

A - 6

第2回! /

AWS Fargate かんたんデプロイ選手権

Tori Hara /  toricls

Sr. Product Developer Advocate
Containers Product, AWS

In Partnership with 

2020.10.20-22



本セッションは…

想定聴講者

- AWS Fargate (for Amazon ECS) 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

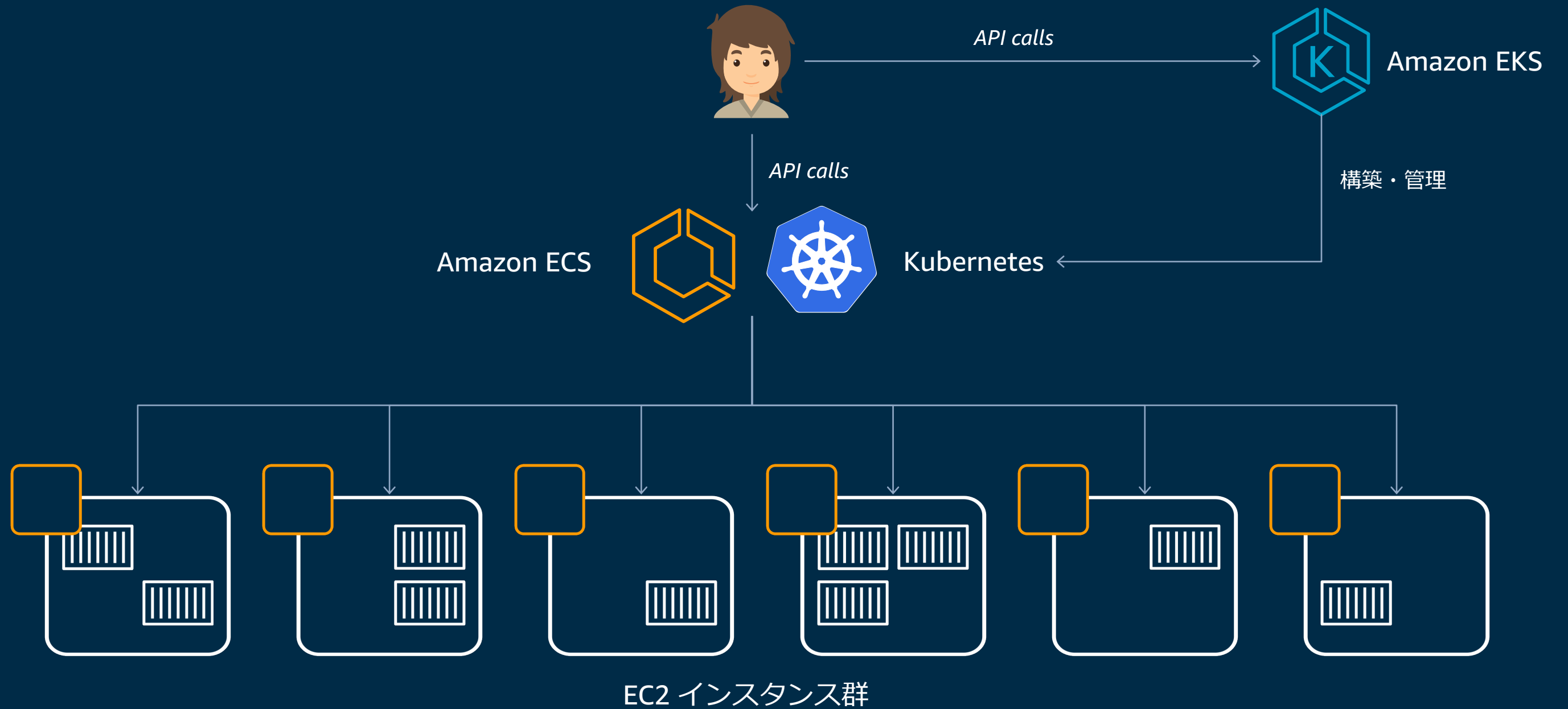
ゴール

- 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



AWS Fargate とは

コンテナオーケストレータの概要



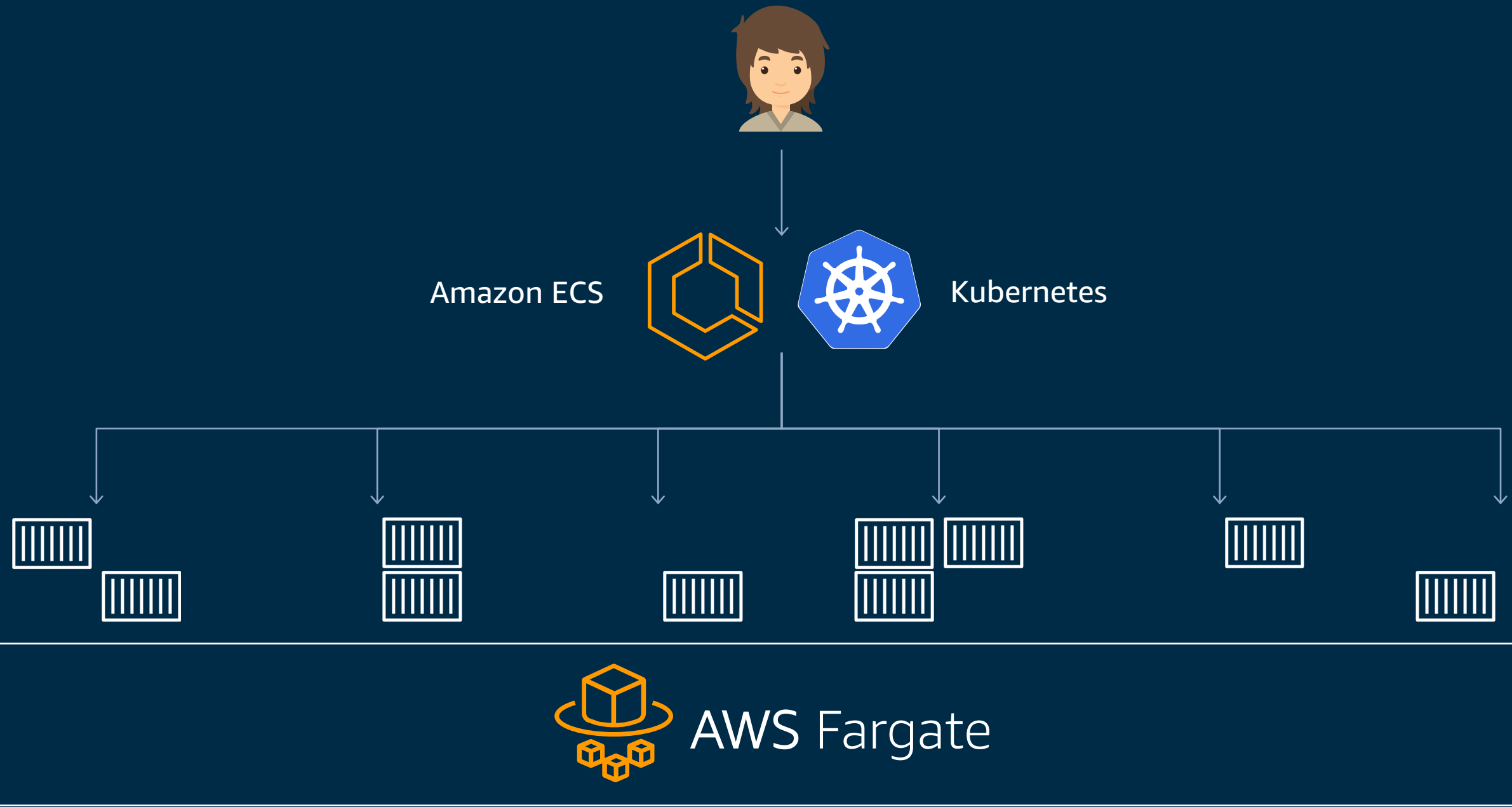
実行環境 EC2 インスタンスの運用業務

- OS やエージェント類へのパッチ当て・更新
- 実行中のコンテナ群に基づく最適なリソース使用率を保つための EC2 インスタンス数のプランニングやスケーリング



EC2 インスタンス群

Meet AWS Fargate



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ルリング
コンテナの起動時間・使用リソースに応じた料金設定

セキュリティ

タスク (Pod) はそれぞれが独立した分離単位を持ち、実行環境のカーネル、CPU、メモリ、ENI を他のタスク (Pod) と共有しない

AWS サービスとの連携

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

Quick start with “AWS Copilot CLI”



Quick start with "AWS Copilot CLI"



やりたいこと

アプリと Dockerfile は手元にある、あとはこれを Fargate で動かしたい

How To

```
$ copilot init
```



Quick start with "AWS Copilot CLI"



```

AWS Copilot
% ls
Dockerfile README.md index.html
% cat Dockerfile
FROM nginx
EXPOSE 80
COPY index.html /usr/share/nginx/html
% █
```

Quick start with "AWS Copilot CLI"



```

AWS Copilot
% copilot init
Welcome to the Copilot CLI! We're going to walk you through some questions
to help you get set up with an application on ECS. An application is a collection of
containerized services that operate together.

What would you like to name your application? [? for help] copi-demo
```

Quick start with "AWS Copilot CLI"



```
⌘ 1 AWS Copilot
% copilot init
Welcome to the Copilot CLI! We're going to walk you through some questions
to help you get set up with an application on ECS. An application is a collection of
containerized services that operate together.

Application name: copi-demo

Which service type best represents your service's architecture? [Use arrows to move, type to filter, ?
for more help]
> Load Balanced Web Service
  Backend Service
```

Quick start with "AWS Copilot CLI"



```
⌘ #1 AWS Copilot
% copilot init
Welcome to the Copilot CLI! We're going to walk you through some questions
to help you get set up with an application on ECS. An application is a collection of
containerized services that operate together.

Application name: copi-demo
Service type: Load Balanced Web Service

What do you want to name this Load Balanced Web Service? [? for help] front-end
```



Quick start with "AWS Copilot CLI"



```

AWS Copilot

% copilot init
Welcome to the Copilot CLI! We're going to walk you through some questions
to help you get set up with an application on ECS. An application is a collection of
containerized services that operate together.

Application name: copi-demo
Service type: Load Balanced Web Service
Service name: front-end

Which Dockerfile would you like to use for front-end? [Use arrows to move, type to filter, ? for more
help]
> ./Dockerfile

```

Quick start with "AWS Copilot CLI"



```

AWS Copilot

% copilot init
Welcome to the Copilot CLI! We're going to walk you through some questions
to help you get set up with an application on ECS. An application is a collection of
containerized services that operate together.

Application name: copi-demo
Service type: Load Balanced Web Service
Service name: front-end
Dockerfile: ./Dockerfile

Ok great, we'll set up a Load Balanced Web Service named front-end in application copi-demo listening on
port 80.

✓ Created the infrastructure to manage services under application copi-demo.

✓ Wrote the manifest for service front-end at copilot/front-end/manifest.yml
Your manifest contains configurations like your container size and port (:80).
```



Quick start with "AWS Copilot CLI"



```
Application name: copi-demo
Service type: Load Balanced Web Service
Service name: front-end
Dockerfile: ./Dockerfile
Ok great, we'll set up a Load Balanced Web Service named front-end in application copi-demo listening on port 80.

✓ Created the infrastructure to manage services under application copi-demo.

✓ Wrote the manifest for service front-end at copilot/front-end/manifest.yml
Your manifest contains configurations like your container size and port (:80).

✓ Created ECR repositories for service front-end.

All right, you're all set for local development.

Would you like to deploy a test environment? [? for help] (y/N) y
```


Quick start with "AWS Copilot CLI"



- ✓ Created the infrastructure to manage services under application `copi-demo`.
- ✓ Wrote the manifest for service `front-end` at `copilot/front-end/manifest.yml`. Your manifest contains configurations like your container size and port (`:80`).
- ✓ Created ECR repositories for service `front-end`.

All right, you're all set for local development.

Deploy: Yes

- ```
.: Creating the infrastructure for the test environment.
- Virtual private cloud on 2 availability zones to hold your services [In Progress]
- Internet gateway to connect the network to the internet [In Progress]
- Public subnets for internet facing services [In Progress]
- Private subnets for services that can't be reached from the internet [In Progress]
- Routing tables for services to talk with each other [In Progress]
- ECS Cluster to hold your services [In Progress]
- Application load balancer to distribute traffic [In Progress]
```

# Quick start with "AWS Copilot CLI"

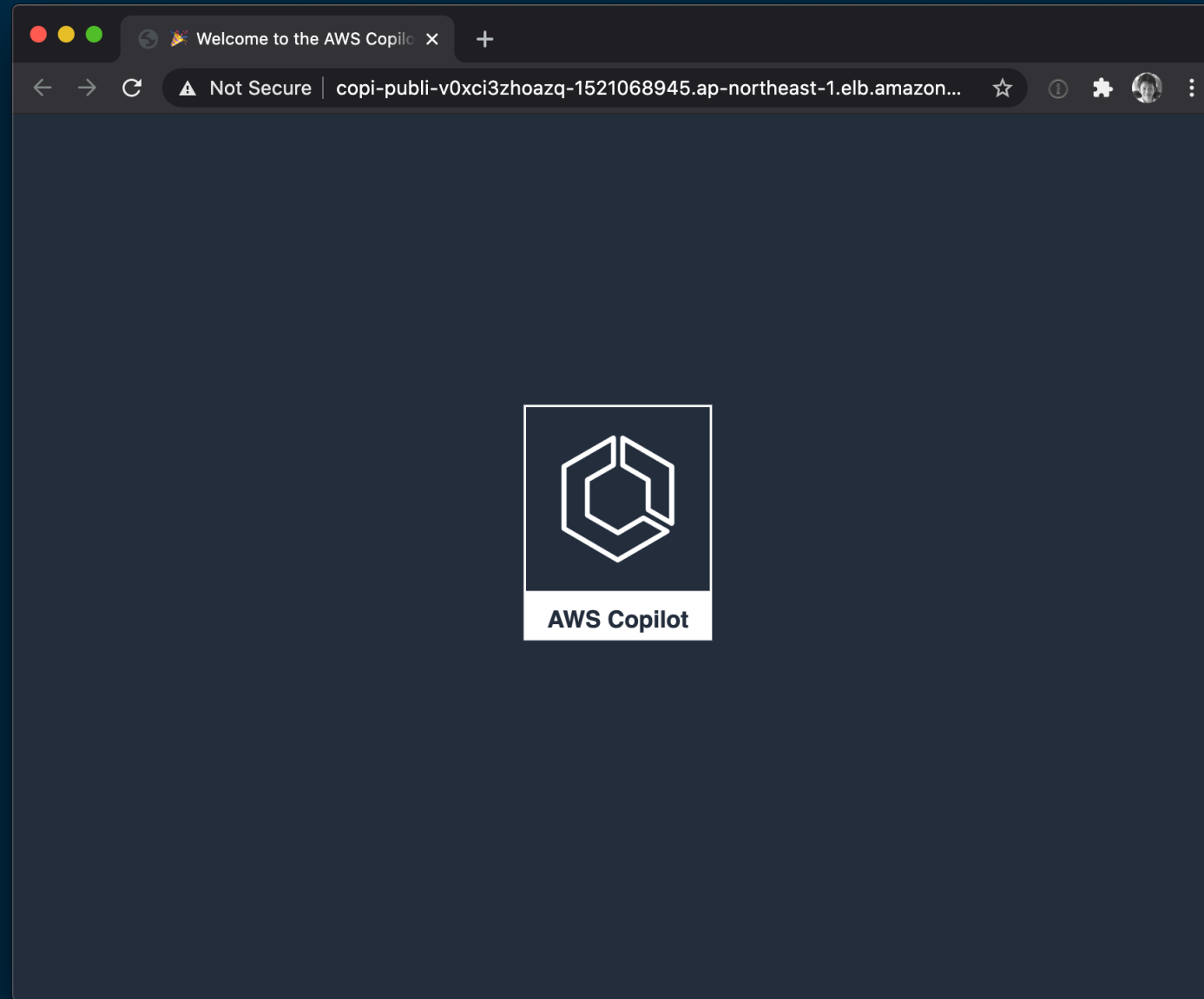


```
---> Using cache
---> 2a2bccb8a34f
Successfully built 2a2bccb8a34f
Successfully tagged 184834139833.dkr.ecr.ap-northeast-1.amazonaws.com/copilot-demo/front-end:975886a
Login Succeeded
The push refers to repository [184834139833.dkr.ecr.ap-northeast-1.amazonaws.com/copilot-demo/front-end]
14933325658e: Pushed
550333325e31: Pushed
22ea89b1a816: Pushed
a4d893caa5c9: Pushed
0338db614b95: Pushed
d0f104dc0a1f: Pushed
975886a: digest: sha256:f81af7aee1662c4d4f90347fed81d43282970b193544cbc7bc20952e682d8e88 size: 1570
```

✓ Deployed front-end, you can access it at <http://copi-Publi-V0XCI3ZH0AZQ-1521068945.ap-northeast-1.elb.amazonaws.com>.



# Quick start with "AWS Copilot CLI"



# AWS Fargate に対応したデプロイツール

(for Amazon ECS)

# Awesome-ecs からの抜粋



nathanpeck/awesome-ecs

- AWS CLI
- AWS CloudFormation
- Terraform
- AWS Copilot
- AWS CDK
- ecs-deploy
- ecspresso
- Docker Compose ECS integration

etc.



# 各デプロイツールの守備範囲や背景を 読み解く

# AWS Management Console

ECS オブジェクトとそれらの関連を示す図

The diagram illustrates the relationship between ECS objects. It features a central yellow square representing a container definition. This square is enclosed within a dashed-line square representing a task definition. The task definition is further enclosed within a larger dashed-line square representing a service. Finally, the service is enclosed within the largest dashed-line square representing a cluster. Lines connect the labels 'Container definition', 'Task definition', 'Service', and 'Cluster' to their respective levels in the diagram.

Container definition

Task definition

Service

Cluster

## コンテナの定義

[編集](#)

以下のコンテナのイメージを選択し、すばやく使用を開始するか、使用するコンテナイメージを定義します。

|                                                                                     |                                                                                   |
|-------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|
| <b>sample-app</b><br>イメージ : httpd:2.4<br>メモリ : 0.5GB (512)<br>cpu : 0.25 vCPU (256) | <b>nginx</b><br>イメージ : nginx:latest<br>メモリ : 0.5GB (512)<br>cpu : 0.25 vCPU (256) |
|-------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|

# AWS Management Console

## Pros

GUI で Fargate タスクを作成できる

ECS 以外の AWS サービスもまとめて設定可 e.g. Application Auto Scaling, CodeDeploy

実行中のタスクやサービスなどの情報閲覧が容易

## 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はあまりない

👉 継続的なデプロイに使うのは難しそう





## 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 パイプラインがシェル芸になりがち
- リソース依存関係まで考えたデプロイには高度なスクリプト技術が必要
- ロールバックを想定したスクリプトは難しい



## 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
```

# 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堅牢
- 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



# 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

```
55 Resources:
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"
```

# 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

# 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
- 記述量が多くなりがち
- 記述量が多くなりがち
- 記述量が多くなりがち
  - Terraform Modules の利用で書く量は減らせる
  - が、ECS 向けのモジュールが(デプロイツール同様)たくさんある



```
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
 const vpc = new ec2.Vpc(this, 'MyVpc', { maxAzs: 2 });
 const cluster = new ecs.Cluster(this, 'Cluster', { vpc });

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

 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-application-load-balanced-service>

## 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との親和性が高く、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・管理可能
- 抽象度が高く多くのパラメーターを省略でき、記述量を減らせる
- 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、意識していないとなぜそういうリソースが作られるのか説明できなくなる可能性も



New!

# ECS extensions for CDK (Developer Preview)



```
const svcDesc = new ServiceDescription();
svcDesc.add(new Container({
 cpu: 1024,
 memoryMiB: 2048,
 trafficPort: 80,
 image: ContainerImage.fromRegistry('ecs-sample'),
}));
```

```
svcDesc.add(new HttpLoadBalancerExtension());
svcDesc.add(new AppMeshExtension({ mesh }));
svcDesc.add(new FireLensExtension());
svcDesc.add(new XRayExtension());
svcDesc.add(new CloudwatchAgentExtension());
svcDesc.add(new ScaleOnCpuUtilization({
 initialTaskCount: 2,
 minTaskCount: 2,
}));
```

```
const mySvc = new Service(stack, 'my-service', {
 environment: environment,
 serviceDescription: svcDesc,
});
```

## “Extensions” による拡張

- Elastic Load Balancing
- AWS App Mesh
- AWS FireLens
- AWS X-Ray
- AWS CloudWatch Agent
- CPU ベースのアプリケーション  
自動スケーリング

etc.

<https://github.com/aws/aws-cdk/tree/master/packages/%40aws-cdk-containers/ecs-service-extensions>



# AWS Copilot (再掲, Developer Preview)



☰ AWS Copilot CLI  aws/copilot-cli 1k Stars · 100 Forks

## Your toolkit for containerized applications on AWS

AWS Copilot is an open source command line interface that makes it easy for developers to **build, release, and operate** production ready containerized applications on Amazon ECS and AWS Fargate.

[Get started →](#)

```
File: copilot/frontend/manifest.yml
1 # The manifest for the "frontend" service.
2
3 # Your service name will be used in naming your resources like log
 groups, ECS services, etc.
 name: frontend

~/D/p/my-app >>> copilot pipeline status
Pipeline Status

Stage Transition Status

Source ENABLED Succeeded
└─ SourceCodeFor-emoji-race
Build ENABLED Succeeded
```

<https://aws.github.io/copilot-cli/>



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel.

# AWS Copilot (再掲, Developer Preview)



```
AWS Copilot
% copilot --help
👩🏻🚀 Launch and manage applications on Amazon ECS and AWS Fargate.

Commands
Getting Started 🌱
 init Create a new ECS application.
 docs Open the copilot docs.

Develop ✨
 app Commands for applications.
 Applications are a collection of services and environments.

 env Commands for environments.
 Environments are deployment stages shared between services.

 svc Commands for services.
 Services are long-running Amazon ECS services.

 task Commands for tasks.
 One-off Amazon ECS tasks that terminate once their work is done.

Release 🚀
 pipeline Commands for pipelines.
 Continuous delivery pipelines to release services.
```

<https://aws.github.io/copilot-cli/>



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel.

# AWS Copilot (再掲, Developer Preview)



New!

```
Release 🚀
 pipeline Commands for pipelines.
 Continuous delivery pipelines to release services.

 deploy Deploy your service.

Addons 🐱
 storage Commands for working with storage and databases.

Settings ⚙️
 version Print the version number.
 completion Output shell completion code.

Flags
-h, --help help for copilot
-v, --version version for copilot

Examples
Displays the help menu for the "init" command.
`$ copilot init --help`
%`
```

<https://aws.github.io/copilot-cli/>



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel.

# AWS Copilot (再掲, Developer Preview)



New!

## Pros

- 最低限必要なのはアプリケーションと Dockerfile のみ
- アーキテクチャが “*Load Balanced Web Service*”, “*Backend Service*” のようなユースケースの選択に抽象化されており、それらに適した AWS リソースをまとめて構築可能
- 抽象度が高く、AWS CDK よりも更に少ない記述量
- 複数 AWS アカウントや複数環境へのデプロイをサポートし、それらへの継続的デリバリを実施する CI/CD パイプラインを作成可能
- CloudFormation による任意のリソース持ち込みを可能にする “Add-on”
  - この CloudFormation テンプレートを AWS CDK で書くことも当然可能
- バックエンドが CloudFormation
- プロダクションレディな環境を CloudFormation で定義する勉強材料に



# AWS Copilot (再掲, Developer Preview)



## Cons

- 現時点では Developer Preview のステージであり、細かい設定変更は未サポート e.g. 『セキュリティグループで特定 IP アドレスからのみアクセスを許可したい』

💡 AWS Copilot については Appendix も合わせてご覧ください



# ecspresso



祝 🎉 v1.0.0! /  
kayac/ecspresso

```
usage: ecspresso --config=CONFIG [<flags>] <command> [<args>
```

## Flags:

```
--help Show context-sensitive help (also try --help)
--config=CONFIG config file
--debug enable debug log
```

## 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
```

```
init --region=REGION --service=SERVICE [<flags>]
 create service/task definition files by existing ECS service
```

```
diff [<flags>]
 display diff for task definition compared with latest definition
```





## Pros

- 非常に薄いレイヤーで作られた AWS API ラッパー
- 画面から作ったタスクや、既存の ECS サービスを後から CI/CD に載せることが想定されていてとても好き

```
$ ecspresso init --region ap-northeast-1 --cluster default --service myservice --config config.yaml
2019/10/12 01:31:48 myservice/default save service definition to ecs-service-def.json
2019/10/12 01:31:48 myservice/default save task definition to ecs-task-def.json
2019/10/12 01:31:48 myservice/default save config to config.yaml
```

```
$ ecspresso deploy --config config.yaml
```

- テンプレートティング + 環境変数利用での定義ファイル出力が大好き  
e.g. `{{ env `FOO` `bar` }}` や `{{ must_env `FOO` }}` など





## Pros

- *tfstate* ファイル内の値を参照できて好き

config.yaml

```
region: ap-northeast-1
cluster: default
service: test
service_definition: ecs-service-def.json
task_definition: ecs-task-def.json
plugins:
- name: tfstate
 config:
 path: terraform.tfstate # path to
```

ecs-service-def.json

```
{
 "networkConfiguration": {
 "awsVpcConfiguration": {
 "subnets": [
 "{{ tfstate `aws_subnet.private-a.id` }}"
],
 "securityGroups": [
 "{{ tfstate `data.aws_security_group.default.id` }}"
]
 }
 }
}
```

## Cons

- VPC やロードバランサーなど、Fargate タスク・サービス以外のリソースは事前に作成済みであることを前提としており、AWS についてある程度の事前知識が必要
- また、それらリソースの作成・管理は別ツールの併用が前提

💡 これらは ecspresso において意図的にそのようにデザイン・実装されていることに由来すると思われるため、実は Cons らしい Cons でもない

# Docker Compose ECS integration (Beta)



docker/compose-cli

New!

```
Docker ECS integration
% ls
docker-compose.yml
% cat docker-compose.yml
version: '3.7'
services:
 nginx:
 image: nginx:1.19.3-alpine
 ports:
 - 80:80
%
% docker context create ecs my-ecs-env
? Select AWS Profile sandbox
? Region ap-northeast-1
? Enter credentials No
Successfully created ecs context "my-ecs-env"
%
% docker compose up --context my-ecs-env
[+] Running 5/9
.: dockerecstest CREATE_IN_PROGRESS 14.4s
:: Cluster CREATE_COMPLETE 6.0s
:: LogGroup CREATE_COMPLETE 3.0s
:: DockerecstestDefaultNetwork CREATE_COMPLETE 7.0s
:: NginxTCP80TargetGroup CREATE_COMPLETE 1.0s
.: CloudMap CREATE_IN_PROGRESS 9.4s
```



# Docker Compose ECS integration (Beta)



docker/compose-cli

New!

## Pros

- Docker Compose を利用したローカル開発と非常に親和性が高い
- ロードバランサや CloudWatch ロググループなど Fargate タスクの実行に必要なものが一通り作成される
- バックエンドが CloudFormation

## Cons

- 現時点では Beta であり、例えば ECS サービスの更新は未サポート
- VPC やサブネットなどのリソースは事前に作成済みであることを前提としており、未指定の場合は default リソースが利用される
- Docker Compose が管理しない AWS リソースの作成・管理には別ツールの併用が必要



マッチしそうなツールはありましたか？

# 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 / これから AWS をはじめる

AWS Copilot 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抽象化されたものでやりたい

- ecs-deploy / ecspresso / Docker Compose ECS integration のいずれかと上記ツールの組み合わせ
- AWS CDK + ECS extensions for CDK
- AWS Copilot



# Thank you!

Tori Hara /  toricls  
AWS

# Appendix



# Appendix

💡 AWS Copilot - <https://aws.github.io/copilot-cli/>

- AWS Copilot のご紹介 <https://aws.amazon.com/jp/blogs/news/introducing-aws-copilot/>
- AWS Copilot によるコンテナアプリケーションの自動デプロイ  
<https://aws.amazon.com/jp/blogs/news/automatically-deploying-your-container-application-with-aws-copilot/>

💡 Docker Compose ECS integration - <https://github.com/docker/compose-cli>

- Deploying Docker containers on ECS <https://docs.docker.com/engine/context/ecs-integration/>

💡 第1回 AWS Fargate かんたんデプロイ選手権

<https://speakerdeck.com/torics/the-very-first-aws-fargate-easy-deployment-tooling-championship>

