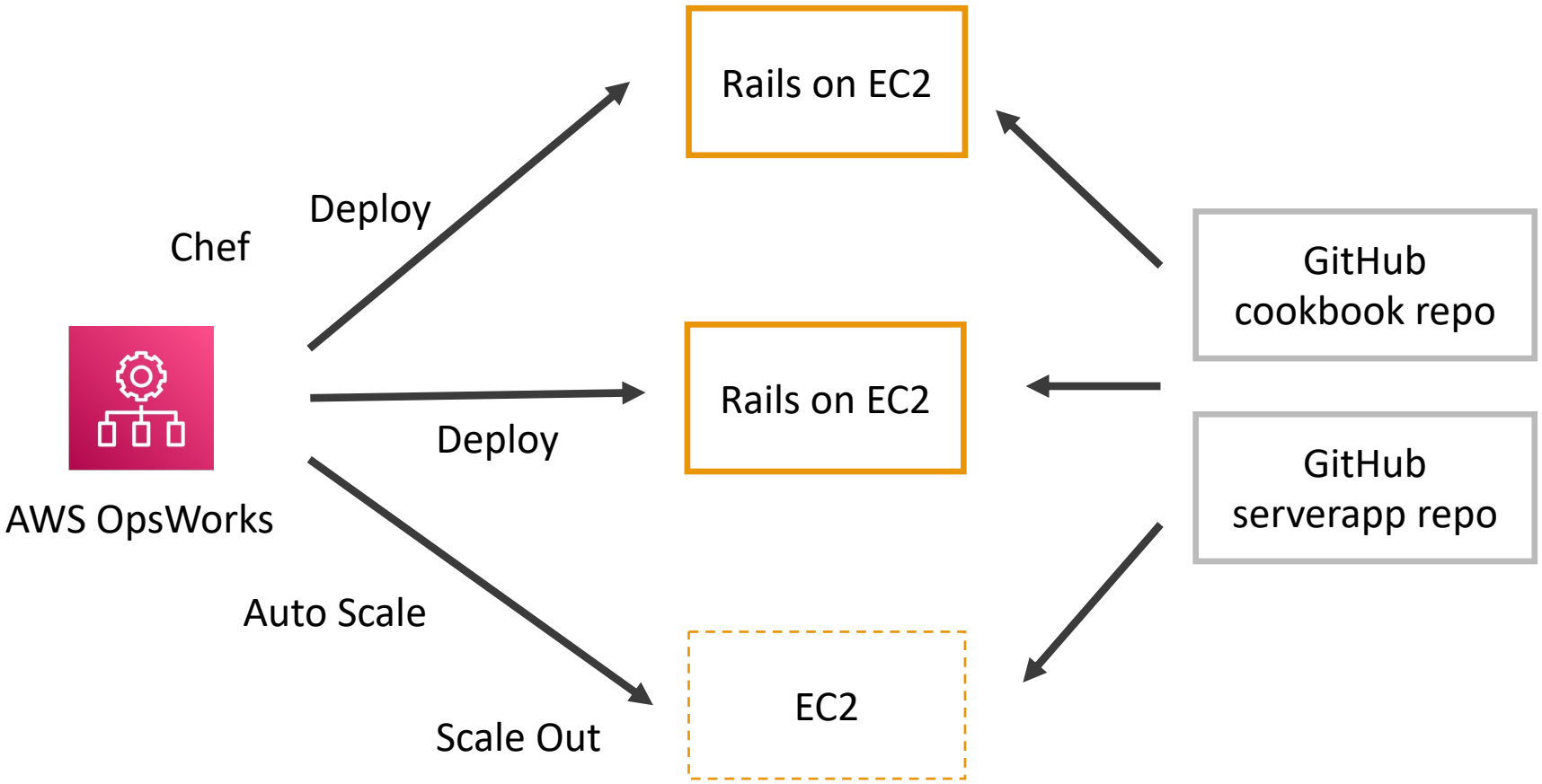
1. サービスリリース当初（2015年）のシステム
2. Kubernetes移行前（2018年頃）の課題
3. Kubernetesを選ぶにあたって検討・準備したこと
4. Kubernetesを運用する際に感じた不安
5. 本番移行の進め方
6. 本番移行後に起きたこと
7. まとめ

サービスリリース当初（2015年）のシステム

サービスリリース当初（2015年）のシステム

- iOS/Android
- Ruby on Rails
- AWS
  - OpsWorks
  - S3
  - Route 53
  - SES
  - SNS

AWS OpsWorksによるオーケストレーション



当初（2015年）のシステム

- AWS OpsWorksによるオーケストレーション
- Chef、Cookbookによるインフラの自動化
- Railsアプリケーションのデプロイにも対応
- EC2インスタンスを統合管理
- OpsWorks独自のオートスケール（時刻ベース、ロードベースなど）
- EC2のモニタリング機能の統合
- ユーザー管理、SSHキーの管理機能

などなど。当時は効率よく開発、デプロイができていた。

Kubernetes移行前（2018年頃）の課題

Kubernetes移行前（2018年頃）の課題

- Cookbookの管理、コードの依存関係が複雑に
- Chefによるデプロイ速度の問題
- オートスケールが遅い（ピーク帯に問題が起きやすい）
- SSHしてホスト内で作業をすることの煩わしさ
- Docker使って開発したい（＝本番でもDocker使いたい）
- オンデマンドインスタンス前提でコスト高
- Cron実行のSPOF問題
- WebUIでポチポチ作業、ワークロードの種類や台数が増えるとつらい
- OpsWorks独自仕様（上位Cookbook、エージェントの謎アップデートなど）

Kubernetesを選ぶにあたって検討・準備したこと

Kubernetesを選ぶにあたって検討・準備したこと

- 2018年の夏頃にシステムリニューアルの検討を開始
- オーケストレーションとしてAmazon ECS、Amazon EKSどちらを選ぶか
  - EKSの東京リージョンはまだ来てなかった頃、今ほど機能もない
  - S3に大量のデータを持っているため、AWS以外の選択肢はない
- Kubernetesの知識がほぼない中、オレゴンリージョンのEKSを触ってみる
  - Kubernetes完全ガイドを買って勉強したり他社事例を漁りまくる
  - OSSである点、開発が活発である点、エコシステムの充実ぶりに魅力を感じた
- 「家族アルバム みてね」のシステムにフィットするかどうか
  - いくつかのサービス、Rails＆Sidekiq、GPUが必要なワークロード、多数のCron設定
  - Kubernetesであれば一通り対応できるだろうという予測
  - AWS Solution Architectの方々と何度も相談を繰り返した

Kubernetesを運用する前に感じた不安

Kubernetesを運用する前に感じた不安

- Secretsの扱いどうするのが正解なのか
- コンテナイメージのCI/CDの実装イメージがなかなかできなかった
- db:migrateやassets:precompileをどこでどのタイミングで実行するのか（できるのか）
- KustomizeやHelmを選ぶことは正しいのか
- EKSのアップデートはどういう手順を取るべきなのか
- オートスケールは正しく機能するのか（Cluster AutoscalerやHPAなど）
- AWSリソースのクォーターに引っかからないか
- CronJobは時刻通りにちゃんと起動するのか
- 従来と同レベルのモニタリングができるのか
- AWSアカウント、VPC、クラスターの分割単位どうするのが良いのか

Kubernetesを運用する前に感じた不安（それぞれの解）

- Secretsの扱いどうするのが正解なのか

  => External Secretsを採用。AWS Systems Manager Parameter Storeと連携。

- コンテナイメージのCI/CDの実装イメージがなかなかできなかった

  => GitHub ActionsとArgo CDを採用。GitOpsが便利。利用事例や開発頻度なども参考に。

- db:migrateやassets:precompileをどこでどのタイミングで実行するのか（できるのか）

  => db:migrateはArgo CDのPreSync hookを利用しSync前に実行される。
  assets:precompileはmasterブランチにマージされたタイミングでGitHub Actionsで実行される。

- KustomizeやHelmを選ぶことは正しいのか

  => Argo CDを使うということで、Helmを採用。Kustomizeは使っていない。

- EKSのアップデートはどういう手順を取るべきなのか

  => 既存のクラスターをそのままアップデートするのはリスクが高いため、新バージョンのクラスターを新たに作って移行する手順に。

Kubernetesを運用する前に感じた不安（それぞれの解）

- オートスケールは正しく機能するのか（Cluster AutoscalerやHPAなど）

  => Job Workerで検証する。本番で徐々にチューニングしていく。ある程度知見が溜まったらAPIにも。

- AWSリソースのクォーターに引っかからないか

  => ハマりポイントはAWSから事前にヒアリング。

- CronJobは時刻通りにちゃんと起動するのか

  => 開発環境やステージング環境でテスト。

- 従来と同レベルのモニタリングができるのか

  => CloudWatch Container Insights、New Relic Infrastructure、Prometheus、Grafanaなどを活用してきた（今はPrometheus、Amazon Managed Service for Prometheus、Grafanaを利用）。

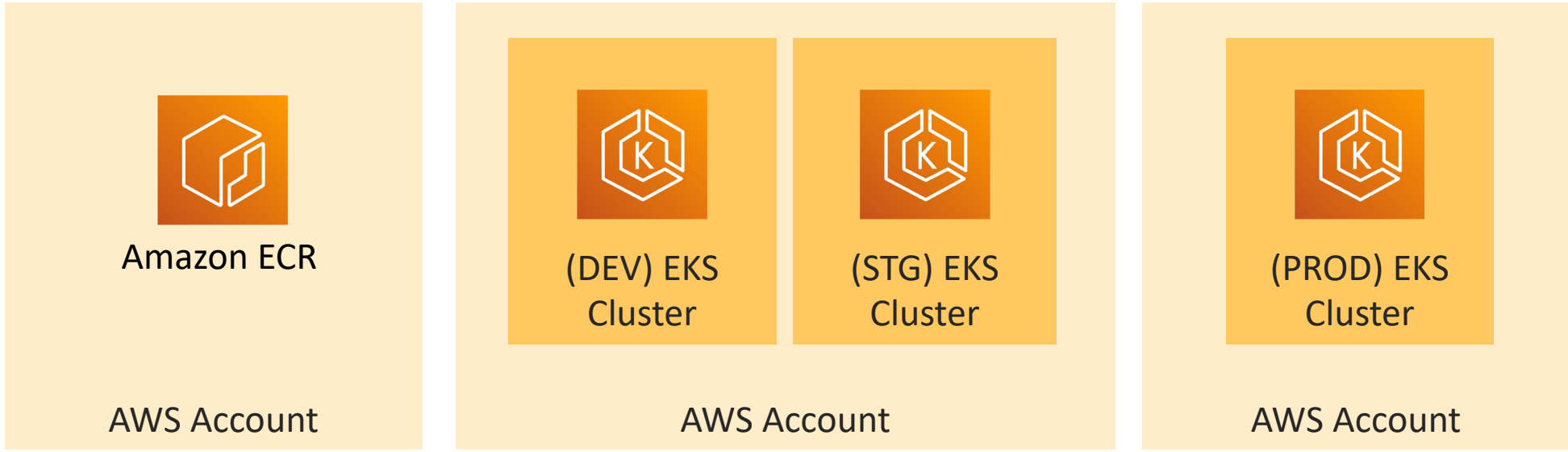
- AWSアカウント、VPC、クラスターの分割単位どうするのが良いのか

  => 次ページ以降に紹介

現在の構成

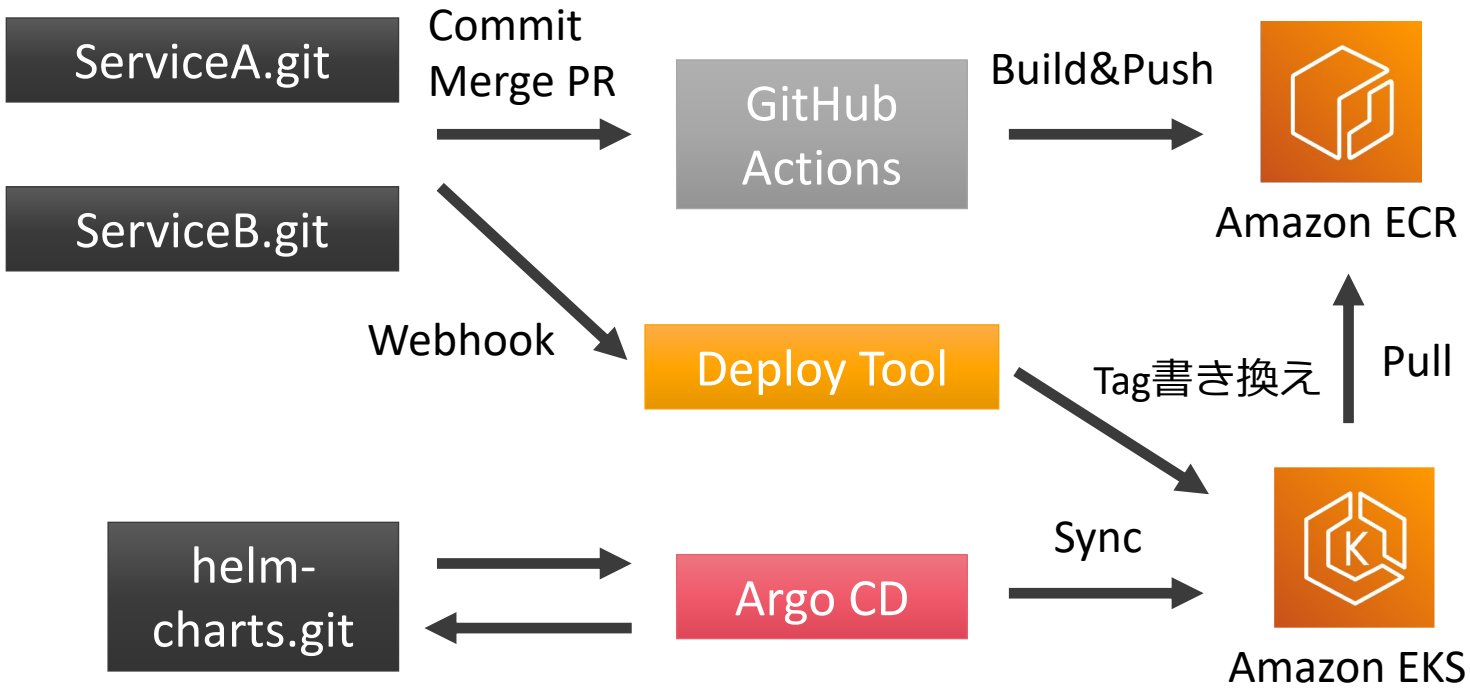
現在の構成（アカウントとEKSクラスター）

- 本番とそれ以外の環境でAWSアカウントを分ける（元々１アカウントだったが後から分割）
- EKSクラスターとVPCは環境ごとに用意
  - 最初サービス毎に用意したが管理コストが高く、後で統合した
- ECRは独立したAWSアカウントに



現在の構成（デプロイ）

- Kubernetesの構成はHelm Chartで管理され、Argo CDによって同期されている
- Helm Chart内のコンテナイメージはGitHub ActionsでビルドされECRにプッシュされる
- GitHub Actionsの実行が成功するとDeploy Toolに通知され、Kubernetesに反映される



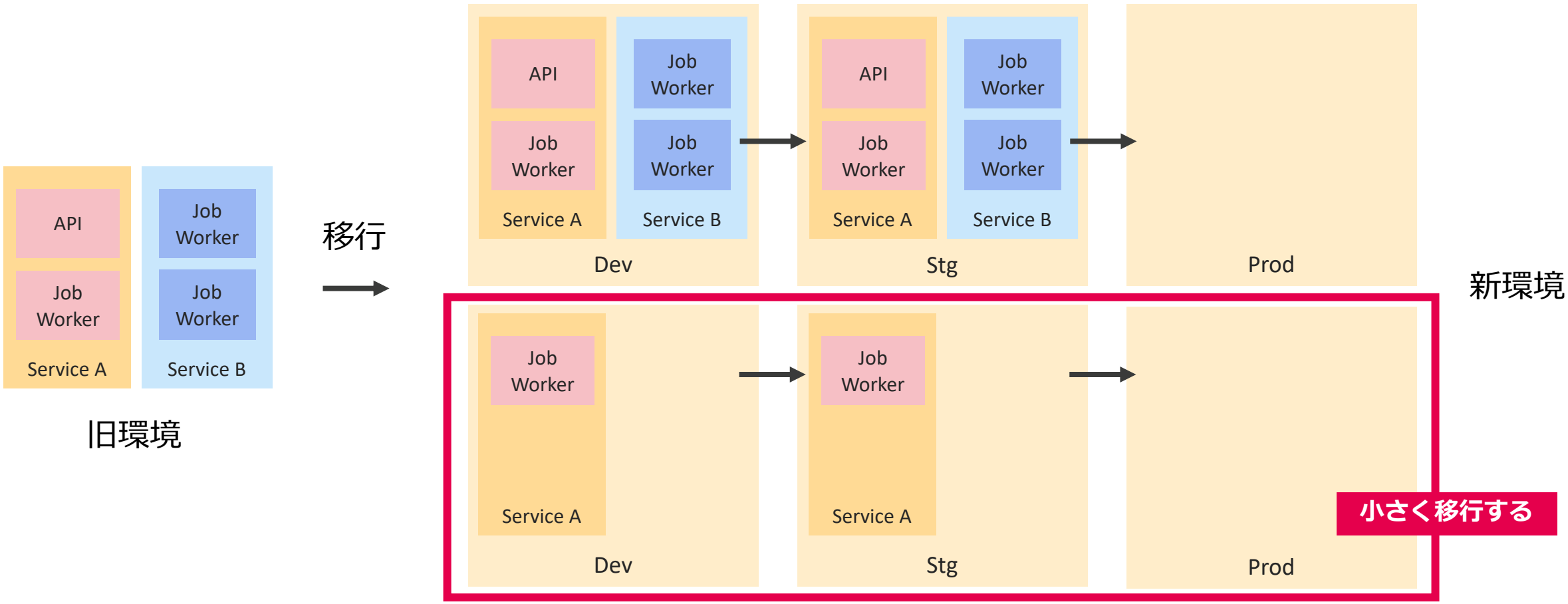
移行の進め方

移行の進め方①

- 新VPC、サブネット構築、EKSの作成、新旧VPC Peering
- Argo CD、Cluster Autoscaler、Fluent Bit、New Relicなど基盤共通のソフトウェアをセットアップ
- デプロイを新旧環境同時におこなう仕組みを作る
  - 片方の環境だけデプロイするのを禁止する
- ジョブワーカーを先に移行し、APIリクエストを受け付けるものは後に
  - APIはALBの加重ルーティングによって新旧の割合を変えて徐々に移行する
- 新旧で同時に起動して困るものは旧環境側の停止→新環境側の開始（停止時間がどのくらい許容できるかについて確認しておく）
- 移行が終わったものは旧環境から削除

移行の進め方②

- Terraform Workspacesを利用して各環境のコードをできるだけ共通化
- ワークロードごとに開発環境、ステージング、本番と移行することで移行時のリスクを減らす



本番移行後に起きたこと
（1つだけ紹介）

本番移行後に起きたこと（1つだけ紹介）

不定期に502/504エラーが発生

- EKS managed node groupsを導入した際、Spotインスタンスを利用した場合でもAWS Node Termination Handlerが不要になると思っていた。

  https://aws.amazon.com/jp/blogs/containers/amazon-eks-now-supports-provisioning-and-managing-ec2-spot-instances-in-managed-node-groups/

  > **To handle Spot interruptions, you do not need to install any extra automation tools on the cluster, for example: AWS Node Termination Handler.**

  と書いてある

- エラー発生時、CloudTrailにBidEvictedEventというイベントが記録されていた。
- EC2 Auto Scalingにはキャパシティリバランスというのがあり、 EKS managed node groupsではこれがオプトインされている。
- AWS Node Termination Handler(NTH)を導入したところエラーが解消された。

2022.8.17追記修正 本スライド公開時にはNTHが導入されていないとキャパシティリバランスのイベントには対応できないと記載しておりましたが、不確定な情報であるため記載を修正いたしました。

まとめ

まとめ

- Kubernetesは実際触ってみないとわからないことが多い
  - ブログや書籍、他社の事例で学べることも多いが、触ることが何より学べる
  - AWSサポートやSAとの相談はめちゃくちゃありがたい
  - 世の中の情報量は増えてきているので従来よりはハードルが下がっていると思う
- サービスの特性やシステムの規模によってKubernetesの向き不向きはある
  - 小さく始めるにはECSの方が良いという印象
  - Kubernetesの学習、キャッチアップ、運用をチーム全体で継続できるかどうか
- システムのリプレース作業は小さく行い、早くフィードバックを得られるようなサイクルを回すことが大事

ありがとうございました