TOKYO

DEV DAY

2019.10.03-04

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

DEV DAY
TOKYO

A - 1

2019.10.03-04

AWS Fargate かんたんデプロイ選手権

Yasuhiro “Tori” Hara

Specialist Solutions Architect, Containers
Amazon Web Services Japan

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Yasuhiro “Tori” Hara

Specialist Solutions Architect, Containers
Amazon Web Services Japan

\---

ERP パッケージベンダー R&D チーム SDE
➡ UI 自動テスト SaaS
➡ クラウド利用の SI + MSP にて、コンテナやサーバーレスによる設計・開発・運用
Web 技術利用のゲームやビジネスアプリケーション、ML/DL 環境構築運用など
➡ 現職

《 好きな AWS のサービス 》AWS Fargate, AWS Lambda

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

本セッションは…

想定聴講者

- AWS Fargate の利用を検討している
- 細かい話は抜きにしてまずはコンテナを AWS 上で動かしたい
- デプロイや CI/CD パイプラインについて考えるのが好き

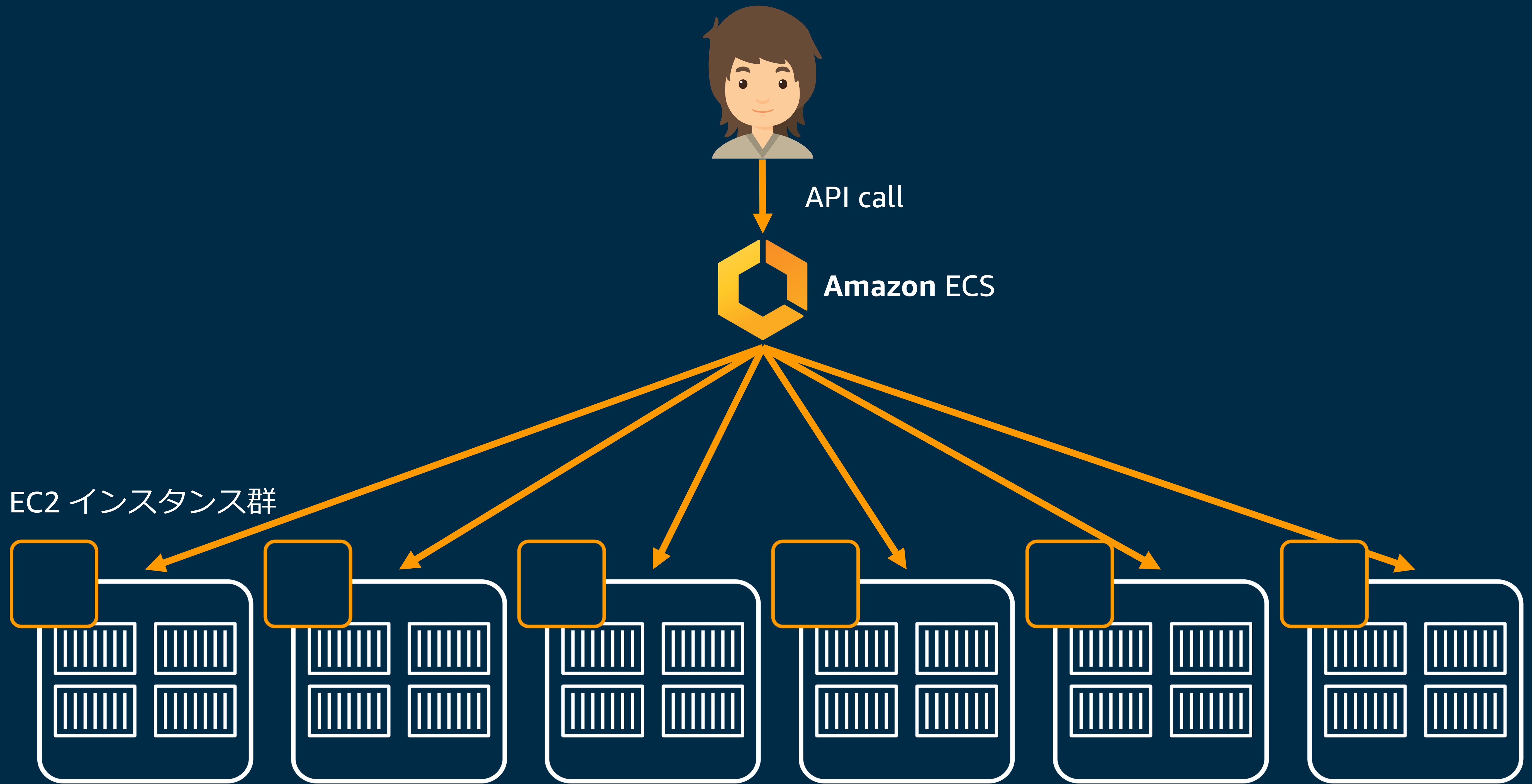
ゴール

- AWS Fargate の特徴と、AWS Fargate を利用したコンテナ実行方法を知る
- 様々なデプロイツールの特徴を知り、デプロイへの探究心を高める
- 自社・自チームのデプロイ方法についてあらためて考えていただくキッカケを作る

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS Fargate とは

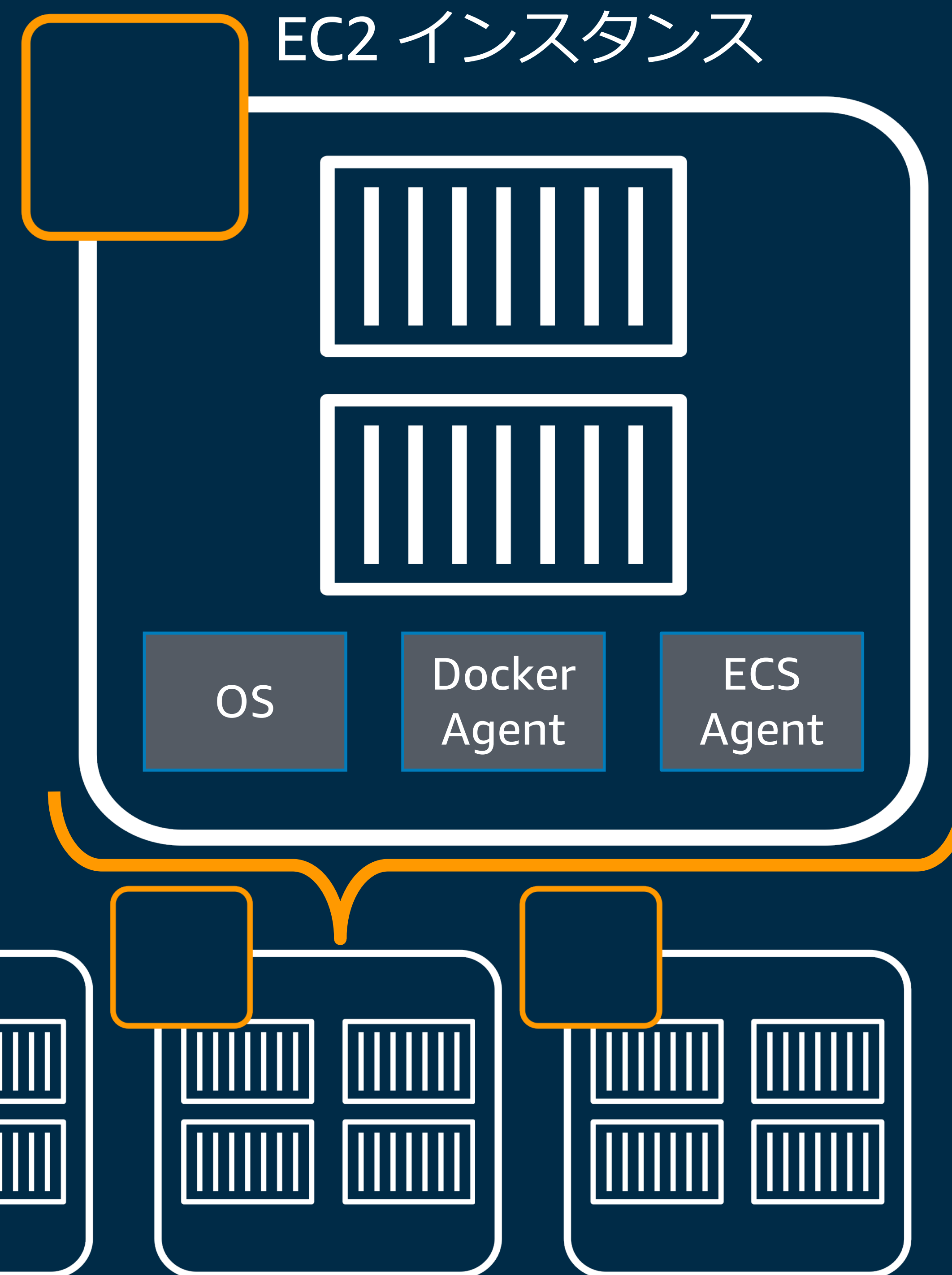
© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.



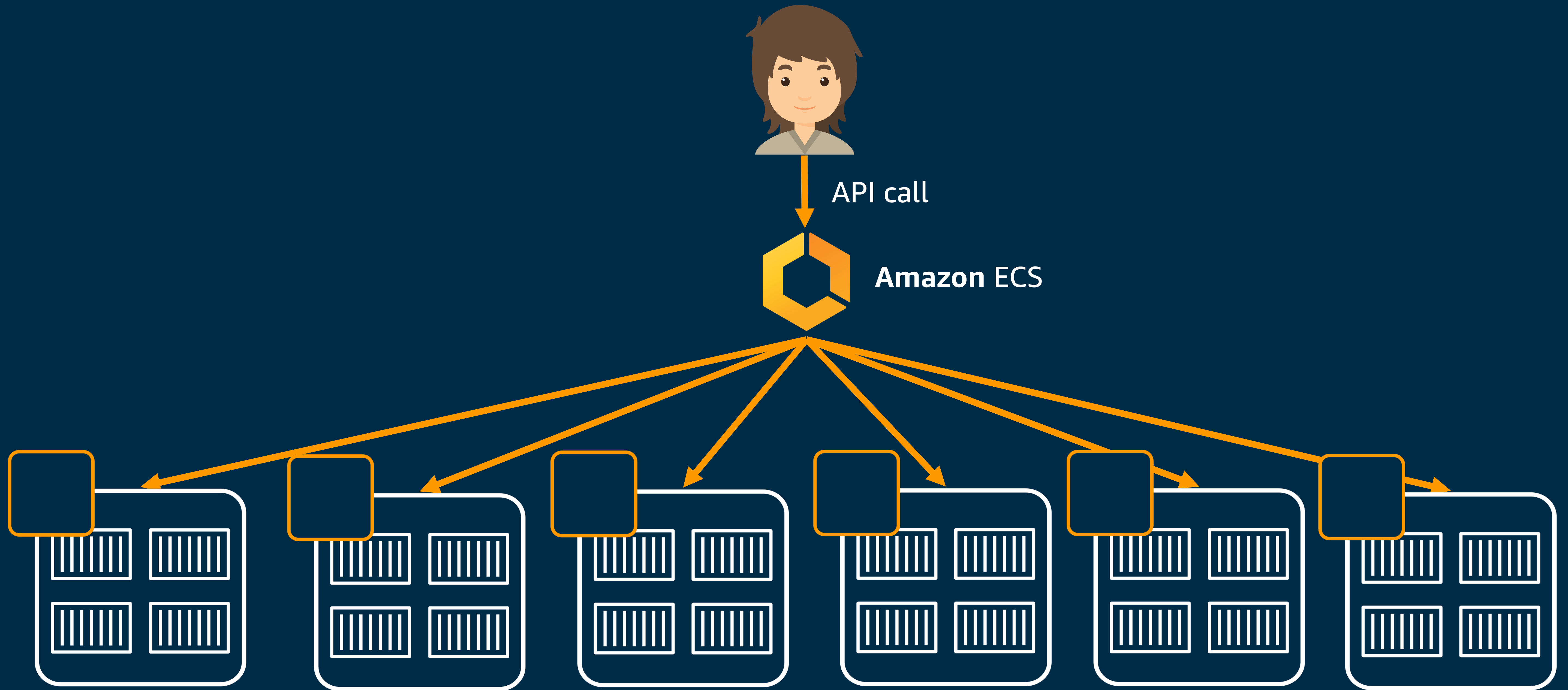
© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

実行環境 EC2 インスタンスの運用業務

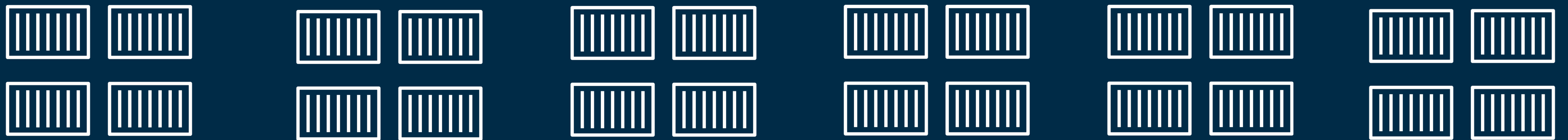
- OS やエージェント類へのパッチ当て・更新
- 実行中のコンテナ数に基づく、最適なリソース使用率を保つための EC2 インスタンス数のスケーリング



© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.
aws



© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.
aws



© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws

AWS Fargate

AWS マネージド

EC2 インスタンスのプロビジョン、スケール、管理不要

コンテナネイティブ

仮想マシンを意識しないシームレスなスケーリング
コンテナの起動時間・使用リソースに応じた料金設定

AWS サービスとの連携

VPC ネットワーキング、Elastic Load Balancing、IAM、
CloudWatch、etc.

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS Fargate を利用したコンテナのデプロイ

register
Task Definition

アプリケーションの実行に必要な
コンテナ群を定義する
e.g. コンテナイメージ URL、CPU、
メモリなど

run
Task

- タスク定義から実行される「タスク」
- 「FARGATE 起動タイプ」を利用

create
Service

- 複数タスクの実行状態を維持
- ELBとの連携
- Unhealthy なタスクは自動的に置き換えられる

create
Cluster

- アイソレーションの境界
- IAM パーミッションの境界

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws

AWS Fargate を利用したコンテナのデプロイ

```
# コンテナイメージのビルドとプッシュ
$ docker build –t xxx.dkr.ecr.us-west-2.amazonaws.com/myapp:latest .
$ docker push xxx.dkr.ecr.us-west-2.amazonaws.com/myapp:latest

# ECS タスク定義を作成し、登録
$ aws ecs register-task-definition --cli-input-json my-ecs-task-def.json
```

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS Fargate を利用したコンテナのデプロイ

```
# ECS タスクの実行

$ aws ecs run-task --task-definition ${YOUR_TASK_DEF_ARN}

# ECS サービス定義の登録、サービスの実行 (常駐型アプリケーションの実行)

$ aws ecs create-service --service-name my-service \
    --cli-input-json my-ecs-service.json
```

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

実際の ECS タスク定義のイメージ (抜粋)

```
{
  "family": ”myapp",
  "cpu": "1 vCpu",
  "memory": "2 gb",
  "containerDefinitions": [
    {
     "name":“frontend",
     "image":"xxx.dkr.ecr.us-west-2.amazonaws.com/frontend“,
     "cpu": 256,
     "memoryReservation": 1024
    },
    {
     "name":“api",
     "image":"xxx.dkr.ecr.us-west-2.amazonaws.com/api",
     "cpu": 768,
     "memoryReservation": 1024
    }
  ]
}
```

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

DEV DAY

Quick start with “Fargate CLI”

© 2019, Amazon Web Servic… rights reserved.

Quick start with “Fargate CLI”

やりたいこと

Docker Hub にある nginx:latest を Fargate で動かしてみたい

How To

1. デフォルト VPC にポート80番を開放したセキュリティグループを作る
2. タスクを実行する

```
$ fargate task run web \
      --image nginx:latest \
      --security-group-id ${SG_ID}
[i] Running task web
```

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws

Quick start with “Fargate CLI”

動作確認

```
$ fargate task ps

ID     IMAGE         ...    RUNNING   IP              CPU   MEMORY
0a...  nginx:latest  ...    34s       34.222.183.234  256   512

$ curl 34.222.183.234

<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
```

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Quick start with “Fargate CLI”

タスクの停止

```
$ fargate task stop web
[i] Stopped 1 task
```

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws

Quick start with “Fargate CLI”

やりたいこと

同じく nginx:latest を Fargate で常駐サービスとして動かしてみたい

How To

```
$ fargate service create ngx-svc \
      --image nginx:latest \
      --security-group-id ${SG_ID}
[i] Created service ngx-svc
```

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Quick start with “Fargate CLI”

動作確認

```
$ fargate service ps ngx-svc

ID     IMAGE          ...     RUNNING     IP               CPU   MEMORY
f1...  nginx:latest   ...     30s         34.216.225.232   256   512

$ curl 34.216.225.232

<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
```

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Quick start with “Fargate CLI”

awslabs/fargatecli

サービスの削除

```
$ fargate service scale ngx-svc 0
[i] Scaled service ngx-svc to 0

$ fargate service destroy ngx-svc
[i] Destroyed service ngx-svc
```

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws

Quick start with “Fargate CLI”

やりたいこと

コンテナイメージをビルドして Fargate で常駐サービスとして動かす

How To

```
$ ls
Dockerfile src
$ fargate service create myapp \
      --security-group-id ${SG_ID}
[>] docker login xxx.dkr.ecr.us-west-2.amazonaws.com/myapp
[>] docker build --tag xxx.dkr.ecr.us-west-2.amazonaws.com/myapp:20191002163132 .
Sending build context to Docker daemon   2.56kB
Step 1/2 : FROM alpine:3.8
 ~ snip ~
[i] Created service myapp
```

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Quick start with “Fargate CLI”

サービスの削除

```
$ fargate service scale myapp 0
[i] Scaled service myapp to 0

$ fargate service destroy myapp
[i] Destroyed service myapp
```

※ CNN などを運営する米ターナー社によるフォーク “turnerlabs/fargate” もあるが、
ここで紹介した “awslabs/fargatecli” とはインターフェースが異なるので注意

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

DEV DAY

Under the Hood: Fargate CLI

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Under the Hood: Fargate CLI

AWS アカウントにデフォルトで存在するリソースを利用

- VPC、サブネット、サブネットとルートテーブルの関連付け

ユーザー自身で作成

- セキュリティグループ

Fargate CLI による自動作成

- ECR リポジトリ
- ECS クラスタ
- ECS タスク実行 IAM ロール
- ECS タスク定義
- ECS サービス定義

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws

Under the Hood: Fargate CLI

Pros

- 各種必要リソースが自動生成されるため、それらを見て概念や各リソース定義の中身を勉強できる (see also `--verbose`)
- 細かい設定も可能だが、とにかく簡単

Cons

- 本番環境で使うにはちょっとアドホック?
- Infrastructure as code や CI/CD パイプラインについてもう少し考えて環境を作りたい

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

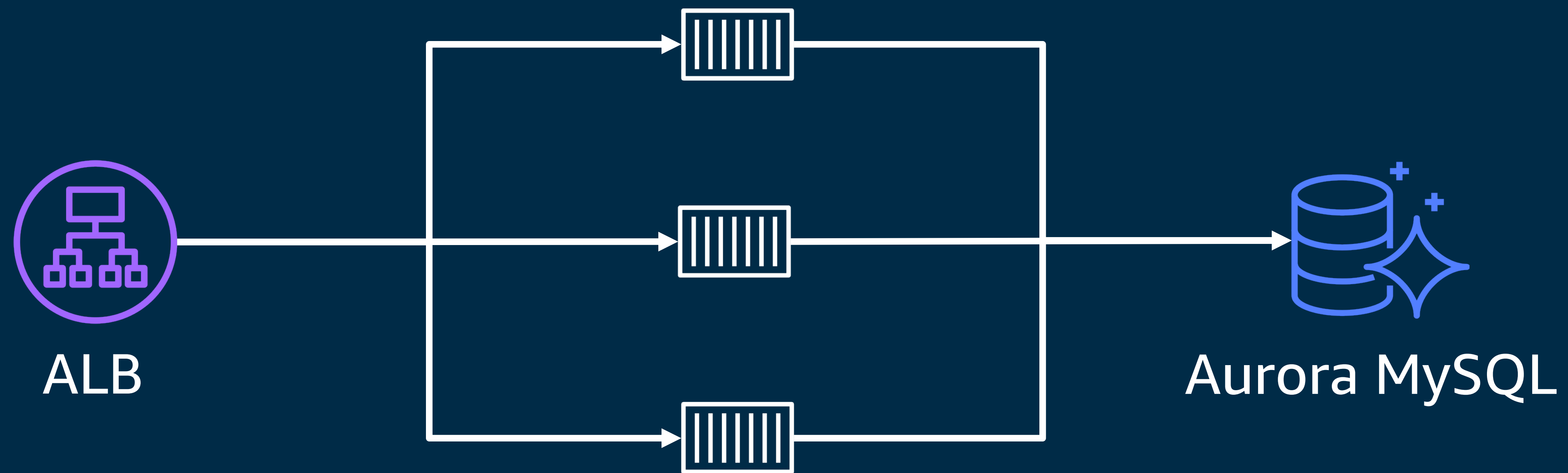
aws

DEV DAY

AWS Fargate に対応したデプロイツール

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

例えばこんな環境へのデプロイ



コンテナ on AWS Fargate

(ECS サービス)

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws

例えばこんな環境へのデプロイ

Fargate 以外に考えなければいけないことがある

- ECS 関連以外の AWS リソースはどのようにして作成・更新するか
- ECS サービス更新時にタスク(コンテナ)が正常に起動しない場合、ロードバランサのヘルスチェックが失敗してサービスインできない場合にどのようにしてロールバックするか

ALB

Aurora MySQL

コンテナ on AWS Fargate
(ECS サービス)

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws

Awesome-ECS からの抜粋

nathanpeck/awesome-ecs

- AWS CLI
- ECS CLI
- AWS CloudFormation
- Terraform
- AWS CDK
- ecs-deploy
- ecspresso

etc.

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

なぜこんなにもたくさんあるのか？

- AWS のサービスは MVP からスタート
- ローンチ時点で判明しているユースケースを必要十分にカバーする機能だけを実装
- ローンチ後、新たにユーザーから寄せられたフィードバックやユースケースを勘案し、後方互換性を保つ努力をしながら機能追加が行われる
- 既存ユーザー利用における動作保証を重視しながら、サービスのカバーするユースケースを広げていく

👉 全てのユースケースが 100% 同じものであることはほとんどない

本日 PM3 @fujiwara さんの”隙間家具”セッションもぜひご参加ください！

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws

DEV DAY

各デプロイツールの背景を読み解く

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS Management Console

起動タイプの互換性の選択

タスクを起動する場所に基づいて、タスク定義との互換性を持たせる起動タイプを選択します。

FARGATE

タスクサイズに基づく価格

ネットワークモード awsvpc が必要

AWS が管理するインフラストラクチャ、管理する Amazon EC2 インスタンスはありません

EC2

リソース使用量に基づく価格

複数のネットワークモードを使用可能

Amazon EC2 インスタンスを使用した自己管理インフラストラクチャ

***必須**

キャンセル　次のステップ

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws

AWS Management Console

Pros

GUI で Fargate タスクを実行できる

GUI であるため、実行中のタスクやサービスなどの情報を閲覧しやすい

Cons

デプロイが手作業になり再現性がない

ミスオペレーションのリスク

CI/CD パイプラインとの親和性なし

👉 閲覧目的には良いが、継続的なデプロイに使うのは難しそう

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws

AWS CLI

aws/aws-cli

Pros

- AWS の新サービスリリース時やサービスアップデートのタイミングで概ねそれらに対応している
- AWS API の呼び出しパラメーターをほぼ全て利用可能

Cons

- CI/CD パイプラインの実態がシェル芸になりがち
- リソースの依存関係をコマンドの実行順に反映する必要がある
- ロールバックを想定したスクリプトを書こうとすると複雑化しがち

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws

ecs-deploy

silinternational/ecs-deploy

```
Examples:
  Simple deployment of a service (Using env vars for AWS settings):

    ecs-deploy -c production1 -n doorman-service -i docker.repo.com/doorman:latest

  All options:

    ecs-deploy -k ABC123 -s SECRETKEY -r us-east-1 -c production1 -n doorman-service -i docker.repo.com

  Updating a task definition with a new image:

    ecs-deploy -d open-door-task -i docker.repo.com/doorman:17

  Using profiles (for STS delegated credentials, for instance):

    ecs-deploy -p PROFILE -c production1 -n doorman-service -i docker.repo.com/doorman -t 240 -e CI_TIM
```

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

ecs-deploy

silinternational/ecs-deploy

Pros

- 1つのシェルスクリプトの中で AWS の API を呼んでいるだけなので、中身を読んで理解しやすい
- コミュニティで揉まれているので、AWS CLI を使った自前シェルスクリプトを書くより安心・安全
- ロールバックなども考慮されている

Cons

- ネットワークやロードバランサー、Fargate サービスなどのリソースは他のツールで作成済みであることを前提にしている
- シェルスクリプトであるため、つい魔改造したくなるし、しやすい

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.
aws

ECS CLI

aws/amazon-ecs-cli

Available Commands

- ecs-cli
- ecs-cli configure
- ecs-cli up
- ecs-cli down
- ecs-cli scale
- ecs-cli ps
- ecs-cli push
- ecs-cli pull
- ecs-cli images
- ecs-cli license
- ecs-cli compose
- ecs-cli compose service
- ecs-cli logs
- ecs-cli check-attributes
- ecs-cli registry-creds
- ecs-cli local
- Using Docker Compose File Syntax
- Using Amazon ECS Parameters

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

ECS CLI

Pros

- Docker Compose 定義ファイルを利用した ECS サービスの作成・更新に対応しているため、ローカル開発で Docker Compose を利用している場合に導入が容易
- ECS 固有の機能(Parameter Store や FireLens など)もサポート

Cons

- これ一つでは全ての必要 AWS リソースを用意できないため、他ツール(e.g. CloudFormation など)の併用が前提
- ECS CLI 単体ではロールバックの機能を保有していないため、デプロイがうまくいかない場合のロールバックが複雑になる

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS CloudFormation / Terraform

Infrastructure as code

YAML ファイルなどに必要な AWS リソースについて宣言し、リソースの宣言的デプロイメントを可能にする

```
56
57    # The task definition. This is a simple metadata description of what
58    # container to run, and what resource requirements it has.
59    TaskDefinition:
60      Type: AWS::ECS::TaskDefinition
61      Properties:
62        Family: !Ref 'ServiceName'
63        Cpu: !Ref 'ContainerCpu'
64        Memory: !Ref 'ContainerMemory'
65        NetworkMode: awsvpc
66        RequiresCompatibilities:
67          - FARGATE
68        ExecutionRoleArn:
69          Fn::ImportValue:
70            !Join [':', [!Ref 'StackName', 'ECSTaskExecutionRole']]
71        TaskRoleArn:
72          Fn::If:
73            - 'HasCustomRole'
74            - !Ref 'Role'
75            - !Ref "AWS::NoValue"
76        ContainerDefinitions:
```

```
resource "aws_ecs_service" "mongo" {
  name            = "mongodb"
  cluster         = "${aws_ecs_cluster.foo.id}"
  task_definition = "${aws_ecs_task_definition.mongo.arn}"
  desired_count   = 3
  iam_role        = "${aws_iam_role.foo.arn}"
  depends_on      = ["aws_iam_role_policy.foo"]

  ordered_placement_strategy {
    type  = "binpack"
    field = "cpu"
  }

  load_balancer {
    target_group_arn = "${aws_lb_target_group.foo.arn}"
    container_name   = "mongo"
    container_port   = 8080
  }

  placement_constraints {
    type       = "memberOf"
```

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws

AWS CloudFormation / Terraform

Pros

- 単体で全ての AWS リソースを用意できるため、1つのツールで済む
- Fargate タスクだけでなく、関連する AWS リソースがどのような状態にあるのかを定義ファイルから確認しやすい
- 定義ファイルをバージョン管理システム管理下に置くことで変更履歴や変更理由を追いやすい
- 予定される変更差分を確認してから実行できる
- AWS リソース間の依存関係も定義できる
- 更新できないリソースの場合は新規作成してから古いリソースを消すような挙動を期待できる
- CI/CD パイプラインとの相性が良い

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS CloudFormation / Terraform

Cons

- 使いこなすためには定義する AWS リソースそれぞれについて十分に知っておく必要がある
- 最新の API パラメーターのサポートまでに時間がかかることも
- VPC やデータベースと AWS Fargate タスクのようなライフサイクルが異なるリソースの管理方法を考えられる一定のスキルが必要
- 記述量が多い
- 記述量が多い
- 記述量が多い

- 💡 Fargate タスクを含む必要なリソースを用意するための CloudFormation と Terraform のテンプレートファイルへのリンクを Appendix にて紹介しています

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS CDK

aws/aws-cdk

```
import ec2 = require('@aws-cdk/aws-ec2');
import ecs = require('@aws-cdk/aws-ecs');
import ecs_patterns = require('@aws-cdk/aws-ecs-patterns');
import cdk = require('@aws-cdk/core');

class BonjourFargate extends cdk.Stack {
  constructor(scope: cdk.App, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // Create VPC and Fargate Cluster
    // NOTE: Limit AZs to avoid reaching resource quotas
    const vpc = new ec2.Vpc(this, 'MyVpc', { maxAzs: 2 });
    const cluster = new ecs.Cluster(this, 'Cluster', { vpc });

    // Instantiate Fargate Service with just cluster and image
    const fargateService = new ecs_patterns.NetworkLoadBalancedFargateService(this, "FargateService", {
      cluster,
      image: ecs.ContainerImage.fromRegistry("amazon/amazon-ecs-sample"),
    });

    // Output the DNS where you can access your service
    new cdk.CfnOutput(this, 'LoadBalancerDNS', { value: fargateService.loadBalancer.loadBalancerDnsName });
  }
}

const app = new cdk.App();

new BonjourFargate(app, 'Bonjour');

app.synth();
```

https://github.com/aws-samples/aws-cdk-examples/tree/master/typescript/ecs/fargate-load-balanced-service

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS CDK

aws/aws-cdk

Pros

- コード(e.g. TypeScript, Python…)でインフラを記述できるためデベロッパーとの親和性が高く、ユニットテストも書きやすい
- 他のツールを併用しなくても Fargate を含めた AWS リソースを一通り作成可能
- 抽象度が高く多くのパラメーターを省略でき、記述量が少ない
- バックエンドが CloudFormation

Cons

- 抽象度が高いため記述量が少ないが、意識してないと Fargate CLI の自動作成と同じような状況になりえる

本日 PM2 AWSJ 大村の "AWS CDK" セッションもぜひご参加ください！

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws

ecspresso

kayac/ecspresso

```
usage: ecspresso [<flags>] <command> [<args> ...]

Flags:
  --help           Show context-sensitive help (also t
  --config=CONFIG  config file

Commands:
  help [<command>...]
    Show help.

  version
    show version

  deploy [<flags>]
    deploy service

  create [<flags>]
    create service
```

```
status [<flags>]
  show status of service

rollback [<flags>]
  rollback service

delete [<flags>]
  delete service

run [<flags>]
  run task

register [<flags>]
  register task definition

wait
  wait until service stable
```

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws

ecspresso

kayac/ecspresso

Pros

- 非常に薄いレイヤーで作られた AWS API ラッパー
- aws ecs describe-task-definition/describe-services の出力 JSON を転用できるため、画面から作ったタスクを後から CI/CD に載せられる
- テンプレーティング + 環境変数利用での定義ファイル出力が好き

  e.g. {{ env `FOO` `bar` }} や {{ must_env `FOO` }} など

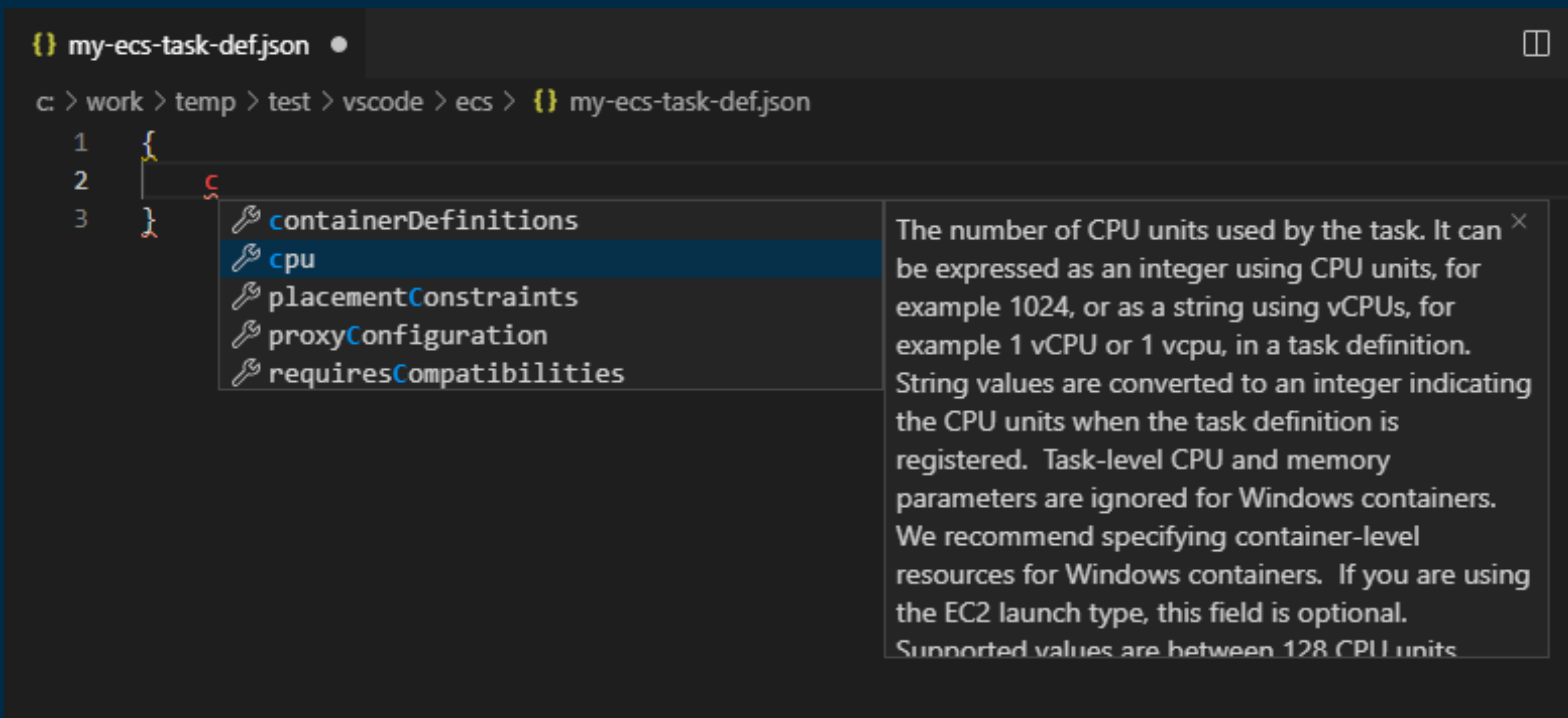
Cons

- VPC やロードバランサーなどの Fargate タスク・サービス以外のリソースは他のツールで作成済みであることを前提にしているため、別のツールの併用が必要

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Visual Studio Code の IntelliSense を利用して Amazon ECS タスク定義作成時の自動補完が可能に

New!



https://aws.amazon.com/about-aws/whats-new/2019/10/amazon-elastic-container-service-now-supports-intellisense-in-visual-studio-code/

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws

New!

Visual Studio Code の IntelliSense を利用して Amazon ECS のタスク定義を自動補完が可能に

利用手順

- Visual Studio Code に AWS Toolkit for VS Code をインストール
- ファイル名末尾に “ecs-task-def.json” を含める

https://aws.amazon.com/about-aws/whats-new/2019/10/amazon-elastic-container-service-now-supports-intellisense-in-visual-studio-code/

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws

DEV DAY

マッチしそうなツールはありましたか？

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

マッチしそうなツールはありましたか？

とりあえず動いているところを見たい

Fargate CLI が一番簡単そう (講演者調べ)

運用も考えつつ、CI/CD を意識してツールを選びたい

1. 単一のツールで全てのリソースを記述したい

AWS CLI, CloudFormation, Terraform, AWS CDK

2. Fargate へのデプロイはより簡素化・抽象化されたものでやりたい

ecs-deploy, ecspresso, ECS CLI, AWS CDK +上記のツール群のどれか

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Thank you!

Yasuhiro “Tori” Hara
yshr@amazon.co.jp

toricls

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

DEV DAY

Appendix

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Appendix

💡 TurnerLabs/Fargate

https://github.com/turnerlabs/fargate

💡 AWS CloudFormation sample templates

1. Networking resources

https://github.com/nathanpeck/aws-cloudformation-fargate/blob/master/fargate-networking-stacks/public-private-vpc.yml

2. AWS Fargate resources including ALB

https://github.com/nathanpeck/aws-cloudformation-fargate/blob/master/service-stacks/private-subnet-public-loadbalancer.yml

💡 Terraform sample templates - Airship Modules

https://airship.tf/

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws

Appendix

- 💡 登壇者個人の選手権優勝者は AWS CDK、次点で CloudFormation です

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.