Amazon EKS

GW 直前！まだまにあうコンテナバケーションwith Amazon EKS

Amazon EKS逆引き辞典

成尾 文秀
ソリューションアーキテクト
アマゾン ウェブ サービス ジャパン株式会社

#EKSMatsuri

自己紹介

#EKSMatsuri

名前

成尾 文秀（なりお ふみひで）

所属

アマゾン ウェブ サービス ジャパン 株式会社

技術統括本部 ソリューションアーキテクト

好きなAWSサービス

Amazon Simple Storage Service (Amazon S3)

Amazon Elastic Container Service (Amazon ECS)

Amazon Elastic Kubernetes Service (Amazon EKS)

aws

本資料の使い方

- Amazon EKS を利用した構築・運用に関する各項目（INDEX）からナレッジを確認
- 最新情報や他ナレッジを検索する際の検索ワードや勘所の参考

注意点

本逆引き資料は 2020/04/30 時点の内容のため利用する際は各項目の最新情報を確認

INDEX

#EKSMatsuri

コンテナイメージ

- Base Image は何を選択したらいいの？
- Dockerfile の書き方で注意すること
- コンテナレジストリは何を使えばいいの？
- コンテナイメージのセキュリティ（脆弱性スキャン）
- コンテナイメージのセキュリティ（権限管理）

Amazon EKS 構築編

- EKS クラスターを作成する方法は？
- EKS クラスター サブネットには何を選ぶの？
- EKS クラスター サブネットのサイズは？
- Data Plane には何を選んだら良いの？
- EKS クラスターの認証とIAM
- IAM ユーザーをEKSクラスターに追加するには？
- Load Balancer の種類と選択
- クラスターエンドポイントのアクセスコントロール
- ストレージを利用するには？
- AWS Fargate
- AWS Fargate での Pod の適切なサイズの選ばれ方
- AWS Fargate Profile

Amazon EKS バージョンとアップグレード

- EKS におけるバージョンの考え方とは？
- EKS クラスターの更新はどうやるの？
- Unmanaged nodes 更新はどうやるの？
- Managed Node Group 更新はどうやるの？

Amazon EKS 運用編

- Amazon VPC CNI plugin の仕組みとパラメーター
- Spot Instance と中断への対応
- Pod に IAM ロールを設定するには？
- Amazon EKS コントロールプレーンのログ記録
- アプリケーションログを扱うには？
- セキュアストリングをAWSで管理するには？

その他

- AWS App Meshとは
- Amazon CloudWatch Container Insightsとは
- EKS の SLA /コンプライアンス対応状況は？
- コストを削減するには？
- サービスクォータ
- Roadmap 情報の入手や機能のリクエストをしたい
- Amazon EKS を学ぶために役立つ情報

aws

#EKSMatsuri

コンテナイメージ

Base Image は何を選択したらいいの？

#EKSMatsuri

デプロイ開始時から実際にトラフィックをさばける状態になるまでの時間はコンテナの運用を考える上では重要なポイント

時間がかかると次のような事が起こる可能性

- 新バージョン（ 新機能、Bug Fix ）のデプロイ、メンテナンス時間への影響
- スケールアウトに時間がかかり、アクセス不可などユーザーへの影響
- テストでの待ち時間が増え、開発サイクルへの影響

aws

Base Image は何を選択したらいいの？

#EKSMatsuri

時間を短縮するために考えるポイント

- コンテナイメージサイズ
- アプリケーション起動までの時間

コンテナイメージサイズを小さくするための方法

- Base Image の選択
    - 軽量コンテナ（ alpine, scratch, busybox, etc. ）
        - サイズに拘りすぎて安定性を損なう事がないよう注意

- Dockerfile の書き方にも注意
    - 不要なパッケージ、レイヤーを削除（ 本資料 Dockerfile のページ参照 ）

aws

Base Image は何を選択したらいいの？

#EKSMatsuri

アプリケーション起動までの時間を小さくするために考えるポイント

- アプリケーション開発言語
  - 実行環境のためのファイル、パッケージが少ないものを選択
  - シングルバイナリ（ Golang, etc. ）は相性が良い
- アプリケーション起動手順
  - パッケージのアップデートを起動後に実施しない
  - コンテナイメージの中に必要な全ファイルをパッケージ
- その他
  - コンテナ実行環境にネットワーク的に近いイメージレジストリを利用
    （例： 利用するAmazon EKS と同一リージョンの Amazon ECR ）

aws

Dockerfile の書き方で注意すること

#EKSMatsuri

Dockerfile の書き方次第でコンテナイメージのサイズに影響するだけでなくビルド時間やセキュリティにも影響

- 不要なコンテナレイヤーは作成しない
  - コンテナイメージは複数のレイヤーにより構成され Dockerfile のコマンド RUN, COPY, ADD はレイヤーを作成する
  - まとめられる RUN はまとめる
    （パッケージ導入の時は可能ならミスを減らすためアルファベット順）

```
RUN apt-get update && apt-get install -y \
  bzr \
  cvs \
  git
```

Best practices for writing Dockerfiles https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

aws

Dockerfile の書き方で注意すること

#EKSMatsuri

- 不要なパッケージはインストールしない
  - 全体としてサイズが大きくなるだけでなく、脆弱性があった場合にはセキュリティリスクにもつながる
  - apt の場合 --no-install-recommends といった推奨パッケージを入れないオプション

- 不要なファイルは削除
  - パッケージをインストール後にリストやキャッシュが残るので削除
    - apt の場合 (apt-get clean && rm -rf /var/lib/apt/lists/*)
    - yum の場合 (yum clean all)
    - apk の場合 (--no-cache)

Best practices for writing Dockerfiles https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

aws

Dockerfile の書き方で注意すること

#EKSMatsuri

- .dockerignore を利用
  - 実行する Dockerfile が置かれているディレクトリ配下にある余計なファイル、ディレクトリまで処理の対象としてしまわないように除外

- ビルドキャッシュを活用
  - Dockerfile に記述された順番に処理される、キャッシュが利用できない場合（イメージ内のファイルのチェックサムの比較で異なる等）には変更された行以降が再実行
    - 頻繁に更新されるものほど Dockerfile の後半に記述

Best practices for writing Dockerfiles https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

aws

Dockerfile の書き方で注意すること

#EKSMatsuri

- マルチステージビルドを利用
  - アプリケーションのコンテナイメージを作る事を考えた場合、アプリケーション**ビルド用**に必要な環境とアプリケーション**実行用**に必要な環境で必要なパッケージが異なる
  - 以前はそれぞれの Dockerfile を用意し、ビルド用イメージを作成しビルド後のファイルはローカルを経由して実行用イメージ渡すといった方式を bash script を使って行うビルダーパターンが一般的だが複雑
  - マルチステージビルドでは、１つの Dockerfile 内で複数のビルドをステージという単位で実行し、ステージ間でファイルをコピーする事が可能なので大幅に処理を簡素化（ 特定ステージだけのビルドを実行したり、外部イメージをステージとして指定するといった事も可能 ）

Best practices for writing Dockerfiles https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

aws

コンテナレジストリは何を使えばいいの？

#EKSMatsuri

コンテナレジストリは、単にコンテナイメージをストアできれば良いだけでなく、デプロイ時に同時に大量の Pull が行われた場合でも要求に応えられるだけのスケーラブルで、冗長性と耐久性がある高可用性なレジストリが求められる

- フルマネージドなコンテナイメージレジストリの Amazon ECR の特徴

- スケーラブルかつ高い可用性
- セキュア (IAM連携、クロスアカウントアクセス、保管イメージの自動的な暗号化)
- Amazon ECS / Amazon EKS / Kubernetes / SageMaker / AWS Batch 等で利用可能
- ライフサイクルポリシーにより、イメージの自動クリーンアップが可能（ 「タグなしイメージはpush登録からn日経過したものは削除」や「n個以上は削除」といったルールを複数定義可能 ）

Amazon ECR https://aws.amazon.com/jp/ecr/

aws

コンテナイメージのセキュリティ(脆弱性スキャン)

#EKSMatsuri

- コンテナのセキュリティを考えると
  - 不要なソフトウェア（パッケージ）を入れない
  - 入れたソフトウェア（パッケージ）の脆弱性がないかイメージをスキャン

- Amazon ECR にある「イメージスキャン」の機能の特徴
  - オープンソースの CoreOS Clair project を利用した脆弱性 (CVEs) の 静的スキャン
  - スキャンはAPIまたはマネジメントコンソールにて実行可能
  - リポジトリ側で「プッシュ時にスキャン」を有効にするとイメージが push されたらスキャン実行
  - 同じイメージに対するスキャンは24時間につき1回迄で超えるとスロットリング
  - スキャン費用は無料

イメージスキャン　https://docs.aws.amazon.com/ja_jp/AmazonECR/latest/userguide/image-scanning.html

aws

コンテナイメージのセキュリティ(脆弱性スキャン)

#EKSMatsuri

ECRのイメージスキャン実行結果はコンソールやAPIから確認

EventBridge からスキャン完了の通知を受信


```
{
   "id":"7bf73129-1428-4cd3-a780-95db273d1602",
   "detail-type":"ECR Image Scan Status Notification",
   "source":"aws.ecr",
   "account":"123456789012",
   "time":"2015-11-11T21:29:54Z",
   "region":"us-east-1",
   "resources":[
      "arn:aws:ecr:us-east-1:123456789012:repository/my-repo"
   ],
   "detail":{
      "repository-name":"my-repo",
      "image-digest":"sha256:901eb80a33204e7d9fef4313423e4affd9c14f8ac2ce1e09852e3e7e7be1d522",
      "scan-status":"COMPLETE"
   }
}
```


スキャン結果は AWS SDKを利用して以下コマンドで取得可能

aws ecr describe-image-scan-findings --repository-name ***prd*** --image-id imageTagt=latest

上記は imageTag で指定しているが imageDigest の指定も可能

レスポンスのfindingsに含まれる情報の例

name：CVE番号、description：詳細、uri：脆弱性に関する追加情報へのリンク、severity：緊急度

イベントと EventBridge　https://docs.aws.amazon.com/ja_jp/AmazonECR/latest/userguide/ecr-eventbridge.html

describe-image-scan-findings　https://docs.aws.amazon.com/cli/latest/reference/ecr/describe-image-scan-findings.html

aws

コンテナイメージのセキュリティ(脆弱性スキャン)

#EKSMatsuri

スキャンの種類

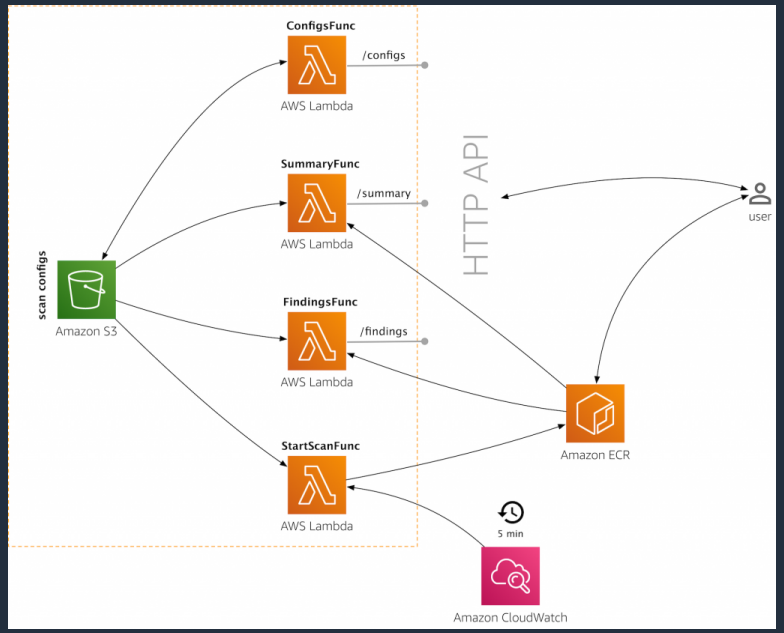
- 動的スキャン (Dynamic scanning)
  - ランタイム環境で実行されるスキャン
  - すでに実行されているコンテナの脆弱性を特定することが可能であり、ビルド時点でインストール済みのソフトウェアに脆弱性が含まれていることが後日発覚した際や、ゼロデイの脆弱性なども検出可能
  - OSS の CNCF Falco, APNパートナーの Aqua Security, Trend Micro, Twistlock など
- 静的スキャン (Static scanning)
  - デプロイ前のフェーズで実行されるスキャンのためコンテナが実行される前に脆弱性に特定することが可能
  - コンテナイメージ内の OS パッケージをスキャンし共通脆弱性識別子 (CVE) を検出
  - ECR のイメージスキャン機能はこちらに該当

aws

コンテナイメージのセキュリティ(脆弱性スキャン)

#EKSMatsuri

ECRの静的スキャンではスキャン後に発見された脆弱性に対応するために
定期スキャンの実施を推奨（以下は aws-samples に公開済）



4つの Lambda

- **ConfigFunc**
  スキャン対象に関する管理（ S3 に保存 ）
- **SummaryFunc**
  スキャン結果のサマリを提供
- **FindingsFunc**
  スキャン詳細結果の Atom フィードを提供
- **StartScanFunc**
  スキャン実行（CloudWatch Event から5分毎に実行）

ECR Container Image Re-Scan
https://github.com/aws-samples/amazon-ecr-continuous-scan

Amazon ECRのネイティブなコンテナイメージスキャン機能について
https://aws.amazon.com/jp/blogs/news/amazon-ecr-native-container-image-scanning/

コンテナイメージのセキュリティ(権限管理)

#EKSMatsuri

ポッドのセキュリティポリシー（PSP）のアドミッションコントローラー

- ルールに対してポッドの作成を検証
- Kubernetes バージョン 1.13 以降
- 設定をYAMLで用意して作成、変更、削除可能
- デフォルトのポッドセキュリティポリシー (eks.privileged) は制限なし＝PSP無効と同様

```
Settings:
  Allow Privileged:                       true
  Allow Privilege Escalation:             0xc0004ce5f8
  Default Add Capabilities:               <none>
  Required Drop Capabilities:             <none>
  Allowed Capabilities:                   *
  Allowed Volume Types:                   *
  Allow Host Network:                     true
  Allow Host Ports:                       0-65535
  Allow Host PID:                         true
  Allow Host IPC:                         true
  Read Only Root Filesystem:              false
  SELinux Context Strategy: RunAsAny
    User:                                 <none>
    Role:                                 <none>
    Type:                                 <none>
    Level:                                <none>
  Run As User Strategy: RunAsAny
    Ranges:                               <none>
  FSGroup Strategy: RunAsAny
    Ranges:                               <none>
  Supplemental Groups Strategy: RunAsAny
    Ranges:                               <none>
```

ポッドのセキュリティポリシー　https://docs.aws.amazon.com/ja_jp/eks/latest/userguide/pod-security-policy.html

aws

コンテナイメージのセキュリティ(権限管理)

#EKSMatsuri

ルールの例

- プロセスを root ユーザーでは実行させない
  runAsUser:
    rule: 'MustRunAsNonRoot'

- コンテナのルートファイルシステムをRead Only にする
  readOnlyRootFilesystem: true

- 特権コンテナを拒否する
  privileged: false

- 特権昇格はさせない
  allowPrivilegeEscalation: false

ポッドのセキュリティポリシー　https://docs.aws.amazon.com/ja_jp/eks/latest/userguide/pod-security-policy.html
Pod Security Policies - Kubernetes　https://kubernetes.io/docs/concepts/policy/pod-security-policy/

aws

#EKSMatsuri

Amazon EKS 構築編

aws

EKS クラスターを作成する方法は？

#EKSMatsuri

- 以下の方法にてEKSクラスター作成
  - eksctl
  - AWS マネジメントコンソール
  - AWS CLI

- eksctl
  - 各種必要な IAM Role, VPC, Subnet や設定を含めて一括設定
  - 実態はCloudFormationを利用
  - AWSのサポート対象（The official CLI for Amazon EKS）

eksctl の開始方法　https://docs.aws.amazon.com/ja_jp/eks/latest/userguide/getting-started-eksctl.html
eksctl　https://eksctl.io/

aws

EKS クラスター サブネットには何を選ぶの？

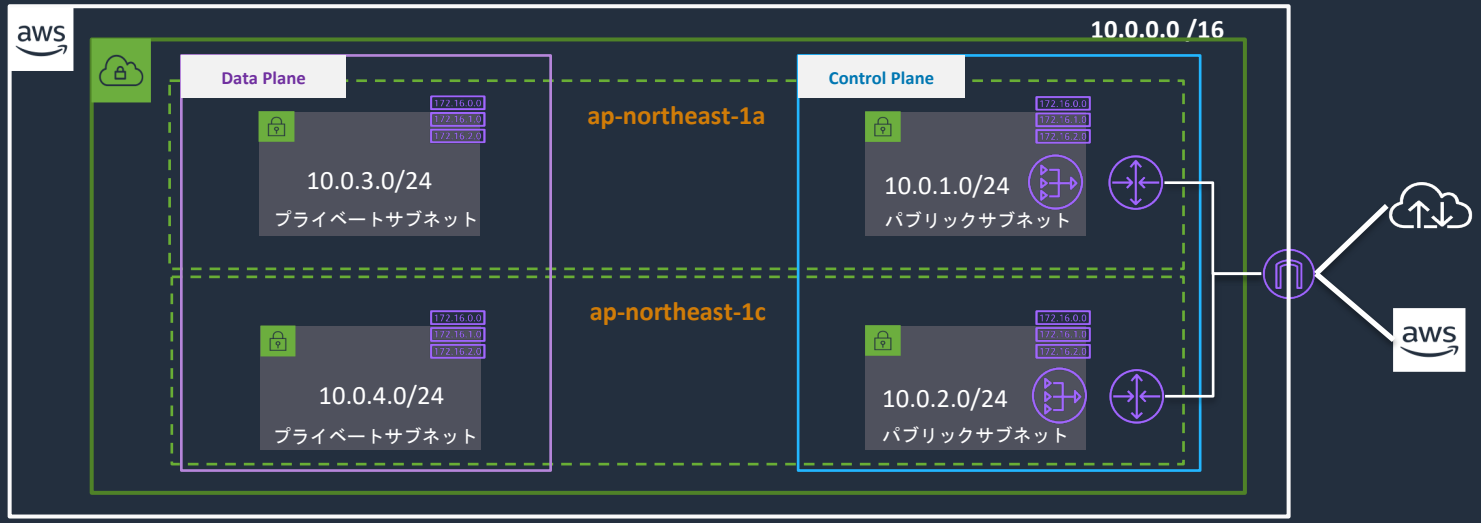
#EKSMatsuri

ドキュメントより抜粋して引用

> Amazon EKS には、2 つ以上のアベイラビリティーゾーンにサブネットが必要です。ネットワークアーキテクチャとしては、ワーカーノードにプライベートサブネットを使用し、Kubernetes にパブリックサブネットを使用して、インターネット向けのロードバランサーを作成することをお勧めします。

ワーカーノードにプライベートサブネットを指定した場合

- API サービスに対する通信といったコンテナからインターネットに向けて通信する場合には NAT ゲートウェイが必要
- Amazon ECR からのコンテナイメージの取得（Pull）は Private Link の作成を推奨



クラスター VPC に関する考慮事項　https://docs.aws.amazon.com/ja_jp/eks/latest/userguide/network_reqs.html

aws

EKS クラスター サブネットのサイズは？

#EKSMatsuri

Control Plane

- マネージドで負荷に応じてスケーリング＝スケーリングする際に消費する IP を考慮
- Load Balancer が消費する IP アドレスの数にも注意
  ALB / CLB はスケーリングのために各サブネットに少なくとも 8 つ以上の IP アドレスを利用可能な状態にしておく
  Load Balancer の数にも注意
- 172.17.0.0/16 を使用しない
  Docker は Amazon EKS クラスターの 172.17.0.0/16 CIDR 範囲で実行されるためクラスターのVPCサブネットが重っている場合はエラーが発生

Data Plane

- 各Pod は指定したサブネットの IPを利用
  Amazon VPC CNI plugin for Kubernetes が IPをPoolしPod起動時に割当を行う（Pool数に注意　詳しくは本資料の運用編に記載）
- EC2を利用する場合には選択したインスタンスタイプ・サイズごとに利用できるIP数の上限に注意
  CPU、Memory のリソースに余裕があっても IP の数が不足すると Pod が起動できない (Pending)
  IPが不足した場合、カスタムネットワークで拡張する事も可能（ただしManaged Node Groupでは利用不可）

aws

Data Plane には何を選んだら良いの？

#EKSMatsuri

- 以下の方法が選択可＆混在可
  - Unmanaged nodes (EC2)
    - Amazon EKS 最適化 Linux AMI（イメージ）以外にカスタムイメージを指定可能
    - BootstrapArguments を利用可（kubelet の追加引数などbootstrap scriptに渡す）
    - スポットインスタンス利用可（複数のインスタンスタイプ指定を推奨）
  - Managed Node Groups (EC2)
    - 最新のAMI（イメージ）への更新がマネジメントコンソール or CLI から簡単に実施
  - Fargate
    - インスタンスに関する管理（タイプ、サイズ、スケーリング）不要

柔軟なほど管理する項目が増え、簡素なほど管理が不要に

-> ワークロードによって選択

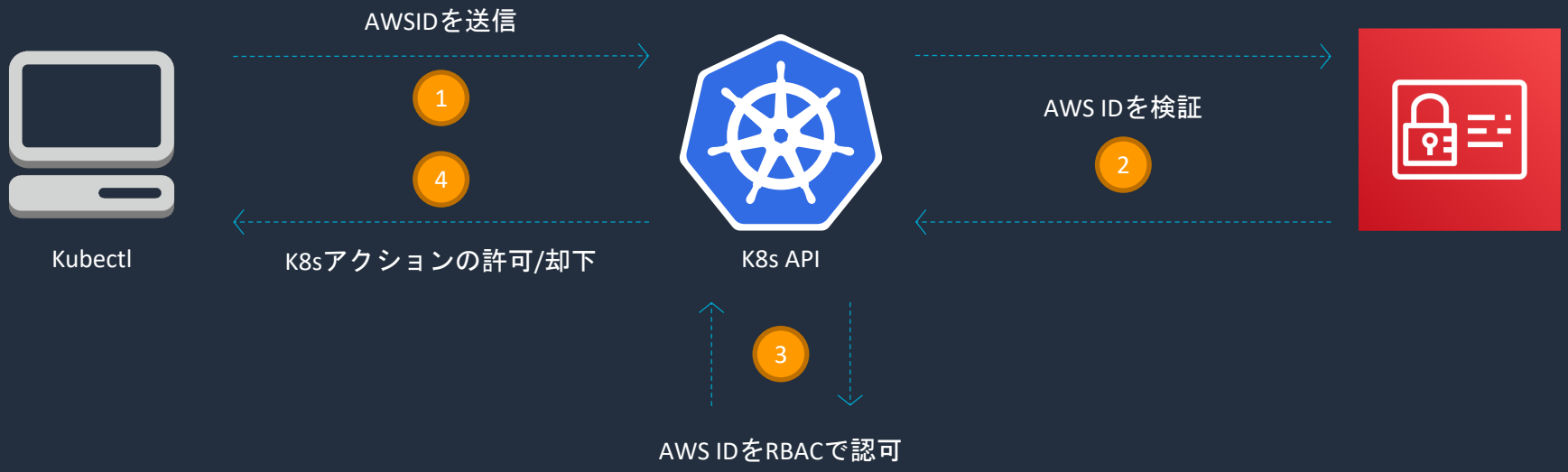
aws

EKS クラスターの認証とIAM

#EKSMatsuri

EKSでは IAMとKubernetes RBAC (Role Based Access Control) が連携

- IAMで許可を与えただけでは Kubernetes API サーバーへのコマンド実行は不可
- Kubernetes 内の aws-auth ConfigMap にIAMユーザーのARNなど設定



クラスター認証の管理　https://docs.aws.amazon.com/ja_jp/eks/latest/userguide/managing-auth.html
クラスターのユーザーまたは IAM ロールの管理　https://docs.aws.amazon.com/ja_jp/eks/latest/userguide/add-user-role.html

aws

IAM ユーザーをEKSクラスターに追加するには？

#EKSMatsuri

クラスター作成時に利用されていた AWS クレデンシャル（IAM ユーザーまたはロール) とは異なるクレデンシャルを使用して kubectl を実行する場合、aws-auth ConfigMap（Worker Nodeをクラスター追加時に作成済）を変更し mapUsers / mapRoles での設定が必要

mapUsersを利用した例

```
...
  mapUsers: |
    - userarn: arn:aws:iam::555555555555:user/admin
      username: admin
      groups:
        - system:masters
    - userarn: arn:aws:iam::111122223333:user/ops-user
      username: ops-user
      groups:
        - system:masters
```

クラスターのユーザーまたは IAM ロールの管理　https://docs.aws.amazon.com/ja_jp/eks/latest/userguide/add-user-role.html
Amazon EKS トラブルシューティング　https://docs.aws.amazon.com/ja_jp/eks/latest/userguide/troubleshooting.html#unauthorized

aws

Load Balancer の種類と選択

#EKSMatsuri

ドキュメントから抜粋して引用

> タイプ LoadBalancer の Kubernetes サービスを通じて Amazon EC2 インスタンスワーカーノードで実行されるポッドの Network Load Balancer および Classic Load Balancer をサポートします。
>
> Classic Load Balancer と Network Load Balancer は、AWS Fargate (Fargate) で実行されているポッドではサポートされません。Fargate 入力の場合は、Amazon EKS (最小バージョン v1.1.4) で ALB Ingress Controller を使用することをお勧めします。

- 互換性などの理由がなければALBまたはNLBを選択
- EC2 の場合： Type: LoadBalancer にて CLB / NLBを、 ALB Ingress Controller を使用して ALB が利用可能
- Fargate の場合は、ALB Ingress Controller を使用して ALB のみ利用可能

- Type: Load Balancer にて NLB を利用する場合には以下 Annotation の追加が必要

```
annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: nlb
```

- ALB Ingress Controller は ALB および必要な AWS サポートリソースの作成をトリガーするコントローラー

ロードバランシング https://docs.aws.amazon.com/ja_jp/eks/latest/userguide/load-balancing.html

aws

クラスターエンドポイントのアクセスコントロール

#EKSMatsuri

エンドポイントのパブリックアクセスの無効をセキュリティを考慮し選択可能

パブリックアクセス	プライベートアクセス	動作
有効	無効	・EKSのデフォルトの設定 ・インターネットからAPIへアクセス可（kubectl） ・VPC内のKubernetes API リクエスト（Control Plane ⇔ Data Plane）はVPC 内ではなく Amazon のネットワークを経由
有効	有効	・インターネットからAPIへアクセス可（kubectl） ・VPC内のKubernetes API リクエスト（Control Plane ⇔ Data Plane）はプライベート VPC エンドポイントを経由
無効	有効	・インターネットからAPIへアクセス不可（kubectl） ・API サーバーへのトラフィックはすべて クラスターの VPC 内からルーティングが必要

Amazon EKS クラスターエンドポイントのアクセスコントロール
https://docs.aws.amazon.com/ja_jp/eks/latest/userguide/cluster-endpoint.html

aws

クラスターエンドポイントのアクセスコントロール

#EKSMatsuri

専用クラスターエンドポイント（パブリックアクセス無効）にアクセスする条件、方法

- アクセスはクラスターの VPC または接続されたネットワーク内
- 接続されたネットワーク
  - AWS Transit Gateway や VPN, Direct Connect, VPC Peering, etc. を使用してクラスターのVPC と接続されたネットワーク
- Amazon EC2 踏み台ホスト
  - インスタンスをクラスターの VPC の Public Subnet で起動しSSH 経由で実行
  - EKS Control Plane 側 Security Group にて インバウンドルールで EC2 踏み台ホストからPort 443 での通信の許可が必要
- AWS Cloud9
  - クラスターの VPC に AWS Cloud9 IDE を作成
  - EKS Control Plane 側 Security Group にて インバウンドルールで Cloud9 の Security Group から Port 443 での通信の許可が必要

Amazon EKS クラスターエンドポイントのアクセスコントロール
https://docs.aws.amazon.com/ja_jp/eks/latest/userguide/cluster-endpoint.html

aws

ストレージを利用するには？

#EKSMatsuri

- ワークロード上必要であれば、Kubernetes 1.14 以降のクラスターでは Amazon EBS CSI ドライバー 、それ以前のバージョンではインツリー Amazon EBS ストレージプロビジョナーを使用して永続的なストレージを利用可能（ 現在 Fargate は特権を持つコンテナは未サポートのため利用不可 ）

- CSI (Container Storage Interface) ドライバーを使用することでKubernetesのリリースサイクルとCSIドライバーのリリースサイクルを分離

- Amazon CSIドライバーとして、EBS、EFS、FSx for Lustre の3種類を提供
    - 本番稼働用に Amazon EKS によって十分にテスト済、ただしベータリリースのため今後変更がある際は、次のバージョンへの移行手順を提供
    - EBS（ブロックストレージ）、EFS（NFS ファイルシステム）、FSx for Lustre （S3と連携可能な高速処理用ファイルシステム）と用途ごとに合ったものを選択

ストレージ　https://docs.aws.amazon.com/ja_jp/eks/latest/userguide/storage.html

aws

AWS Fargate

#EKSMatsuri

特徴

- Worker node のプロビジョニング、設定、スケーリング不要
- インスタンスタイプ、サイズの選定不要、スケールタイミングの最適化不要
- 各 Pod はそれぞれが独立した環境で実行され、カーネルやCPU/メモリリソース、ENI は Pod 間で共有されない
- 利用した時間課金 1 秒単位（ 最低利用 1 分 ）

考慮事項

- CLB および NLB は 未サポート（ ALB を ALB Ingress Controller で利用 ）
- daemonset は未サポート、対応できるものは sidecar パターンで対応
- 特権を持つコンテナは未サポート（ ディスクのマウント etc. ）
- pod manifest にて HostPort または HostNetwork は指定不可
- Private Subnet のみ対応
- GPU 未対応

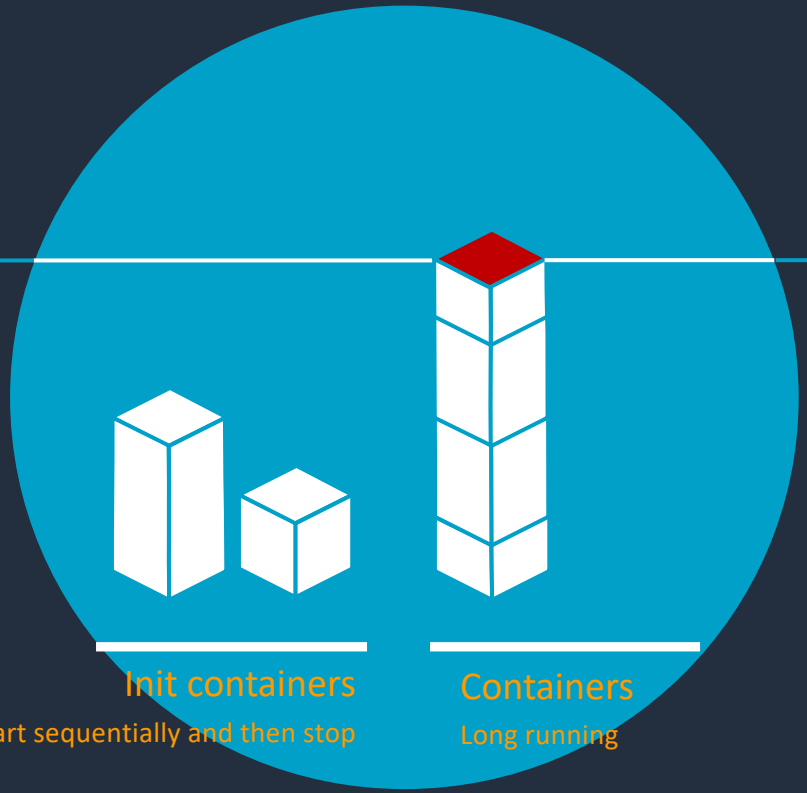
AWS Fargate https://docs.aws.amazon.com/ja_jp/eks/latest/userguide/fargate.html

aws

AWS Fargate での Pod の適切なサイズの選ばれ方

#EKSMatsuri

This



Kubernetes components

+256MB

CPU

Closest config (rounded up) is picked

MEM

CPU	Memory
256 (.25 vCPU)	512MB, 1GB, 2GB
512 (.5 vCPU)	1GB, 2GB, 3GB, 4GB
1024 (1 vCPU)	2GB, 3GB, 4GB, 5GB, 6GB, 7GB, 8GB
2048 (2 vCPU)	Between 4GB and 16GB in 1GB increments
4096 (4 vCPU)	Between 8GB and 30GB in 1GB increments

Fargate sizing combinations

aws

#EKSMatsuri

AWS Fargate Profile とは何か

動作

- セレクター（namespace, label）にマッチした Pod が Fargate にスケジュールされる
- 1つの Profile には5つまでセレクターを指定でき、いずれかにマッチすれば Fargate にスケジュールされる (= OR 条件)
- 複数 Profile があり複数とマッチした場合にはランダムで選択
  - マニフェスト側の記述で指定も可能 (ただし指定した Profile とのセレクターマッチは必要）
    eks.amazonaws.com/fargate-profile: profile_name

-> 条件にマッチしたものだけ Fargate へその他は EC2 の Worker Node へ、といった使い分け

-> AZ ごとに Profile を作成しマニフェスト側の指定で起動させる AZ を固定

削除

- Profile を削除すると削除対象の Profile を利用して起動していた Pod も削除
- 削除後、他の Profile とマッチしたら再スケジュール、マッチしなかったら Pending (マニフェストで Profile を指定しておらず、同一クラスタに EC2 がある場合はそちらへ)

AWS Fargate Profile - Amazon EKS https://docs.aws.amazon.com/eks/latest/userguide/fargate-profile.html

aws

#EKSMatsuri

Amazon EKS バージョンとアップグレード

aws

EKS におけるバージョンの考え方とは？

#EKSMatsuri

- EKS は以下の 2 つのバージョンで管理
  - Kubernetes バージョン
  - プラットフォームバージョン

- Kubernetes バージョン
  - Kubernetes プロジェクトからリリースされてからしばらくしてEKS側でもサポート
  - EKS は 3つの Kubernetes バージョンをサポート

    ※2020年4月16日現在でサポートしているバージョンは：1.15, 1.14, 1.13

- プラットフォームバージョン
  - マイナーバージョンごとに eks.1 から始まり、更新のたびにインクリメント
  - コントロールプレーン設定の有効化や、セキュリティ修正プログラムのタイミングで更新
  - 利用しているKubernetes バージョンの中の最新プラットフォームバージョンに自動更新

    ※段階的にロールアウトするため時間がかかる場合がある、最新がすぐに必要なら新規でクラスター作成

Amazon EKS Kubernetes バージョン　https://docs.aws.amazon.com/ja_jp/eks/latest/userguide/kubernetes-versions.html#
プラットフォームバージョン　https://docs.aws.amazon.com/ja_jp/eks/latest/userguide/platform-versions.html

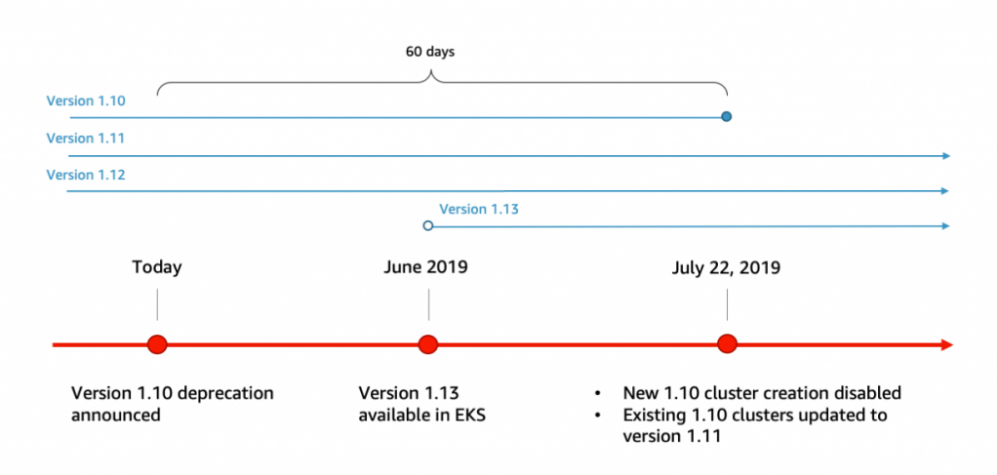
aws

EKS におけるバージョンの考え方とは？

#EKSMatsuri

- Kubernetes プロジェクトは活発で最近の傾向を見ると3ヶ月ごとにマイナーバージョンアップ

- EKS がサポートを廃止するバージョンについては廃止予定日の最低 60 日前に廃止を発表
    - 廃止したバージョンを利用した新規のクラスターは作成できなくなる
    - 次のサポートされているバージョンの最新プラットフォームバージョンに自動更新



Amazon EKS Kubernetes バージョン　https://docs.aws.amazon.com/ja_jp/eks/latest/userguide/kubernetes-versions.html#

aws

EKS クラスターの更新はどうやるの？

#EKSMatsuri

- 本番環境を新バージョンへ更新する前に、別環境にてアプリケーションの動作テストの実施を推奨
- クラスター更新にはクラスターに指定したサブネットから、2〜3 の空き IP が必要
- 既存のクラスターを更新をする前に、control plane と worker nodes のバージョンを確認し、 worker nodes が古い場合は先に worker nodes の更新を行う

  control plane 側：kubectl version –short
  worker nodes 側：kubectl get nodes

- クラスター更新は以下のいずれかで実施可能（詳細はドキュメントに記載）
  - eksctl
  - マネジメントコンソール
  - AWS CLI
- クラスター更新はKubernetes アドオン（CNI, DNS, KubeProxy）は変更しないため、クラスター更新後にドキュメントに記載の手順に従ってアドオンの更新を推奨

Amazon EKS クラスターの Kubernetes バージョンの更新　https://docs.aws.amazon.com/ja_jp/eks/latest/userguide/update-cluster.html

aws

Unmanaged nodes 更新はどうやるの？

#EKSMatsuri

- 更新の方法は 2 つ
  - 新しいワーカーノードグループへの移行
    - 新しいバージョンでワーカーノードグループを作成して pod を移行
  - 既存のワーカーノードグループの更新
    - 既存のノードを 1台ずつ 最新の AMI を使うよう更新
    - ワーカーノードグループを eksctl で作成している場合には未サポート

  -> 既存の nodes を残しながら移行する「新しいワーカーノードグループへの移行」方式を推奨

- 「新しいワーカーノードグループへの移行」の手順
  - eksctl で 新しいワーカーノードグループを作成

```
eksctl create nodegroup --cluster default --version 1.15 –name standard-1-15 \
--node-type t3.medium --nodes 3 --nodes-min 1 --nodes-max 4 --node-ami auto
```

  - クラスターに新しいワーカーノードグループが追加されたのを kubectl get nodes で確認してから
    古いワーカーノードグループを削除（古い node への taint を利用した NoSchedule や drain も実行される）

```
eksctl delete nodegroup --cluster default --name standard-workers
```

※ drain 中に 全て pod が落ちた状態にならないようPDB (Pod Disruption Budget) の利用を推奨

新しいワーカーノードグループへの移行　https://docs.aws.amazon.com/ja_jp/eks/latest/userguide/migrate-stack.html

aws

Managed Node Group 更新はどうやるの？

#EKSMatsuri

- 更新方法
  - マネジメントコンソールから対象の Managed Node Group を選択し「Update group version」を選択
    - 「Update group version」は利用可能なアップデートがある場合にのみ表示
  - 「Update AMI release version」から以下オプションを選択して確認
    - ローリング更新
      - PDB (Pod Disruption Budget) 設定を優先して更新を行い pod が node から 15 分間 drain できない場合、更新処理は失敗してエラー
    - 強制更新
      - PDB (Pod Disruption Budget) 設定を優先せず、強制的に node を再起動

マネージド型ノードグループの更新　https://docs.aws.amazon.com/ja_jp/eks/latest/userguide/update-managed-node-group.html
マネージド型ノードの更新動作　https://docs.aws.amazon.com/ja_jp/eks/latest/userguide/migrate-stack.html

aws

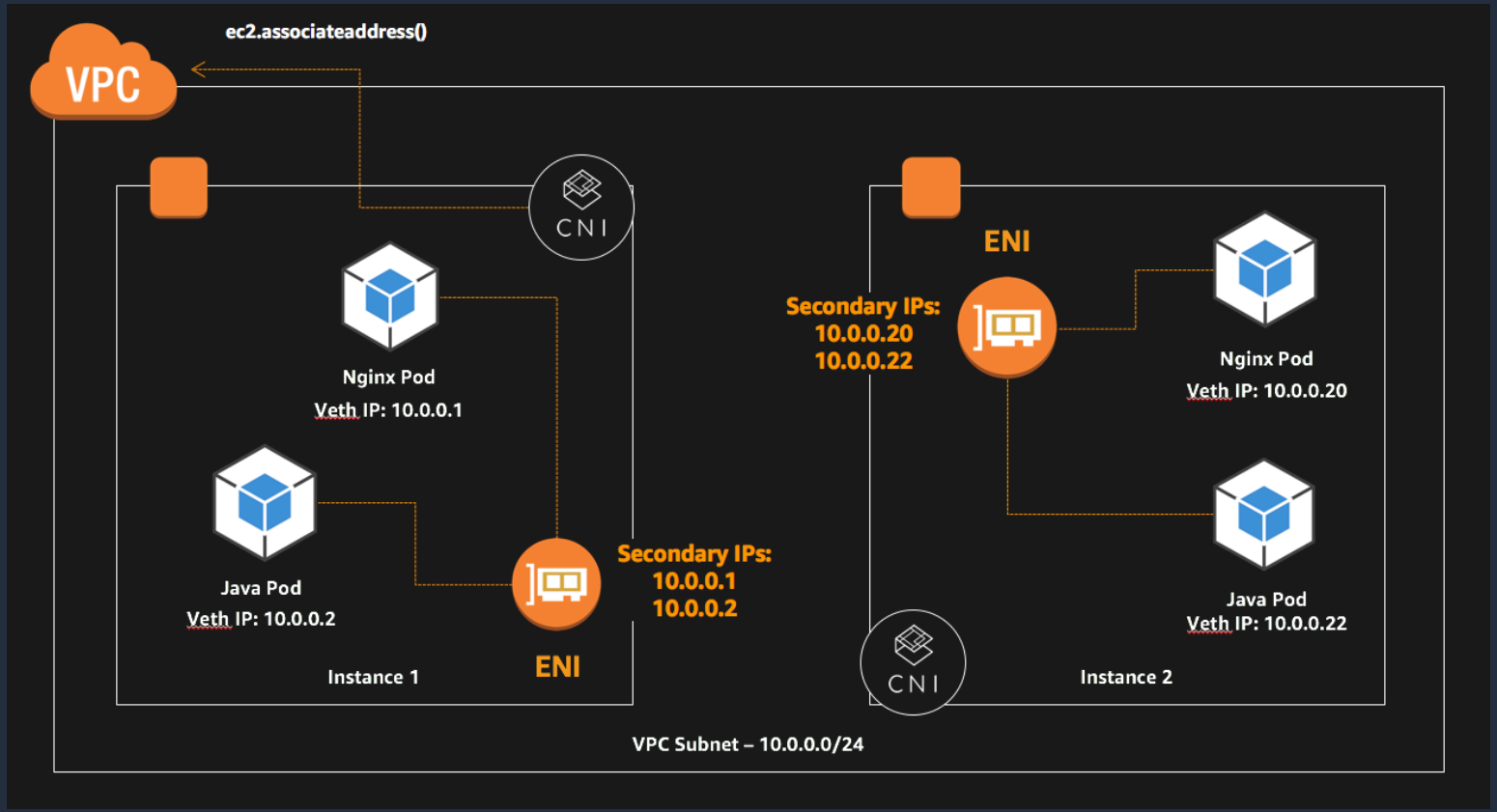
#EKSMatsuri

Amazon EKS 運用編

aws

Amazon VPC CNI plugin の仕組みとパラメーター

#EKSMatsuri

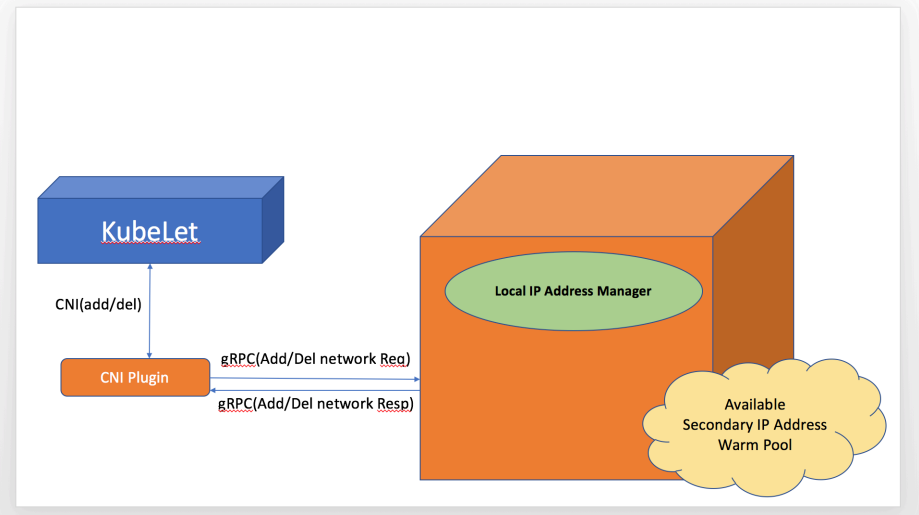


aws

Amazon VPC CNI plugin の仕組みとパラメーター

#EKSMatsuri

- L-IPAM (Local IP Address Manager)
  - 各ホストで利用可能なセカンダリIPアドレスのウォームプールを保持し、Podが追加されたら割当を行う



  - VPCフローログを利用し各Podの送受信されるトラフィックをモニタリング可

aws

Amazon VPC CNI plugin の仕組みとパラメーター

#EKSMatsuri

- 割当可能なIPの数はインスタンスタイプに依存
  - https://docs.aws.amazon.com/ja_jp/AWSEC2/latest/UserGuide/using-eni.html#AvailableIpPerENI
  - 上限＝割当最大数（ENI数 * ENI毎のIP数） - 1（ホスト）
- クーリング期間に注意
  - Pod 削除後、IPアドレスは一定期間 Pod に再割当されない
  - Github にある data_store.go によると現在は以下の設定がされている
  - addressCoolingPeriod = 30 * time.Second
- WARM_IP_TARGET に注意
  - WARM_IP_TARGETに指定されている数だけPoolを行うため意図せずIPが確保
- MINIMUM_IP_TARGET
  - スケールする Pod の数が予測可能な場合、こちらに設定し WARM_IP_TARGETの指定数を減らす事で、常にWARM_IP_TARGETより指定数分 IPが 確保されるのを防ぐ
- CNI Metrics Helper
  - ENI と IP アドレス情報を収集しAmazon CloudWatch にメトリクスを発行するツール
  - CloudWatchでダッシュボードを作成しCNIの状態を可視化

マネージド型ノードグループの更新　https://docs.aws.amazon.com/ja_jp/eks/latest/userguide/update-managed-node-group.html
CNI Metrics Helper　https://docs.aws.amazon.com/ja_jp/eks/latest/userguide/cni-metrics-helper.html

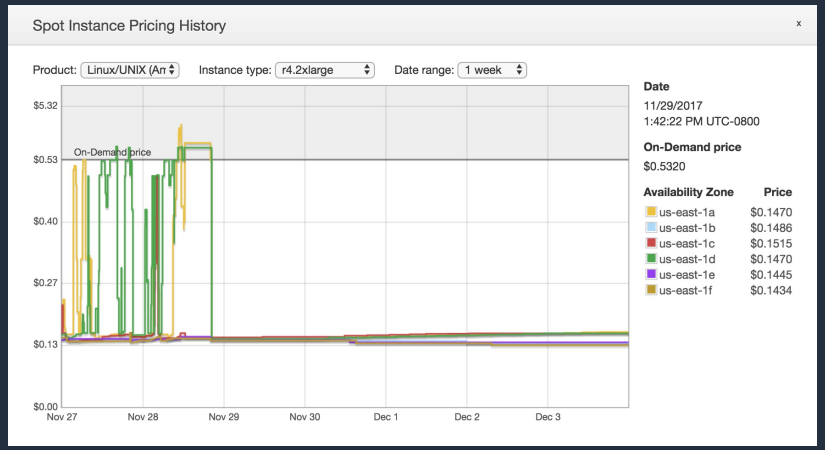
aws

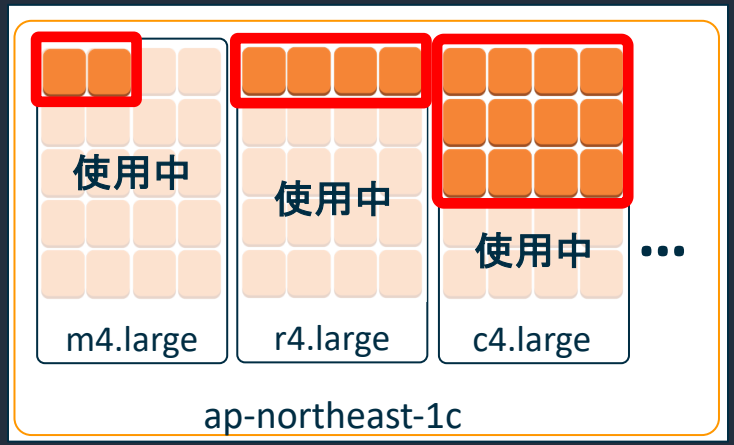
Spot Instance と中断への対応

#EKSMatsuri

スポットインスタンスとは

- Amazon EC2の空きキャパシティを活用し、オンデマンドと比べて最大90%値引き
- スポットインスタンスの価格は長期供給と需要に基づいて徐々に調整
- スポットインスタンスは AWS によって中断されることがありその際は 2 分前に通知
    - Amazon EC2の空きキャパシティが使用できなくなったとき
    - お客様がリクエストした価格(「上限価格」)をスポットインスタンス価格が上回ったとき
- 現在は Unmanaged nodes でのみ利用可能（利用する際は複数のインスタンスタイプ指定を推奨）





aws

Spot Instance と中断への対応

#EKSMatsuri

- AWS Node Termination Handler
  - EC2 メタデータを監視し中断通知が行われたらNodeに対してdrainを実施
  - EC2のメンテナンス予定を検知してNodeに対してcordonを実施
  - Webhook を利用してSlackへの通知
  - Helm対応
  - オープンソース

- 上記を利用してSpot Insnanceの終了を行う
  - 最大でも2 分後には Termination されるため drain を受け取ってから正常終了するまでの時間に注意

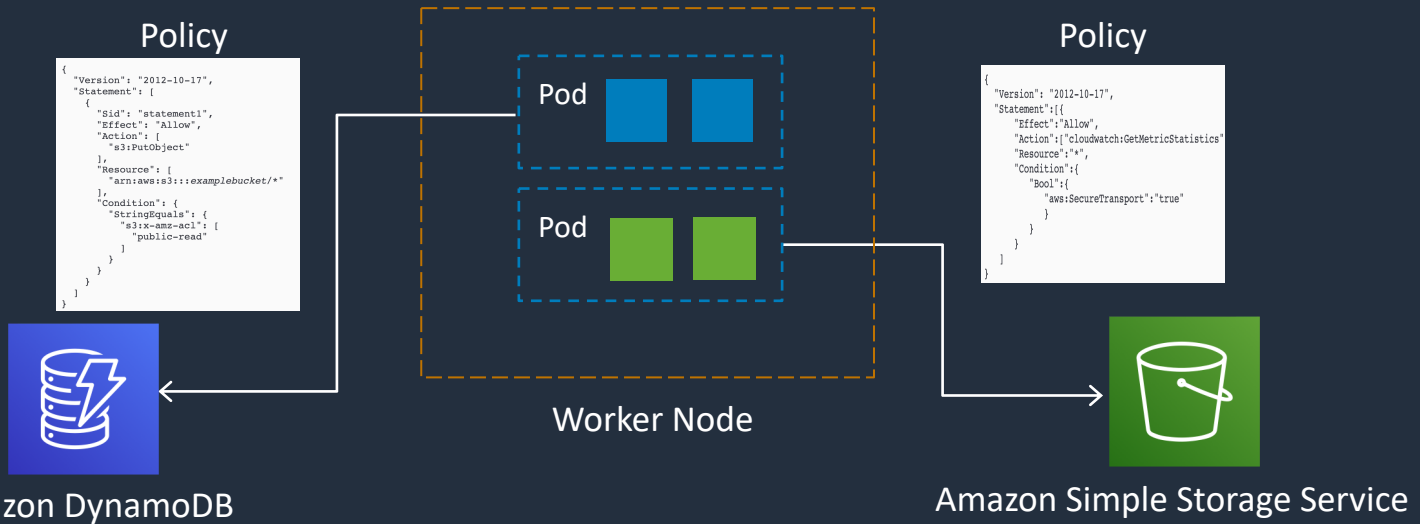
aws-node-termination-handler https://github.com/aws/aws-node-termination-handler

aws

Pod に IAM ロールを設定するには？

#EKSMatsuri

- デフォルトでは、ワーカーノードに設定されている IAM ロールの権限を使用
  - Node上で動作する様々なPodが必要とする全権限が必要となり、それぞれの Pod が自分が必要としない権限まで有しているといったセキュリティ上のリスクがある状態
- サービスアカウントのIAMロール (IAM Roles for Service Accounts) を設定し Pod は必要最小限の権限を有した IAM ロールの権限を使用（設定方法の詳細は本スライド下部のURLを参照）することを推奨



サービスアカウントの IAM ロール https://docs.aws.amazon.com/ja_jp/eks/latest/userguide/iam-roles-for-service-accounts.html

aws

Pod に IAM ロールを設定するには？

#EKSMatsuri

サービスアカウントのIAMロールを設定した以降、Pod がワーカーノードに設定されているIAMロールの権限を使わないようにインスタンスプロファイルへのアクセスを iptables 設定を変更してブロックする事を推奨

既存のワーカーノードで次のコマンドを（root として）実行
新規のワーカーノードに対してもユーザーデータに以下スクリプトを使用し起動時に設定実行

```
yum install -y iptables-services
iptables --insert FORWARD 1 --in-interface eni+ --destination 169.254.169.254/32 --jump DROP
iptables-save | tee /etc/sysconfig/iptables
systemctl enable --now iptables
```

Amazon EC2 インスタンスプロファイルの認証情報へのアクセスの制限
https://docs.aws.amazon.com/ja_jp/eks/latest/userguide/restrict-ec2-credential-access.html

aws

Amazon EKS コントロールプレーンのログ記録

#EKSMatsuri

- コントロールプレーンのログ記録を有効にすると、CloudWatch Logs に Amazon EKS クラスターごとロググループが作成されログが送信
    - CloudWatch Logs データの取り込みおよび保存の費用が発生
- 新規または既存のクラスターごと、以下の各ログタイプを有効/無効に設定
    - API サーバー
        - クラスターへの Kubernetes API リクエストに関するログ
    - 監査
        - Kubernetes API を介したクラスターアクセス（個々のユーザー、管理者、またはシステムコンポーネント）に関するログ
    - 認証 (Authenticator)
        - Amazon EKS 固有のログ、クラスターへの認証リクエスト（Kubernetes RBAC と IAM 認証情報を用いた認証）に関するログ
    - コントローラーマネージャー
        - クラスターコントローラーの状態に関するログ
    - スケジューラ
        - Podの配置（スケジュール）に関するログ

Amazon EKS コントロールプレーンのログ記録　https://docs.aws.amazon.com/ja_jp/eks/latest/userguide/control-plane-logs.html

aws

アプリケーションログを扱うには？

#EKSMatsuri

- ログコレクタを利用して取得
    - オープンソースの Fluentd および Fluent Bit を利用して Amazon Kinesis Data Firehose, Amazon Kinesis Data Streams, Amazon CloudWatch に対して送信可能
- ログは、異常検知のアラート目的でリアルタイム性が求められるもの、ニアリアルタイムに可視化が必要なもの、分析用途で収集のタイミングよりロストするリスクを最小化する方が良い、といったワークロードにより扱いが異なるため AWSの複数のマネージドサービスからワークロードに合わせて選択し組み合わせる
- よく見られる構成
    - ログをストアしたい
        - Amazon Kinesis Data Firehose -> Amazon S3
    - ログを分析したい
        - Amazon Kinesis Data Firehose -> Amazon S3 -> Amazon Athena (or Redshift Spectrum)
        - Amazon Kinesis Data Firehose -> Amazon Elastcisearch Service (+ Kibana)
        - Amazon Kinesis Data Firehose -> Amazon Redshift
        - Amazon Kinesis Data Streams -> Kinesis Data Analytics
    - ログの監視をしたい
        - Amazon CloudWatch (-> Amazon S3)
- Amazon Kinesis Data Firehose の前に Amazon Data Streams を置く事で、最大7日データを保存したり、ストリーミングデータに対するカスタムの処理が出来たりと、運用後の変更が柔軟にできる事から挟む構成もあり
- Fluent Bit を AWS がサポートし AWS for Fluent Bit として各リージョンのECRにイメージ登録済

Fluent Bit による集中コンテナロギング　https://aws.amazon.com/jp/blogs/news/centralized-container-logging-fluent-bit/
AWS for Fluent Bit イメージの使用　https://docs.aws.amazon.com/ja_jp/AmazonECS/latest/developerguide/using_firelens.html#firelens-using-fluentbit
AWS for Fluent Bit による Kubernetes ロギング　https://aws.amazon.com/jp/blogs/news/kubernetes-logging-powered-by-aws-for-fluent-bit/

aws

セキュアストリングをAWSで管理するには？

#EKSMatsuri

KubernetesのSecretsではなくAWS Secrets Manager シークレットまたは AWS Systems Managerパラメータストア を使う場合のパラメーター取得パターン例

- コンテナイメージがAWS SDK を含んでいる場合には直接 API にて取得
- InitContainers で AWS SDK を含むコンテナイメージから AWS Secrets Manager シークレットまたは AWS Systems Manager パラメータストア に対して API で結果を取得しVolume に出力 、その後アプリケーションコンテナ側で Volume から取得

```
...
spec:
  initContainers:
  - name: init-myservice
    image: amazonlinux:2
  containers:
  - name: myapp-container
    image: myapp:1.2
...
```

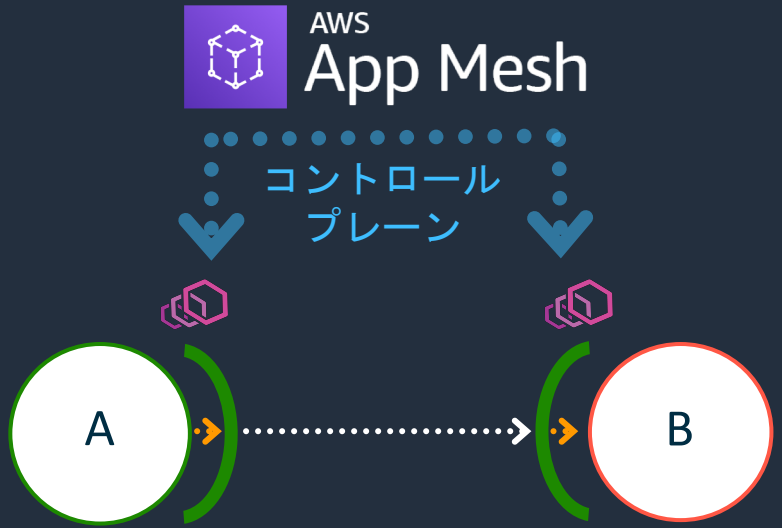
- エコシステム を利用し取得（例: Kubernetes External Secrets　オープンソースのためサポート対象外）

パラメータストア パラメータからの AWS Secrets Manager シークレットの参照
https://docs.aws.amazon.com/ja_jp/systems-manager/latest/userguide/integration-ps-secretsmanager.html

aws

その他

AWS App Meshとは



App Mesh はサービスメッシュのコントロールプレーン

- メッシュ全体構造の定義
- メッシュを構成するプロキシに対し動的に設定を配信

AWS App Meshとは

AWS App Mesh

追加料金なしで使用可能

アプリケーションレベルのネットワーク

- ログ・メトリクス・トレース情報の容易な出力
- クライアントサイドのトラフィック・ルーティングポリシー

クラスタやサービスにまたがるメッシュの構築

- Amazon ECS
- AWS Fargate
- Amazon EKS
- Kubernetes on EC2
- Amazon EC2

マネージドコントロールプレーン

- 容易なオペレーション
- 高いスケーラビリティ

https://aws.amazon.com/jp/app-mesh/

aws

Amazon CloudWatch Container Insightsとは

#EKSMatsuri

- コンテナ化されたアプリケーションのメトリクスとログを収集、集計、要約できるCloudWatchの機能の一つ
- CloudWatchにてタスク、コンテナレベルでのモニタリングが可能
- Container Insights が収集するメトリクスは 自動的に作成される
  - ダッシュボードに集約され、より鋭い洞察を行うことが可能
- AWSが提供するコンテナオーケストレーションツールであるAmazon ECSや、Amazon EKS、および Amazon EC2 の Kubernetes プラットフォームでご利用可能

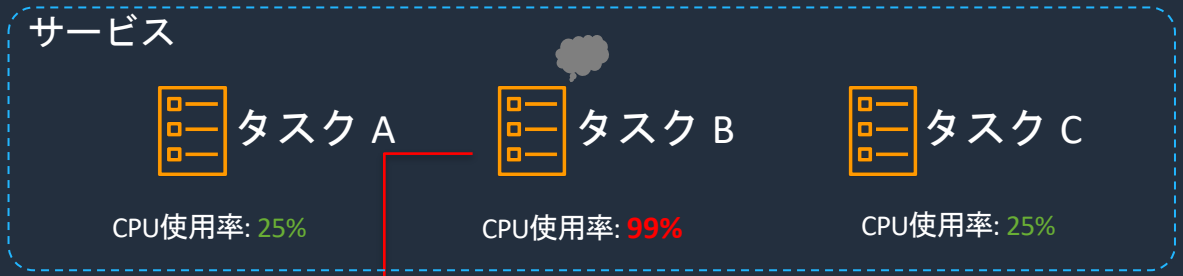
  ※ 2020/04/30 現在、AWS Batch で Container Insights は未サポート

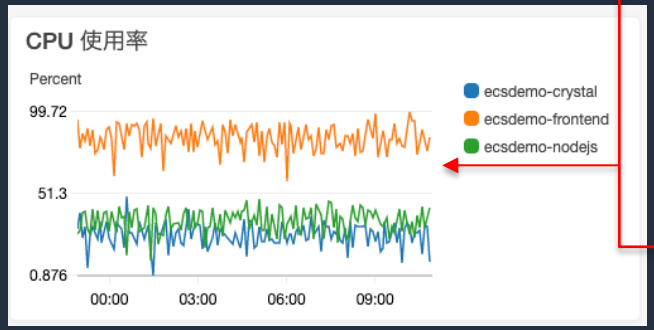
aws

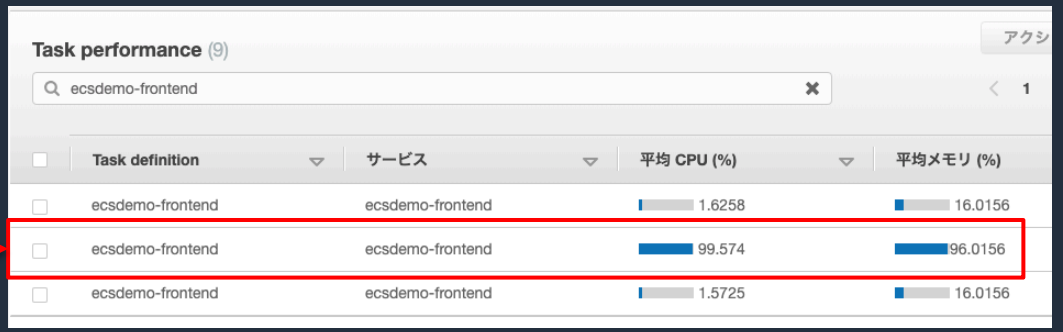
Amazon CloudWatch Container Insightsとは

#EKSMatsuri

- Amazon CloudWatchと統合された、タスクやコンテナレベルでメトリクスやログを取得することが可能







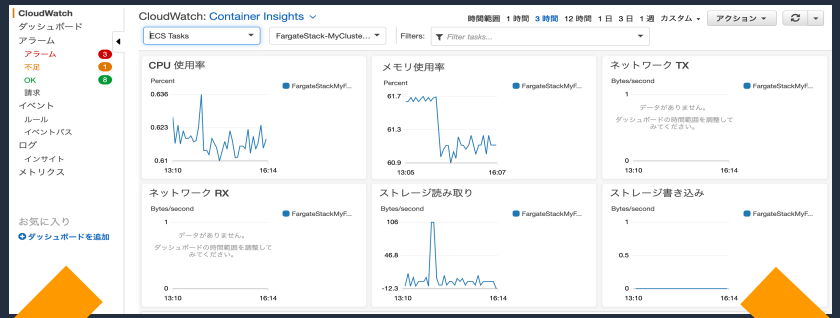
Amazon CloudWatch Container Insightsとは

#EKSMatsuri

- 自動ダッシュボードにてクラスタ単位、サービス単位、タスク単位、コンテナ単位と様々な角度で可視化し、メトリクスを確認しながら詳細な分析や調査が可能
- CloudWatch Logs InsightsやX-Rayとも統合されている
- Container Insightsのダッシュボードを起点により詳細な分析が可能

CloudWatch Logs Insightsの要件の例

- グラフのより詳細な値を見たい
- ログに対し分析クエリを発行したい



Container Insights

X-Ray を使う要件の例

- タスク間の通信をトレースしたい

Amazon CloudWatch Logs Insights

AWS X-Ray

EKS の SLA /コンプライアンス対応状況は？

#EKSMatsuri

SLA:99.95 %
利用できる全リージョンが対応
下回った場合にはその稼働率に応じたクレジットをリクエストにより付与

コンプライアンス（サードパーティーである独立した監査人によって評価）

- HIPAA 適合
- SOC 準拠
- ISO 9001, 27001, 27017, 27018 準拠
- PCI DSS Level 1 準拠

Amazon EKS Service Level Agreement　https://aws.amazon.com/jp/eks/sla/
コンプライアンスプログラムによる AWS 対象範囲内のサービス　https://aws.amazon.com/jp/compliance/services-in-scope/

aws

コストを削減するには？

Spot Instance 以外の選択肢として長期利用が見込まれる場合には Savings Plans を検討

Savings Plansの特徴

- 1年もしくは3年間での1時間あたりの利用金額をコミットすることでEC2、Lambda、Fargateの利用金額を割引
    - コミットした利用金額のボリュームを超えた利用量についてはオンデマンド料金で計算

Savings Plansの選択肢

- Compute Savings Plans（最大 66% 削減）
    - インスタンスファミリー、サイズ、アベイラビリティーゾーン、リージョン、OS、またはテナンシーに関わらずEC2、Fargateに適用
- EC2 Instance Savings Plans（最大 72% 削減）
    - リージョン内で、アベイラビリティーゾーン、サイズ、OS、またはテナンシーに関わらずEC2に適用

Savings Plans の料金　https://aws.amazon.com/jp/savingsplans/pricing/

サービスクォータ

#EKSMatsuri

- 変更可能な AWS アカウントの Amazon EKS のデフォルトのクォータ

リソース	デフォルトのクォータ
Amazon EKS クラスターの最大数 (1 リージョン、1 アカウントあたり)	100
クラスターあたりのマネージド型ノードグループの最大数	10
マネージド型ノードグループあたりのノードの最大数	100
クラスターあたりの Fargate プロファイルの最大数	10
Fargate プロファイルあたりの最大セレクター数	5
Fargate プロファイルセレクターあたりのラベルペアの最大数	100
同時 Fargate ポッドの最大数 (リージョンごと、アカウントごと)	100
1 秒あたりの Fargate ポッド起動の最大数 (リージョンごと、アカウントごと)	1 (最大 10 までの一時バーストあり)

- 変更不可なAmazon EKS のクォータ

リソース	デフォルトのクォータ
クラスターあたりのコントロールプレーンセキュリティグループの最大数 (クラスター作成時に指定)	4
クラスターあたりのパブリックエンドポイントアクセスの CIDR 範囲の最大数	40

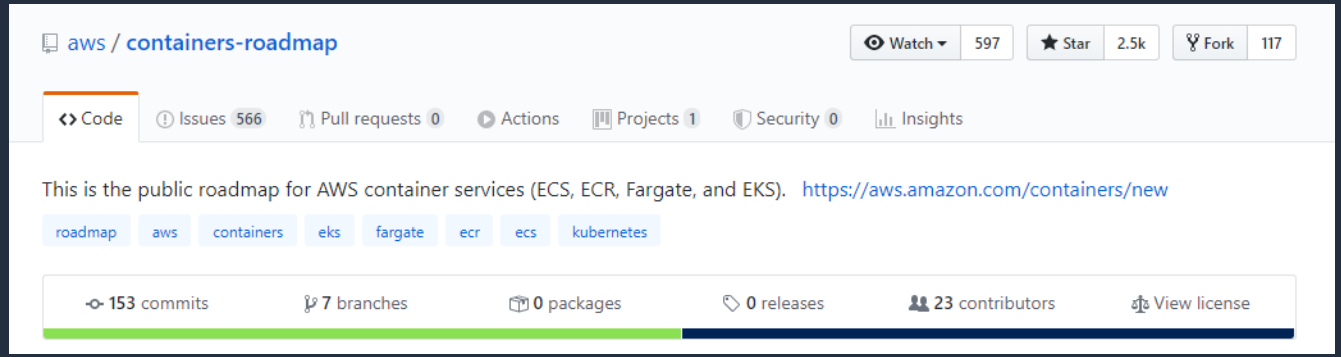
Amazon EKS サービスクォータ https://docs.aws.amazon.com/ja_jp/eks/latest/userguide/service-quotas.html

aws

Roadmap 情報の入手や機能のリクエストをしたい #EKSMatsuri

Roadmap

- コンテナサービス（ECS, ECR, Fargate, EKS）に関するパブリックなRoadmapは github で公開
- フィードバックや機能リクエストの Issue を作成可



containers-roadmap https://github.com/aws/containers-roadmap

Amazon EKS を学ぶために役立つ情報

#EKSMatsuri

Black Belt オンラインセミナー

- AWS 各サービスやソリューションに関するWebinarを実施
- 過去開催したWebinarに関しても「Black Belt + 知りたいサービス名」で検索

Amazon EKS Workshop

- 各自の AWS 環境で EKS 環境を構築しアプリケーションのデプロイ、CI/CD 環境、サービスメッシュなど体系的に学ぶためのコンテンツ

AWS Webinar スケジュール　https://aws.amazon.com/jp/about-aws/events/webinars/
Amazon EKS Workshop　https://eksworkshop.com/

Thank You!