Amazon ECS、 Amazon EKS でのマルチ CPU アーキテクチャコンテナの実行

黄　光川

Amazon Web Services Japan G.K.
Solutions Architect

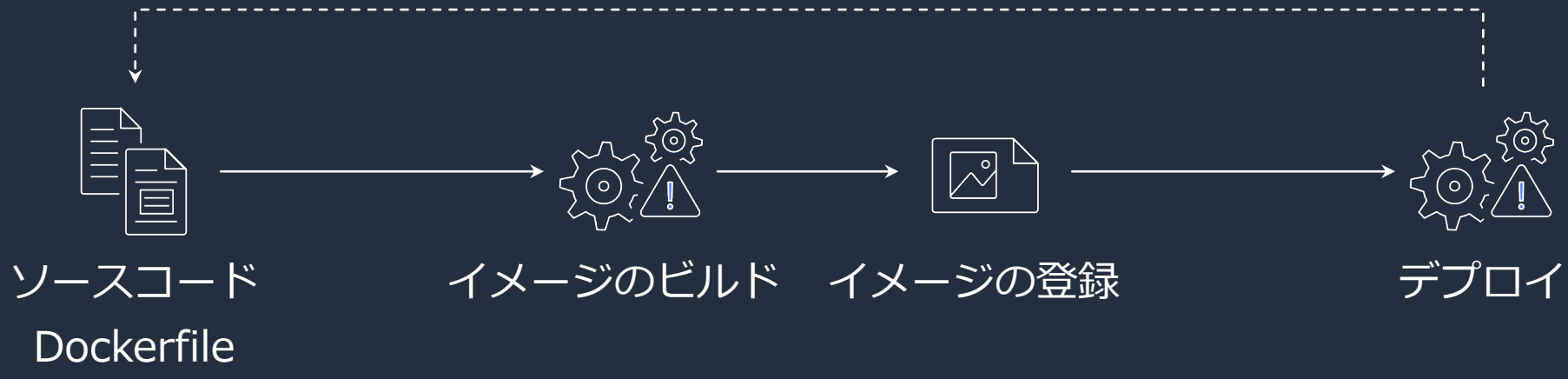
自己紹介

- 名前：
  黄　光川（コウ　コウセン）
- 所属：
  アマゾン ウェブ サービス ジャパン 合同会社
  技術統括本部 / インターネットメディアソリューショングループ
  ソリューションアーキテクト
- 経歴：

  SIer -> iDC -> AdTech -> AWS
- 好きなAWSサービス：
  Amazon ECS, Amazon EKS など
  コンテナ系のサービス

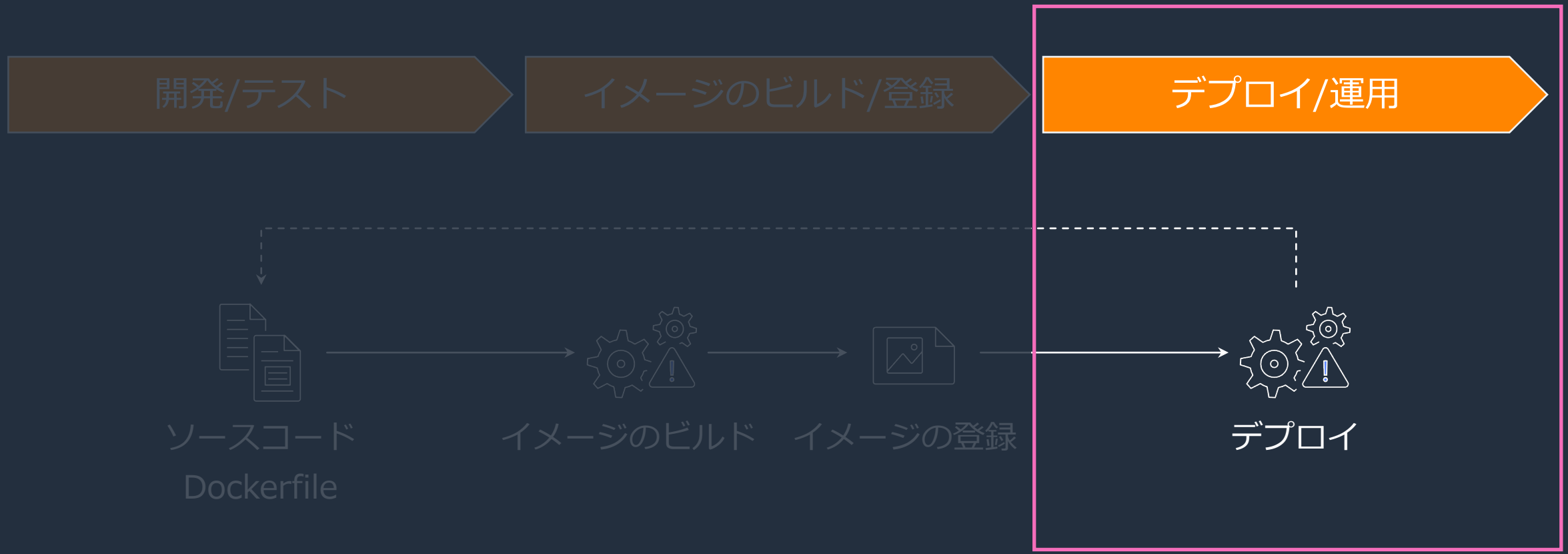
コンテナにおけるソフトウェア開発ライフサイクル

開発/テスト → イメージのビルド/登録 → デプロイ/運用



本セッションでお話すること

Amazon ECS, Amazon EKS で Graviton ベースのコンピュートを
利用する方法について紹介します。



本セッションでお話すること

Amazon ECS, Amazon EKS で x86_64(intel, AMD) と arm64(ARM) の利用・移行

Amazon ECS

- Capacity Providers を利用した Graviton インスタンスの利用、移行方法

Amazon EKS

- Karpenter を利用した Graviton インスタンスの利用、移行方法

本セッションでお話すること

Amazon ECS, Amazon EKS で x86_64(intel, AMD) と arm64(ARM) の利用・移行

Amazon ECS

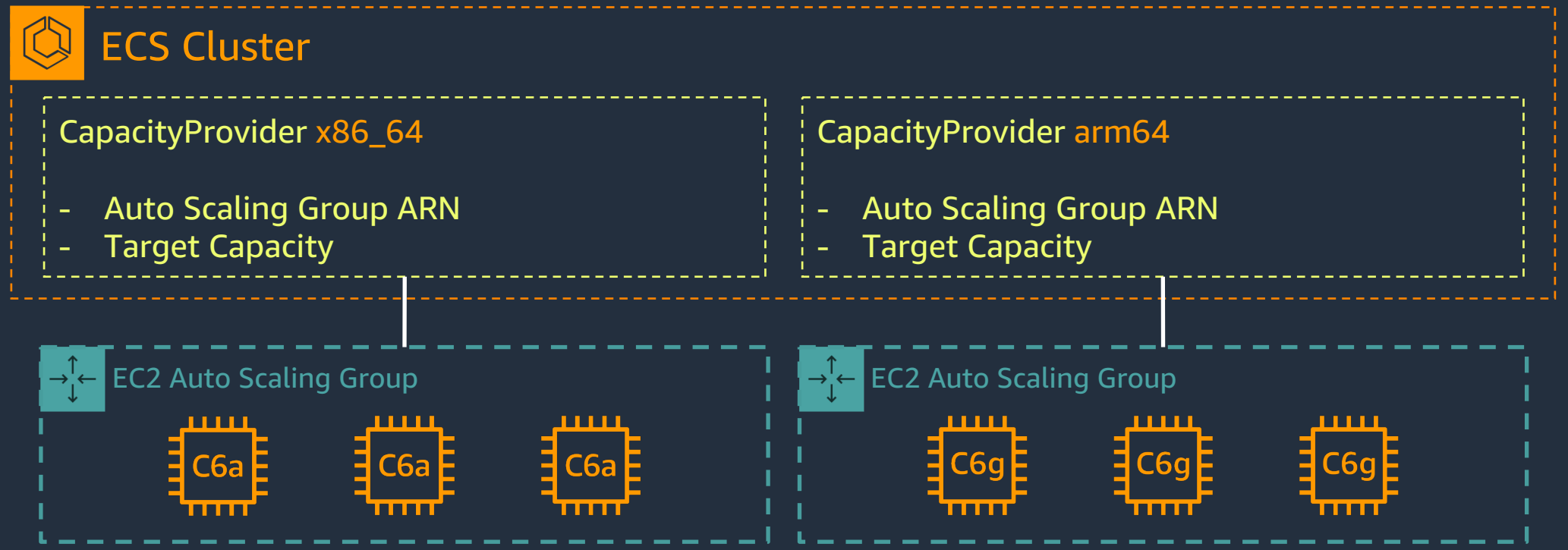
- Capacity Providers を利用した Graviton インスタンスの利用、移行方法

Amazon EKS

- Karpenter を利用した Graviton インスタンスの利用、移行方法

Capacity Provider とは

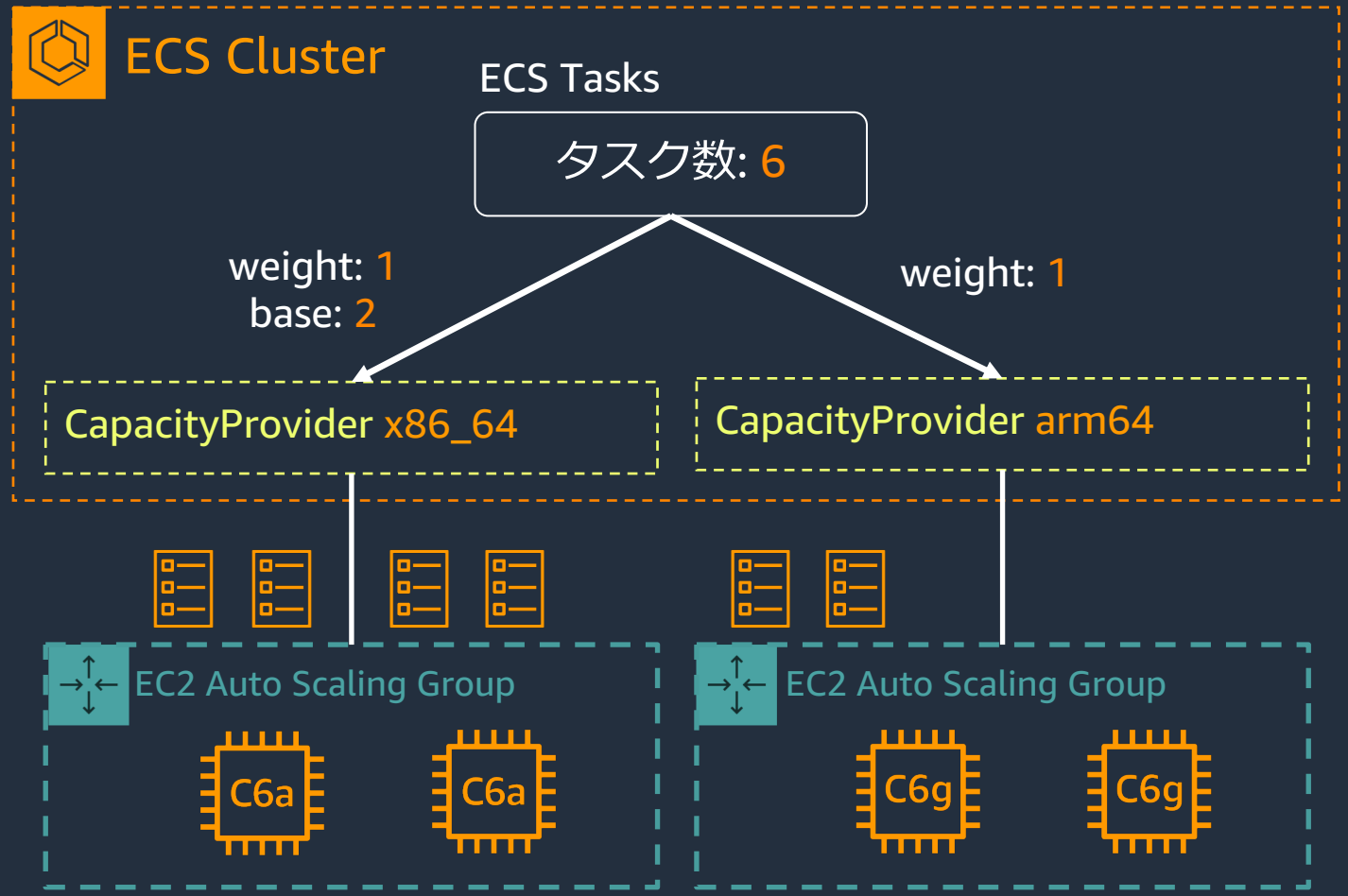
- クラスター内のタスクに対するインフラストラクチャのスケーリングを管理
- 各クラスターには、１つ以上のキャパシティプロバイダーを設定可能



ECS Capacity Providers についての詳細はこちらを参照
https://aws.amazon.com/jp/blogs/news/aws-black-belt-online-seminar-con207-and-con307/

Capacity Provider Strategy

- weight
  - 実行するタスクの総数に対する指定した Capacity Provider での相対的な割合
- base
  - その Capacity Provider での最小実行タスク数




```
$ aws ecs run-task
    --cluster MyEcsCluster
    --task-definition task-def-family:revision
    --count 6
    --capacity-provider-strategy
      capacityProvider=x86_64,weight=1,base=2
      capacityProvider=arm64,weight=1
```

ECS Capacity Providers についての詳細はこちらを参照
https://aws.amazon.com/jp/blogs/news/aws-black-belt-online-seminar-con207-and-con307/

Capacity Providers を利用した Graviton への移行

Capacity Providers を利用した Graviton インスタンスの利用、移行方法

1. 起動テンプレート作成 x 2 (x86_64, arm64 用)
   a. ECS-optimized AMI の確認
   b. aで確認したAMI指定
   c. インスタンスタイプの属性指定 (CPU 製造元: Amazon, AMD, intel)
   d. ECS Cluster に参加する為のユーザデータの追加
2. Auto Scaling Group の作成 x 2 (x86_64, arm64 用)
   a. 1で作成した起動テンプレート指定
3. ECS で Capacity Providers を作成 x 2 (x86_64, arm64 用)
   a. 2で作成した Auto Scaling Group を指定
4. ECS タスク定義作成 (注：aws cli, SDK で作成)
5. ECS サービス・タスクの作成

Amazon ECS で Capacity Providers の利用方法

Capacity Providers を利用した Graviton インスタンスの利用、移行方法

1. 起動テンプレート作成 x 2 (x86_64, arm64 用)
   a. ECS-optimized AMI の確認
   b. aで確認したAMI指定
   c. インスタンスタイプの属性指定 (CPU 製造元: Amazon, AMD, intel)
   d. ECS Cluster に参加する為のユーザデータの追加

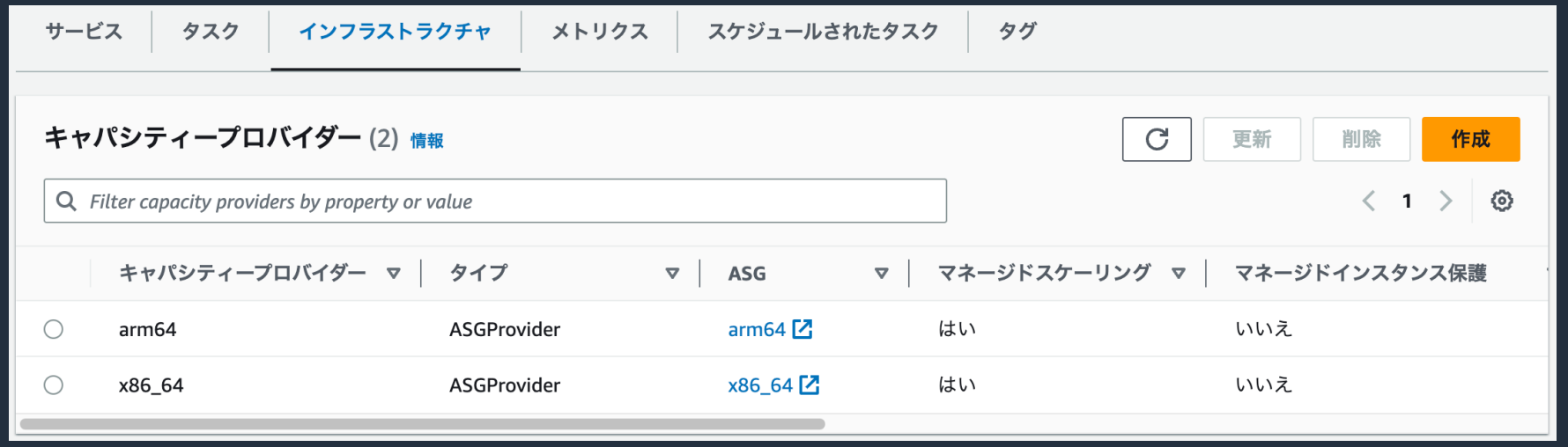

```
# ユーザデータに ECS_CLUSTER を設定
#!/bin/bash
echo ECS_CLUSTER=cluster >> /etc/ecs/ecs.config
sudo iptables --insert FORWARD 1 --in-interface docker+
\ --destination 169.254.169.254/32 --jump DROP
sudo service iptables save
echo ECS_AWSVPC_BLOCK_IMDS=true >> /etc/ecs/ecs.config
```

https://docs.aws.amazon.com/ja_jp/AmazonECS/latest/developerguide/launch_container_instance.html

Amazon ECS で Capacity Providers の利用方法

Capacity Providers を利用した Graviton インスタンスの利用、移行方法

2. Auto Scaling Group の作成 x 2 (x86_64, arm64 用)
   a. 1で作成した起動テンプレート指定
3. ECS で Capacity Providers を作成 x 2 (x86_64, arm64 用)
   a. 2で作成した Auto Scaling Group を指定



Amazon ECS で Capacity Providers の利用方法

Capacity Providers を利用した Graviton インスタンスの利用、移行方法

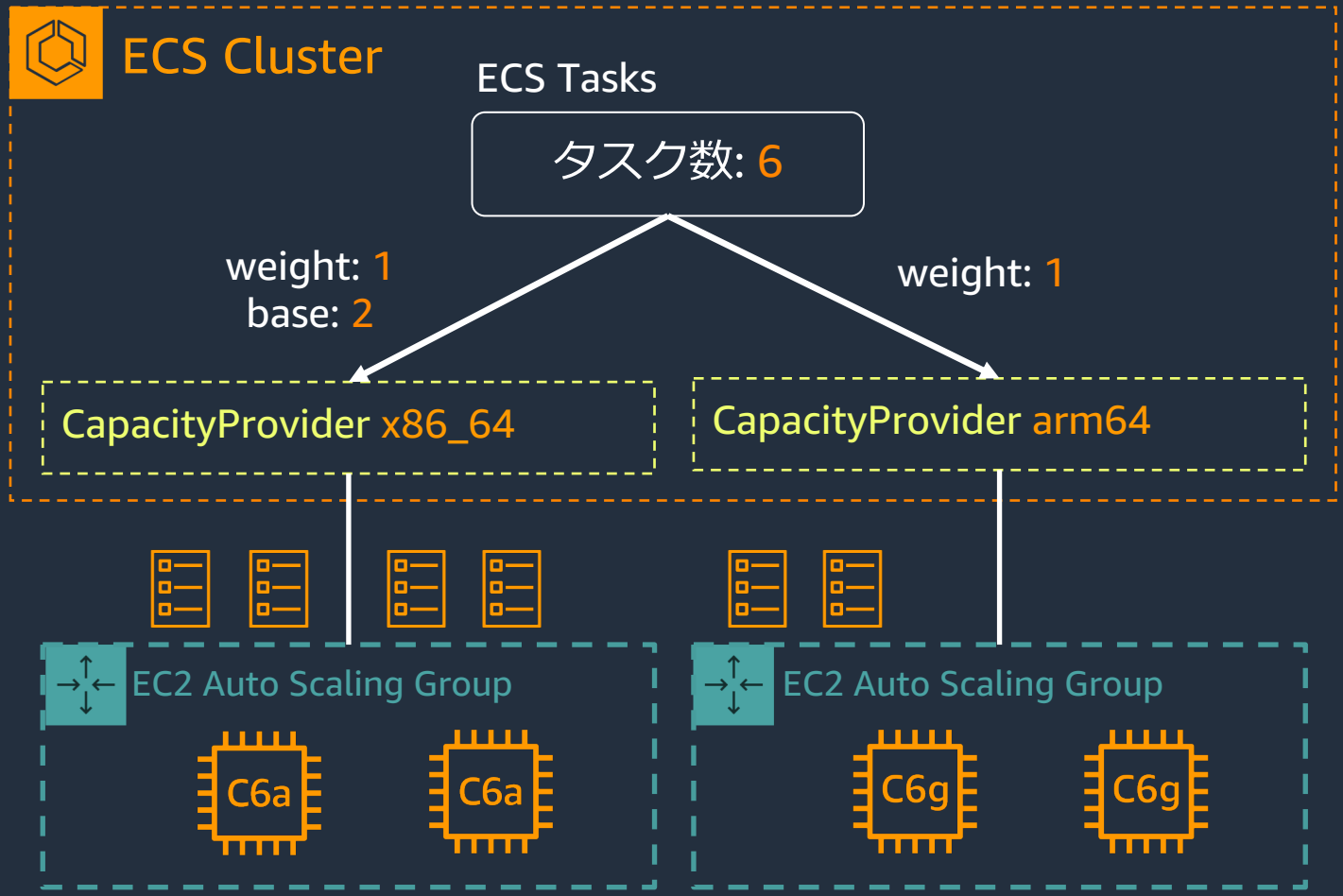
4. ECS タスク定義作成 (注：aws cli, SDK で作成)
5. ECS サービス・タスクの作成

```
aws ecs register-task-definition \
 --cli-input-json file://<path_to_json_file>/taskdef.json
```

注：EC2 の場合、ECS タスク定義は aws cli 若くは SDK などから
runtimePlatform を指定せず作成する
Fargate の場合、runtimePlatform は必須なので、 X86_64 と ARM64 用に
それぞれタスク定義を作成する

x86_64 から arm64 (Graviton) への切替ステップ

1. Capacity Providers を x86_64 と arm64 用に作成
2. weight を変更しながら、x86_64 と arm64 のタスクの割合を調整
   (一定期間並行運用し、検証・評価)
3. x86_64 用の Capacity Providers の base, weight を 0 にする。
   (切替完了)



デモ

本セッションでお話すること

Amazon ECS, Amazon EKS で x86_64(intel, AMD) と arm64(ARM) の利用・移行

Amazon ECS

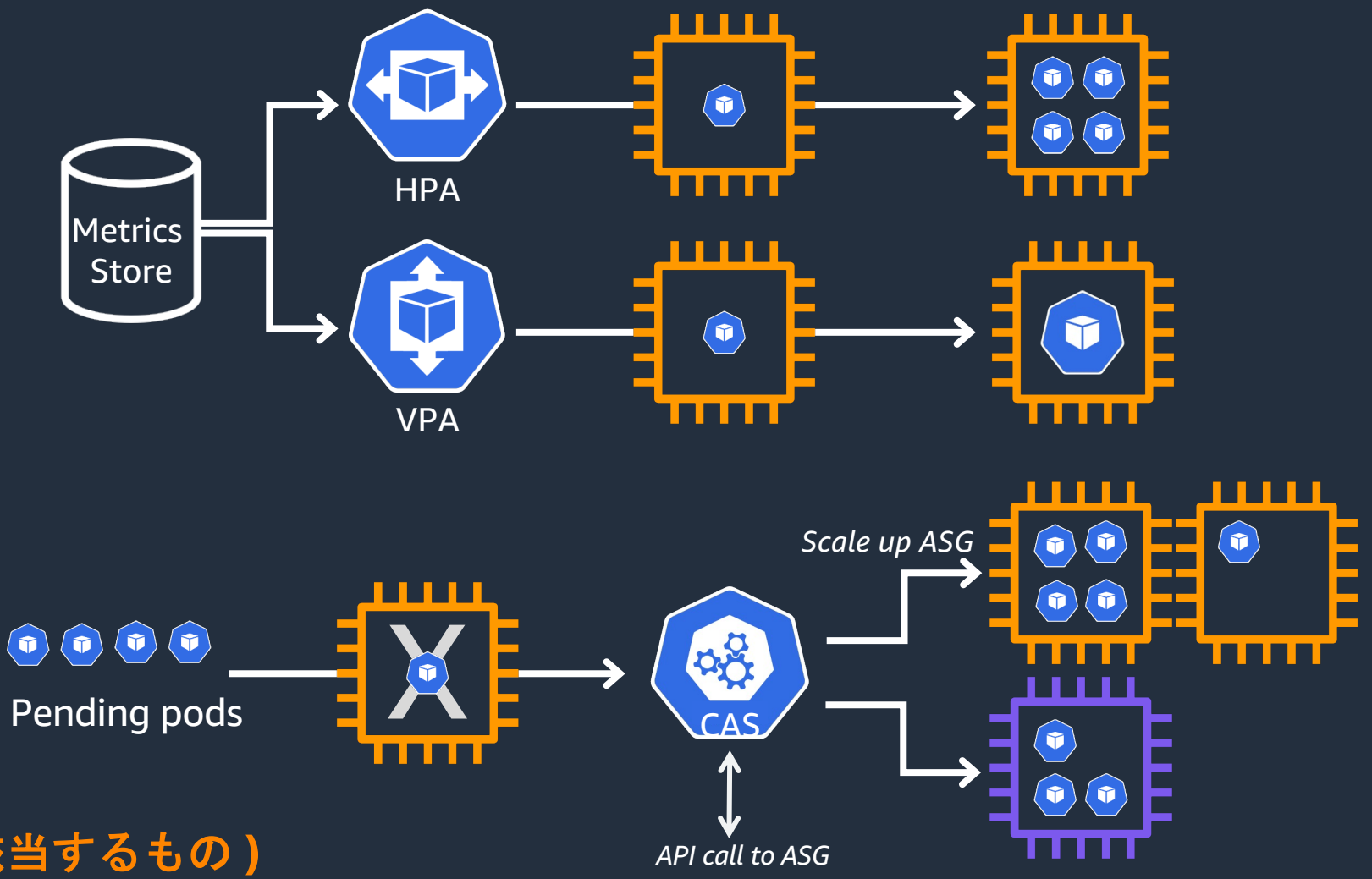
- Capacity Providers を利用した Graviton インスタンスの利用、移行方法

Amazon EKS

- Karpenter を利用した Graviton インスタンスの利用、移行方法

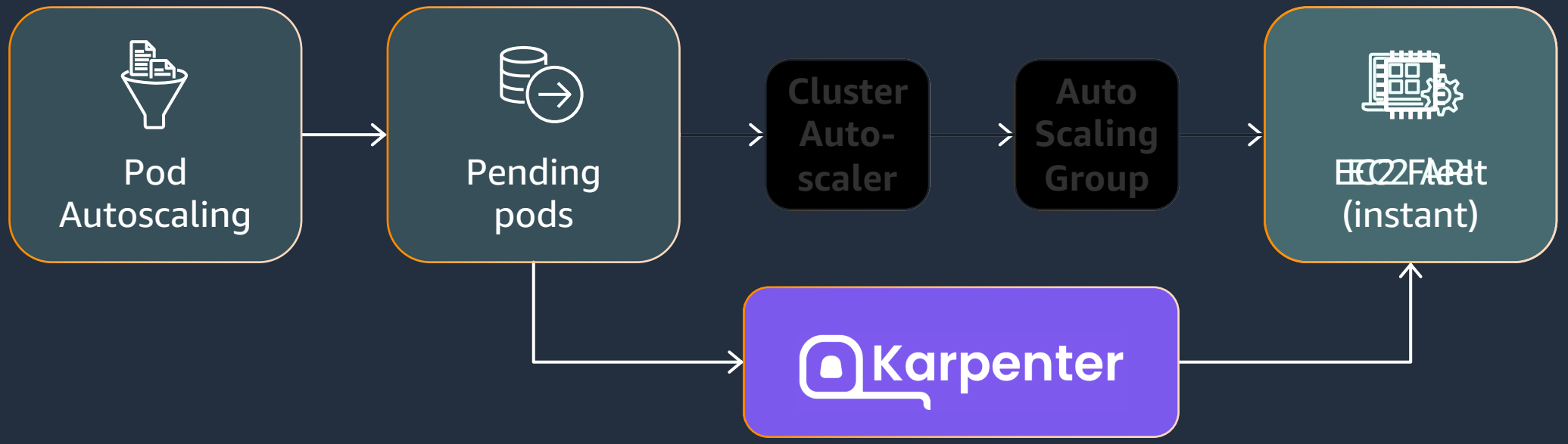
Kubernetes Autoscaling

1. Horizontal Pod Autoscaling (HPA)
2. Vertical Pod Autoscaling (VPA)
3. Cluster Autoscaler (CAS)



(Karpenter は CAS に該当するもの)

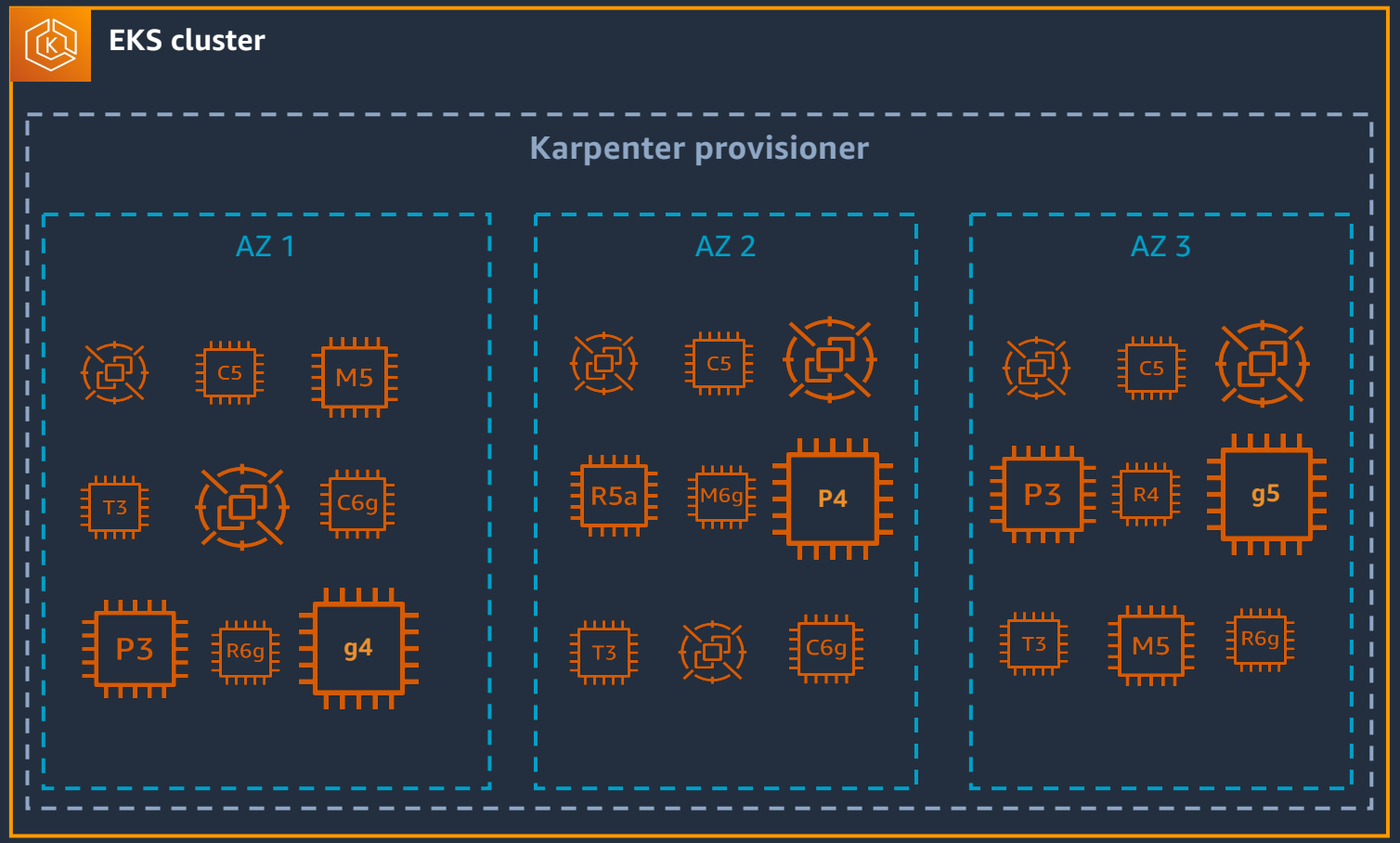
How Karpenter works ?



インスタンスのオーケストレーション責任を
単一システム内に統合

Karpenter によるコンピューティングの柔軟性

Provisioner を使用したノードの自動スケーリング
Provisioner は Kubernetes の Custom Resource Definitions として動作



- 適切なコンピューティングリソースを動的に決定
- 必要に応じてコンピューティングリソースを追加削除
- 大規模な環境でパフォーマンステスト済み
- EKS での利用は AWS サポート対象

Amazon EKS で Graviton への移行・利用方法

Capacity Providers を利用した Graviton インスタンスの利用、移行方法

1. Karpenter のインストール
   a. 新規 EKS クラスターの場合はこちらのドキュメントを参照
      https://karpenter.sh/v0.27.3/getting-started/getting-started-with-karpenter/
   b. 既存の EKS クラスター (Cluster Autoscaler) からの移行はこちらのドキュメントを参照
      https://karpenter.sh/v0.27.3/getting-started/migrating-from-cas/

2. Provisioner の定義

- EKS Blueprints を利用することで検証環境を簡単に構築可能 (Karpenter サンプルあり)
  https://aws.amazon.com/jp/blogs/news/bootstrapping-clusters-with-eks-blueprints/
  https://github.com/aws-ia/terraform-aws-eks-blueprints/tree/main/examples/karpenter

Provisioner の定義

- 不要なノードの削除、サイズの最適化
- インスタンスファミリーの指定 (In で設定)
- インスタンスサイズの指定 （NotIn で除外設定）
- キャパシティタイプの指定
- CPU アーキテクチャの指定 （amd64, arm64)
- ノードテンプレートの指定

```yaml
apiVersion: karpenter.sh/v1alpha5
kind: Provisioner
metadata:
  name: default
spec:
  consolidation:
    enabled: true
  requirements:
    # Include general purpose instance families
    - key: karpenter.k8s.aws/instance-family
      operator: In
      values: [c5, m5, r5]
    # Exclude small instance sizes
    - key: karpenter.k8s.aws/instance-size
      operator: NotIn
      values: [nano, micro, small, large]
    - key: karpenter.sh/capacity-type
      operator: In
      values: ["on-demand", "spot"]
    - key: kubernetes.io/arch
      operator: In
      values: ["amd64", "arm64"]
  providerRef:
    name: default
```

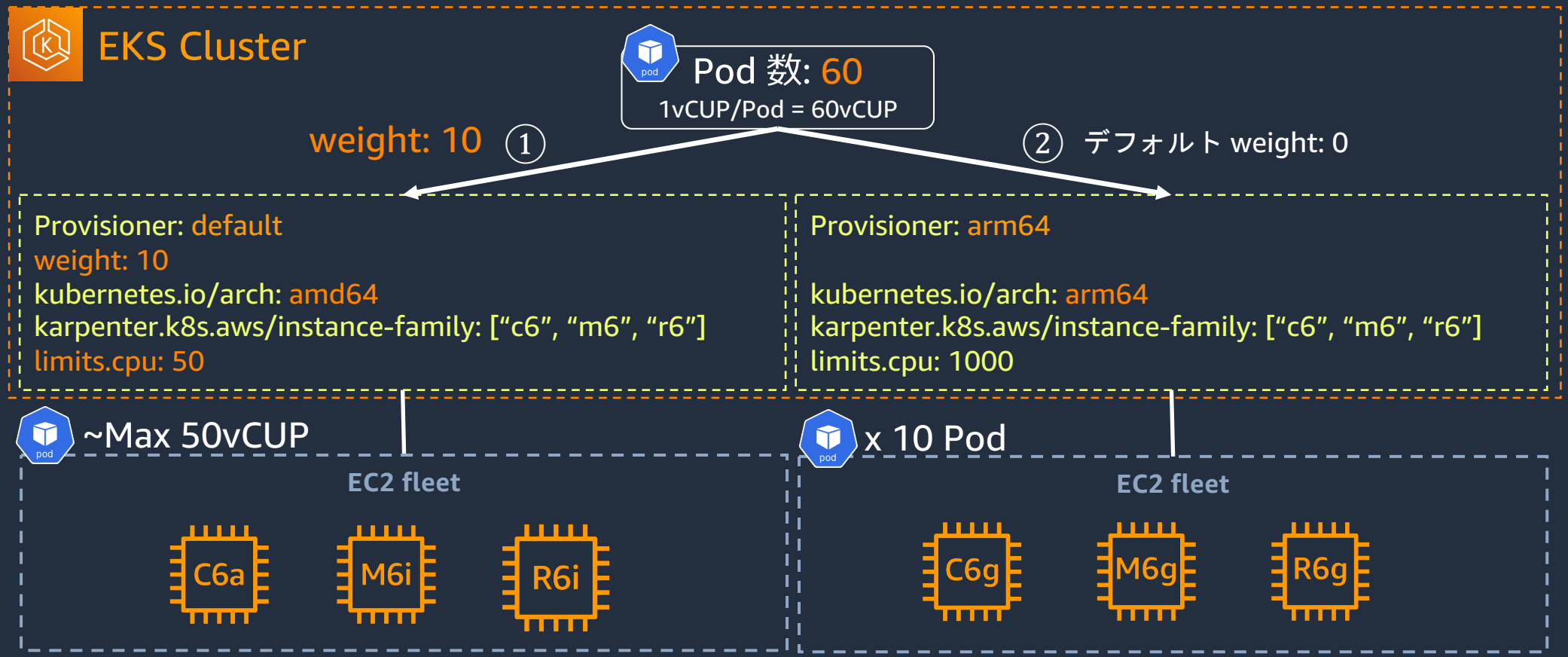
ノードテンプレート定義

- Provisioner の spec.providerRef で ノードテンプレートを指定
- ノードテンプレート (AWSNodeTemplate) を使用して、Provisioner 固有の設定が可能

- spec.subnetSelector (必須)
  - インスタンスに接続するサブネットを指定
- spec.securityGroupSelector (必須)
  - インスタンスに設定するセキュリティグループの設定
- spec.instanceProfile
  - インスタンスプロファイルのオーバラップ
- spec.amiSelector
  - AMI の指定（指定なしの場合 EKS に最適化された AMI を Karpenter が指定する）
- spec.blockDeviceMappings
  - EBS ボリュームの制御
- spec.userData
  - ワーカーノードに適用される UserData を制御
- spec.detailedMonitoring
  - EC2 の詳細モニタリング有効
- spec.tags
  - EC2, EBS などにタグを追加
- status.subnets
  - ワーカーノード起動時に使用される Subnet の指定
- status.securityGroups
  - ワーカーノード用のセキュリティグループ

```yaml
apiVersion: karpenter.k8s.aws/v1alpha1
kind: AWSNodeTemplate
metadata:
  name: default
spec:
  amiFamily: AL2
  subnetSelector:
    karpenter.sh/discovery: my-cluster
  securityGroupSelector:
    karpenter.sh/discovery: my-cluster
  blockDeviceMappings:
  - deviceName: /dev/xvda
    ebs:
      volumeSize: 100Gi
      volumeType: gp3
  tags:
    team: dev
```

Provisioners は複数定義可能



複数の Provisioner の場合、weight が高い方から優先してスケジュールされる。(weight の default 値は 0)
limits は、該当 Provisioner にスケジュールできるリソースの上限を決める。
この性質を利用して、RI/SP を優先したり、x86_64 と arm64 の割合を調整したり、チーム別に容量を決めたり、特定のチームにのみ GPU ノードを用意したりできる。

Provisioners の重み付けを利用した Graviton への移行例

EKS Cluster

```
apiVersion: karpenter.sh/v1alpha5
kind: Provisioner
metadata:
  name: reserved-instance
spec:
  weight: 50
  requirements:
  - key: "node.kubernetes.io/instance-type"
    operator: In
    values: [”c6i.xlarge”]
limits:
    cpu: 80
```

c6i.xlarge は 4vCPU なので、 limits.cpu:80 で20台分確保され、優先して利用される

```
apiVersion: karpenter.sh/v1alpha5
kind: Provisioner
metadata:
  name: default
spec:
requirements:
  - key: ”karpenter.k8s.aws/instance-family"
    operator: In
    values: ["c6”, “m6”, “r6”]
 - key: kubernetes.io/arch
    operator: In
    values: ["amd64”, “arm64”]
 consolidation:
    enabled: true
```

consolidation を true に設定することで、不要なノードを削除し、大きなノードを小さいノードに置き換え、コスト最適化を行う

EC2 fleet

C6i C6i C6i

EC2 fleet

C6i M6g R6g

デモ

まとめ

Amazon ECS, Amazon EKS で x86_64(intel, AMD) と arm64(ARM) の利用・移行

Amazon ECS

- Capacity Providers を複数 (x86_64, arm64) 定義し、Capacity Provider Strategy の weight や base を利用して、使用する CPU アーキテクチャのタスクの比率を調整できる

Amazon EKS

- Karpenter の Provisioner の “kubernetes.io/arch” に [“arm64”, “amd64”] を設定することで、複数の種類の CPU アーキテクチャを混在させることができる。
- Provisioner を複数 (amd64, arm64, etc) 定義し、weight と limits と consolidation を調整することで、割合の調整や RI/SP など必要なリソースを固定できる。

参考資料

- ecsworkshop – Capacity Providers
  https://ecsworkshop.com/capacity_providers/
- Black Belt - CON307 ECS Capacity Providers
  https://pages.awscloud.com/rs/112-TZM-766/images/202109_AWS_Black_Belt_AWS_CON307_ECS_Capacity_Providers.pdf
  https://www.youtube.com/watch?v=45uuyy16RS4

- eksworkshop - Karpenter
  https://www.eksworkshop.com/docs/autoscaling/compute/karpenter/
- Karpenter Best Practices
  https://aws.github.io/aws-eks-best-practices/karpenter/
- Amazon EKS Blueprints for Terraform (Karpenter サンプル)
  https://github.com/aws-ia/terraform-aws-eks-blueprints/tree/main/examples/karpenter

Thank you!

黄　光川

@kousenko