

セクション1・2

2023年8月2日 16:38

<https://github.com/uchidayuma/laravel-docker-compose-environment>

できるようになること

Dockerコンポーズyml

.ymlファイルなどでインフラの設計図を作成できるようになる

目標

DockerComposeで複数コンテナ管理と連携させつつこれまでの開発環境をDocker化できる

↑

複数コンテナを管理

Dockerが必要な理由

2023年8月2日 16:45

環境構築には時間がかかる

Docker・・・プログラムをひとまとめにしたコンテナを使用する仕組み

Dockerを使うと？

設計図を作っておいて、設計図通りにDocker Compose Upをすれば構築できる

開発環境構築時間の削減

OSやパソコンのスペックに関係なく構築できる

ハードウェアやOSの差異を吸収するのでアプリ環境の実行環境の品質アップ



コンテナ

2023年8月2日 16:52

コンテナ：アプリを隔離されたパッケージ化できる

パッケージ化されたソフトが並ぶような構造を構築できる

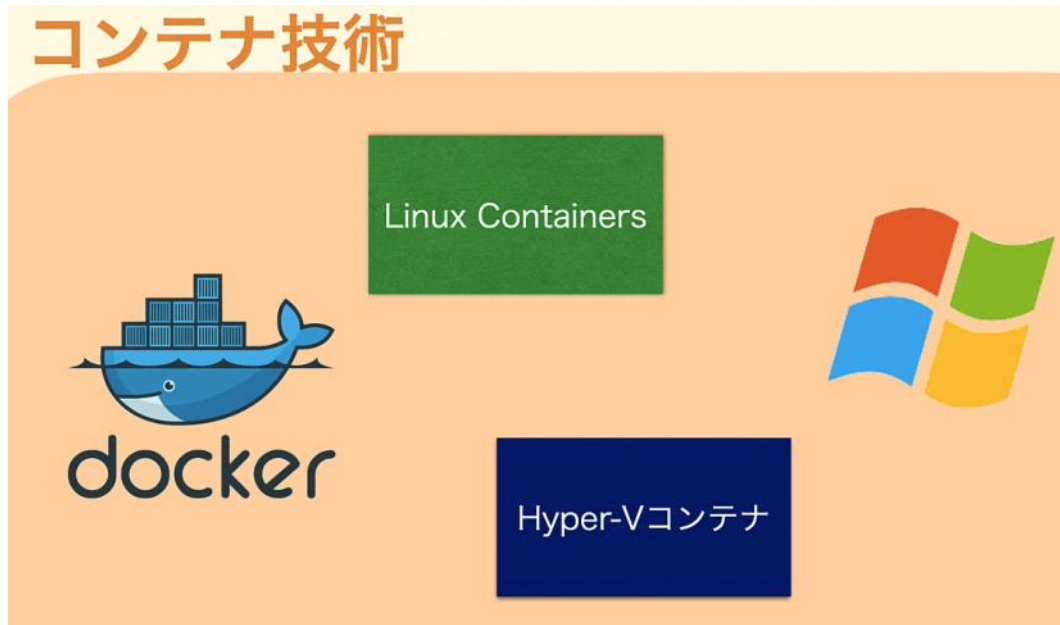
Dockerによってコンテナを管理できる



コンテナとdockerの関係

2023年8月2日 16:56

Docker：コンテナ技術の一つ



dockerが一番使われている（ユーザーに優しい）

仮想環境とDockerの違い

2023年8月2日 16:59

仮想環境：パソコンの中にパソコンを作る技術

例) WindowsPCの中にLinuxサーバーを立てる

- 2つのOSを立ち上げるため重い
- ゲストOSはHDDに保存
- VirtualBoxが有名

ディスク上に存在

コンテナ

Linuxサーバー上に直接ソフトウェアが立ち上がるイメージ

ホストOS・ゲストOSの区切りがない

メモリ上にアプリが存在する



Docker

- メモリ上
- ゲストOSは存在しない
- データは一時的
- Dockerfileで可能

存在場所
ゲストOS
データ管理
コード化



仮想環境

- ディスク上
- ホストOSとゲストOSに分かれる
- 常に保存される
- 不可

インフラのコード化

2023年8月2日 17:05

コード化

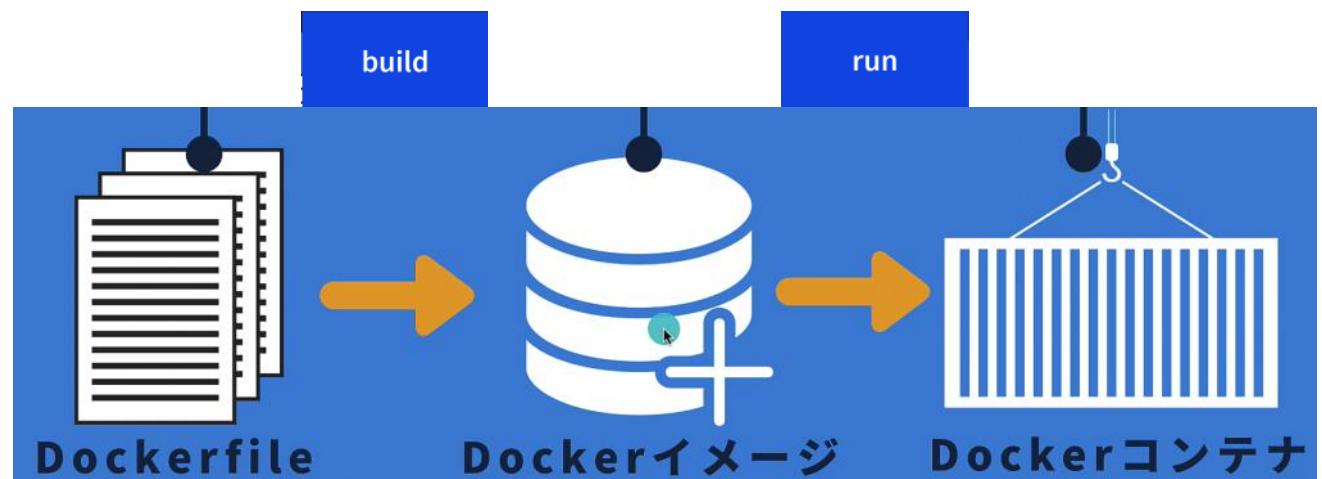
- ファイルを読むと構成がわかる
- 誰でも同じコンテナになる
- 受け渡しが簡単

サーバーで実行するコードを定義(Infrastructure As a Code)

- 設定などを自動で行う

DockerfileとDocker

2023年8月2日 17:14



Dockerイメージ

2023年8月2日 17:19



Dockerタグ

2023年8月2日 17:38



タグを指定しないと最新バージョンが設定される

https://hub.docker.com/_/ubuntu

https://hub.docker.com/_/ubuntu

複数コンテナ管理と連携

2023年8月2日 17:50



コマンドインストール

2023年8月2日 17:59

```
sudo su - ec2-user
```

```
sudo yum install docker
```

```
sudo ss -antp
```

```
sudo ss -antpu
```

```
sudo systemctl restart docker.service
```

```
sudo systemctl status docker
```

```
sudo systemctl start docker
```

```
sudo systemctl enable docker
```

```
docker ps
```

```
sudo usermod ec2-user -G systemd-journal,docker
```

```
docker ps
```

セクション4

2023年8月2日 18:07

run ・ ps

2023年8月3日 9:05

run : dockerコンテナを 1 つ起動するコマンド

→docker imageがない場合は、docker hubから自動で取得

ps : 起動中のコンテナを表示するコマンド

停止中のコンテナは表示されない

→停止中のコンテナも見たい場合は-aオプションをつける

\$ docker run <起動したいイメージ名>

\$ dpcker ps → hello-worldでない

\$ docker ps -a →表示される

フォアグラウンドで動いているプロセスがない場合、コンテナが停止してしまう

helloworldコンテナの中身：ターミナルに"Hello from Docker!"を表示する

※表示後は停止する

\$ docker run --name oyaizu-udemy hello-world

\$ docker ps -a

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
4e944b2672a5	hello-world	"/hello"	11 seconds ago	Exited (0) 10 seconds ago		oyaizu-udemy
07f01023ea2c	hello-world	"/hello"	8 minutes ago	Exited (0) 8 minutes ago		pedantic_shtern

\$docker run --name rmttest --rm hello-world

--rm : 実行後に停止したコンテナを自動で削除してくれる

bashを使って-itオプションでcentosを操作する

\$ docker run -it --name mycentos centos:8 /bin/bash

-it : shellを使用する

centos:8 : centosのバージョン8

→ [root@95500d782260 /]# exit

\$docker ps -a

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
95500d782260	centos:8	"/bin/bash"	30 seconds ago	Exited (0) 6 seconds ago		mycentos

start ・ stop

2023年8月2日 18:07

コンテナ 1 つを操作

いずれも削除はしないので、残る

\$ docker start mycentos

→docker ps -a で確認するとSTATUSの部分がUPになっている

\$ docker stop <CONTAINER ID>

→docker ps -a で確認するとSTATUSの部分がExitedになっている

\$ docker restart <CONTAINER ID or NAMES>

→起動される

exec

2023年8月3日 9:06

コンテナに入るとは？

コンテナ内のコマンドライン(bashやzdh)にアクセスすること
→コマンドライン経由でコンテナを操作できる

起動中のコンテナ内でコマンドを実行する

- ・ 起動中のコンテナに入らずコマンドを実行
- ・ -itオプションを使うとコンテナに入ることができる

```
$ docker exec -it mycentos /bin/bash
```

```
# cat /etc/redhat-release
```

バージョンが返ってくる

```
$ docker exec mycentos cat /etc/redhat-release
```

同じ結果

rm

2023年8月3日 9:16

停止中のコンテナを削除する

-fオプションで起動中のコンテナも強制削除
→コンテナのデータも消えるので注意

```
$ docker rm -f <ID or NAMES>
```

起動中のコンテナも削除

images ・ rmi

2023年8月3日 9:19

image : ローカルにあるimageを全て表示

意外とDockerイメージは容量が大きい！

→定期的に確認がおすすめ！

\$ docker images

rmi : Dockerイメージを削除

起動中のコンテナのイメージは削除できない

→Dockerイメージは依存関係があるので、ベースイメージは削除できない

\$ docker stop <ID>

\$ docker rmi <ID> → イメージ削除できる

build

2023年8月3日 9:27

Dockerfileからイメージを生成

例) sudo vi test/Dockerfile

FROM centos:7

RUN yum update -y

\$ docker build test/

→実行される

docker cp

2023年8月3日 9:34

コンテナのとホストマシンでファイルのやり取りを行うコピーコマンド
ログファイルや設定ファイルの取り出し or 入力を使う

例) sudo vi command/sample.txt

ホストからコンテナにコピー

```
$ docker run -it --name mycentos centos:8 /bin/bash
```

```
$ docker start mycentos
```

```
$ docker cp command/sample.txt mycentos:/opt
```

```
$ docker exec -it mycentos /bin/bash
```

```
# cat /opt/sample.txt → 同じ内容
```

コンテナからホストにコピー ※ホストからコマンドを入力する

```
$ docker cp mycentos:/opt/container.txt /home/ec2-user/test/cp/
```

logs

2023年8月3日 9:47

Dockerコンテナのログ出力

→-fオプションでリアルタイムログ

- ・原因不明のコンテナ停止
- ・アクセスログなどなど

リアルタイムログ

```
$ docker logs -f mycentos
```

必須ではない

2023年8月3日 10:01

docker inspect <NAMES>

Dockerの詳細情報出力

普段は見ない詳細情報をみれる

→トラブル時などに使うことがある

docker pull

Dockerイメージをダウンロード

pullの後ろに*プライベートイメージ

レジストリ付けると、DockerHub以外からダウンロードできる

docker commit

コンテナをイメージ化

attachなどで追加操作

→追加操作を新しいレイヤーにする

→オリジナルイメージ作成

*注意点：必ずアカウントIDを入れる

例) docker commit mycentos <docker hub ID>/oyaizu-centos:<タグ>

docker imageで確認できる

docker push

イメージをDockerHubにアップ

オリジナルイメージを自分のアカウントにアップ

*要ログイン

docker history

イメージの履歴を確認

他人が作ったイメージの中身を知りたいときに使う

Dockerfileがない時に使うコマンド

セクション5

2023年8月3日 10:11



README

コンテナボリューム

2023年8月3日 10:15

コンテナボリューム

Docker運用で一番迷う部分

コンテナ内のボリュームは消える

→ データベースなどの永続データには
使えない??

永続化の実現方法

ホストとディレクトリ共有で解決

→ ホストのディレクトリ
& コンテナのディレクトリ共有

コンテナが削除されてもホストに残る

永続DB

2023年8月3日 10:16

ホストマシンのディスクに書き込み

ホスト：（Dockerをインストールしたマシン）

ボリュームマウント

2023年8月3日 10:18

具体的なボリューム共有

docker run -v /Users/uchidayuma/source:/var/www/html イメージ名



/Users/uchidayuma/source ↔ /var/www/html

オリジナルHTML

2023年8月3日 10:19

```
$ docker run --name mynginx -p 8080:80 nginx:1.16
```

8080:80 ←外側からアクセスするポート：コンテナ内からアクセスするポート

/home/ec2-user/test/index.html → 表示内容をかく

```
$ docker run -v /home/ec2-user/test:/usr/share/nginx/html --name mynginx -p 8080:80 nginx:1.16
```

※/home/ec2-user/test:/usr/share/nginx/html ← nginxのドキュメントルートにファイルを共有する

```
$ docker exec -it mynginx /bin/bash
```

```
# cd /usr/share/nginx/html
```

```
# cat index.html
```

→内容がみれる

本番環境

2023年8月3日 10:40

問題点

- ・ サーバーのストレージと共通する←サーバー 1 つ 1 つと共通するとサーバーによってデータが違う

解決方法：Amazon aurora ←高速

セクション6

2023年8月3日 10:44

Dockerfile

2023年8月3日 10:42

Dockerイメージをコード化したもの

Dockerfileを使うと・・・

- ・ ファイルを読むと構成がわかる
- ・ オリジナルイメージを作成できる
- ・ 設定ファイルなども変更できる

例)



ベストプラクティス

2023年8月3日 10:44

https://docs.docker.jp/engine/articles/dockerfile_best-practice.html

コンテナはエフェメラルである（コンテナに常態を持たない）
エフェメラル・・・停止・破棄可能であり、明らかに最小のセットアップで構築して使える

余計なファイルを置かない

不要なパッケージのインストールを避ける

コンテナ毎に一つのプロセスだけ実行（連携させて使用する）

レイヤの数を最小に

複数行の引数（読みやすい工夫）

RUNとCMD

2023年8月3日 10:51

RUNとCMD

どちらもコマンドを実行

→しかし、実行タイミングが異なる

RUN : DockerfileからDockerimageにビルドする時に一回だけ実行するコマンド

CMD : Dockerfileからできたイメージをコンテナ化するときに実行するコマンド

例) RUN apt-get install -y nginx (Dockerfile→イメージ の際にあらかじめインストールしておきたい)

CMD ["nginx","-g","daemon off;"] (-g : nginxを起動する daemon off : フォアグラウンドで稼働)

CMDのexec形式

Json配列方式なのでコマンドを""で囲う必要がある

vi /run/Dockerfile

FROM ubuntu:20.04

RUN apt-get update -y && ¥

apt-get install -y nginx

CMD ["nginx", "-g", "daemon off;"]

\$ docker build -t dockerfile-run-nginx run ※-t : 名前をつける

→runの中にあるDockerfileのビルドが始まる

\$ docker run -d -p 8081:80 --name dockerfile-run-nginx dockerfile-run-nginx

\$ docker ps

→8081番でnginxが起動していることを確認する

\$ curl <http://localhost:8081>

COPYとADD

2023年8月3日 16:11

どちらもファイルをイメージに追加するコマンド
→ADDはネット経由でも追加できる
基本はローカルからの追加なので、COPYを推奨

```
$ vi /copy/Dockerfile
FROM ubuntu:20.04
RUN apt-get update -y && \
    apt-get install -y nginx
COPY index.html /var/www/html
CMD ["nginx", "-g", "daemon off;"]
```

```
$ docker run -d -p 8082:80 --name dockerfile-copy-nginx dockerfile-copy-nginx
```


ENV

2023年8月3日 16:47

環境変数を設定

- ・ DBのユーザー名など
- ・ 動作環境 (localなど)

ただし、直接書き込みのみで固定値になってしまう。

(ユーザーによって変えられない)

```
$ sudo vi env/Dockerfile
```

```
FROM ubuntu:20.04
```

```
RUN apt-get update -y && \
```

```
    apt-get install -y nginx
```

```
ENV TESTENV="Uchida"
```

```
ENV APP_ENV="production"
```

```
#よく使うのが、local,production
```

```
CMD ["nginx", "-g", "daemon off;"]
```

```
docker build -t dockerfile-env-nginx env
```

```
$ docker run -d -p 8083:80 --name dockerfile-env-nginx dockerfile-env-nginx
```

```
$ docker inspect → 確認できる
```

```
"Config": {
  "Hostname": "354505347dd6",
  "Domainname": "",
  "User": "",
  "AttachStdin": false,
  "AttachStdout": false,
  "AttachStderr": false,
  "ExposedPorts": {
    "80/tcp": {}
  },
  "Tty": false,
  "OpenStdin": false,
  "StdinOnce": false,
  "Env": [
    "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
    "TESTENV=Uchida",
    "APP_ENV=production"
  ]
}
```

Docker メリット

2023年8月3日 17:50

1. コード化されたファイルを共有することで、どこでも誰でも同じ環境が作れる。
2. 作成した環境を配布しやすい。
3. スクラップ & ビルドが容易にできる。

アプリケーションが動く状態にしたものをひとまとめにしたもの

Dockerさえ入っていればコンテナ

コンテナで完結しているので依存するパッケージのバージョンに左右されない

故障してもコンテナを入れなおせば動くようになる

OSのバージョンが違ってても

pullしてそのまま使える

コンテナ 1 からコンテナ 2 のデータを取りに行くことはできない それぞれ分離

データを残さない → DBなどで困る 共有してデータをはき出しておく

カスタムMariaDBイメージ

2023年8月3日 17:06

mariadb/Dockerfile

```
FROM mariadb:10.4
RUN apt-get update -y
COPY my.conf /etc/mysql/conf.d
COPY create-table.sql /docker-entrypoint-initdb.d
ENV MYSQL_USER=root
ENV MYSQL_DATABASE=docker
ENV MYSQL_ROOT_PASSWORD=root
```

create-tables.sql

```
CREATE TABLE persons (
  id int,
  lastname varchar(255),
  firstname varchar(255),
  address varchar(255),
  city varchar(255)
);
```

my.conf

```
[client]                # clientセクション: mysqlクライアントツールへの設定
port=3306
socket=/tmp/mysql.sock
```

```
[mysqld]                # mysqldセクション: mysqlサーバーへの設定
port=3306
socket=/tmp/mysql.sock
key_buffer_size=16M
max_allowed_packet=8M
```

```
[mysqldump]            # mysqldumpセクション: バックアップコマンドへの設定
quick
```

```
[mysqld_safe]          # mysqld_safeセクション: 起動ファイル設定  
log-error=/var/log/mysqld.log  
pid-file=/var/run/mysqld/mysqld.pid
```

```
$ docker run -d --name mymariadb mymariadb
```

```
$ docker images ← 確認
```

```
$ docker exec -it mymariadb /bin/bash ← コンテナに入る
```

```
#mysql -u root -p      ※パスワードは環境変数でrootになるようにかかっている
```

```
[(none)] show databases; → dockerが作られている
```

```
[(none)] user docker;
```

```
[(none)] show tables; → personsが作られている
```

コンテナとイメージの違い

2023年8月3日 17:55

<https://www.kagoya.jp/howto/cloud/container/dockerimage/#:~:text=Docker%E3%82%A4%E3%83%A1%E3%83%BC%E3%82%B8%E3%82%B3%E3%83%B3%E3%83%86%E3%83%8A%E3%81%AE%E5%8B%95%E4%BD%9C,Docker%E3%82%A4%E3%83%A1%E3%83%BC%E3%82%B8%E3%81%8C%E5%BF%85%E8%A6%81%E3%81%A7%E3%81%99%E3%80%82>

インスタンス作成

2023年8月2日 17:21

ロールの切り替え→LabUserRole

SandBox . . .

DeveloperAccessExt

t3.small

ws-keypair

VPC - 必須 情報

vpc-0053d35eaada0023a (prod-handson-vpc01)
10.18.48.0/20

サブネット 情報

subnet-0ce10186f8935105e prod-handson-vpc01-sub-prv01a
VPC: vpc-0053d35eaada0023a 所有者: 157094121738
アベイラビリティゾーン: ap-northeast-1a 利用可能な IP アドレス: 250
CIDR: 10.18.50.0/24

インスタンス ID
i-0c0da61326b2a3bde (wslab1-srv1)

IAM ロール
インスタンスにアタッチする IAM ロールを選択するか、まだ作成していない場合は新しいロールを作成します。選択したロールによって、現在インスタンスにアタッチされているロールが置き換えられます。

WS_Lab_EC2Role

新しい IAM ロールを作成

キャンセル IAM ロールの更新

システムズマネージャーへの権限を許可
→セッションマネージャーから接続できる

sudo su - ec2-user

セクション1

2023年8月4日 8:06

https://github.com/uchidayuma/udemy-laravel8-mysql-simple-memo/tree/feature_ecs



EKS

kubernetes・・・Googleの全てのサービスに利用
20億個のコンテナがある

Amazon ECS・・・AWS独自のコンテナオーケストレーションツール（今回はこっちを使用する）
GUIで操作できる

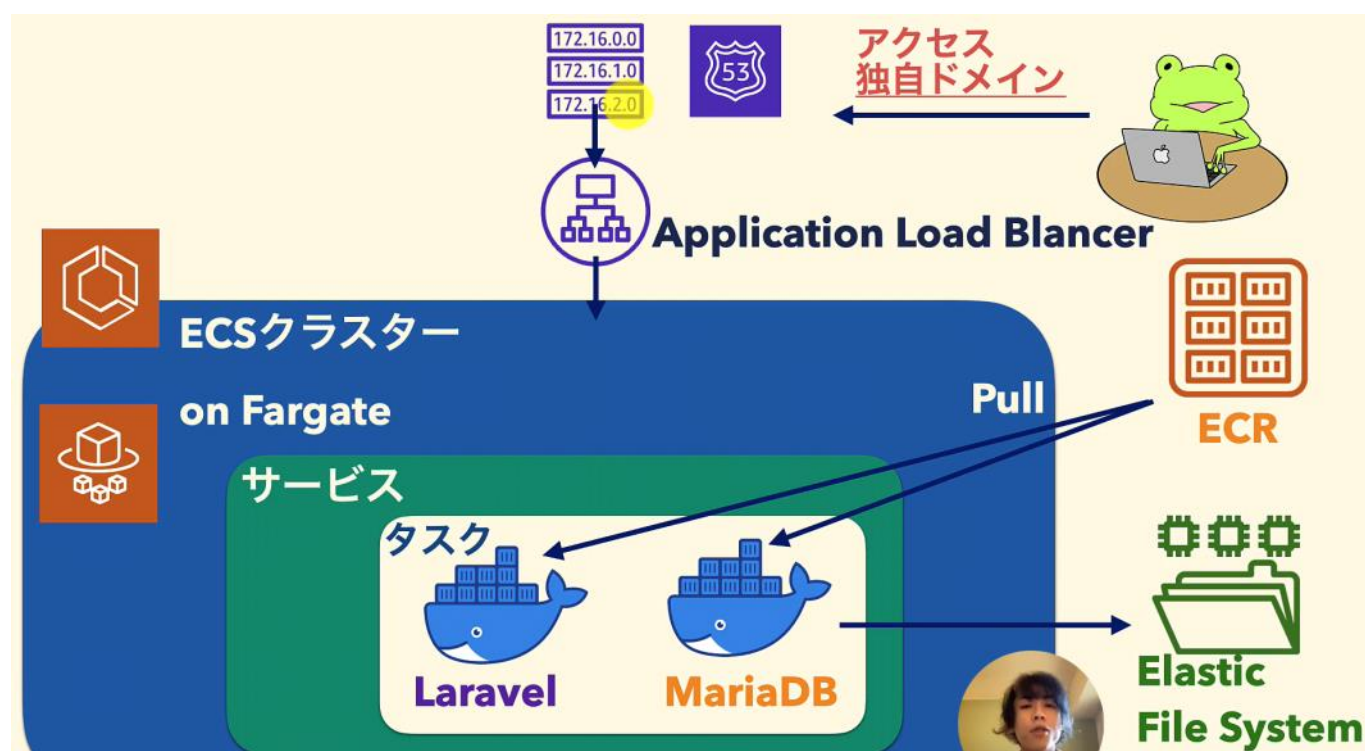
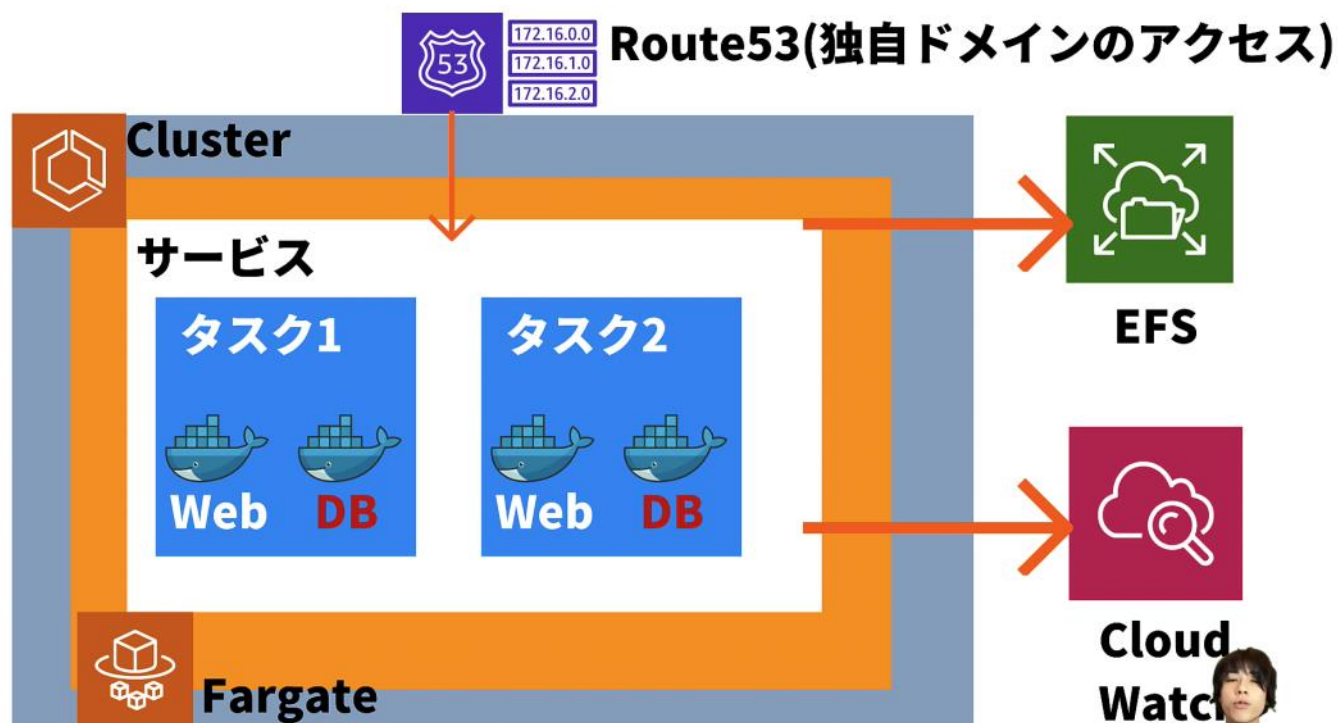
Dockerのコンテナサービス

様々なシステムやサービスの配置/設定/管理を自動化してくれるオーケストレーション

	
<h2>kubernetes</h2>	<h2>ECS</h2>
<ul style="list-style-type: none">• 大規模アプリ向け• 管理用サーバーが必要• 設定ファイルが細かい• 運用コスト：高い	<ul style="list-style-type: none">• サーバー1台～• 管理用サーバーは不要• ブラウザから設定も可• 運用コスト：低い

全体像

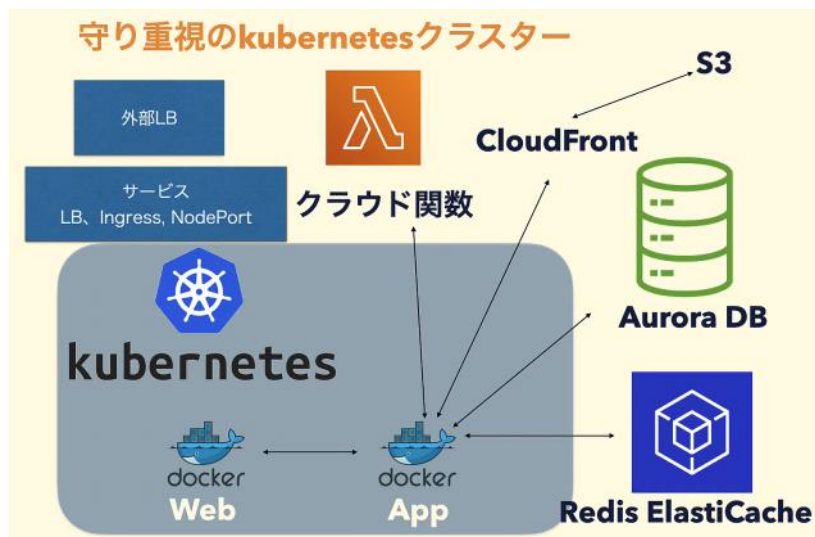
2023年8月4日 8:14



Route53にドメインでアクセスがきたらApplication Load Blancerに流す → ECSクラスターと繋ぐ
ECR・・・コンテナのレジストリ
Elastic File System・・・データの永続化
Fargate・・・サーバーレス
サービス・・・タスクの管理者

データベースを外部に任せる

2023年8月4日 8:23



セクション3

2023年8月4日 8:33

なぜDockerで本番運用するのか

様々なメリットとカプセル化の関係



品質↑



構築コスト↓



人に依存しない



コード化

なぜDockerで本番運用するのか

様々なメリットとカプセル化の関係



テスト
環境



ロール作成

「IAM」 → 「ロールを作成」

Elastic Container Serviceの4つそれぞれ作成する

他の AWS のサービスのユースケース:

Elastic Container Service

▼

- ☐ Elastic Container Service
Allows ECS to create and manage AWS resources on your behalf.
- ☐ Elastic Container Service Autoscale
Allows Auto Scaling to access and update ECS services.
- ☐ Elastic Container Service Task
Allows ECS tasks to call AWS services on your behalf.
- ☐ EC2 Role for Elastic Container Service
Allows EC2 instances in an ECS cluster to access ECS.

Elastic Container Service Taskはポリシーを[AmazonECSTaskExecutionRolePolicy](#)のみにする

EC2インスタンス

2023年8月4日 9:26

```
sudo yum install docker
```

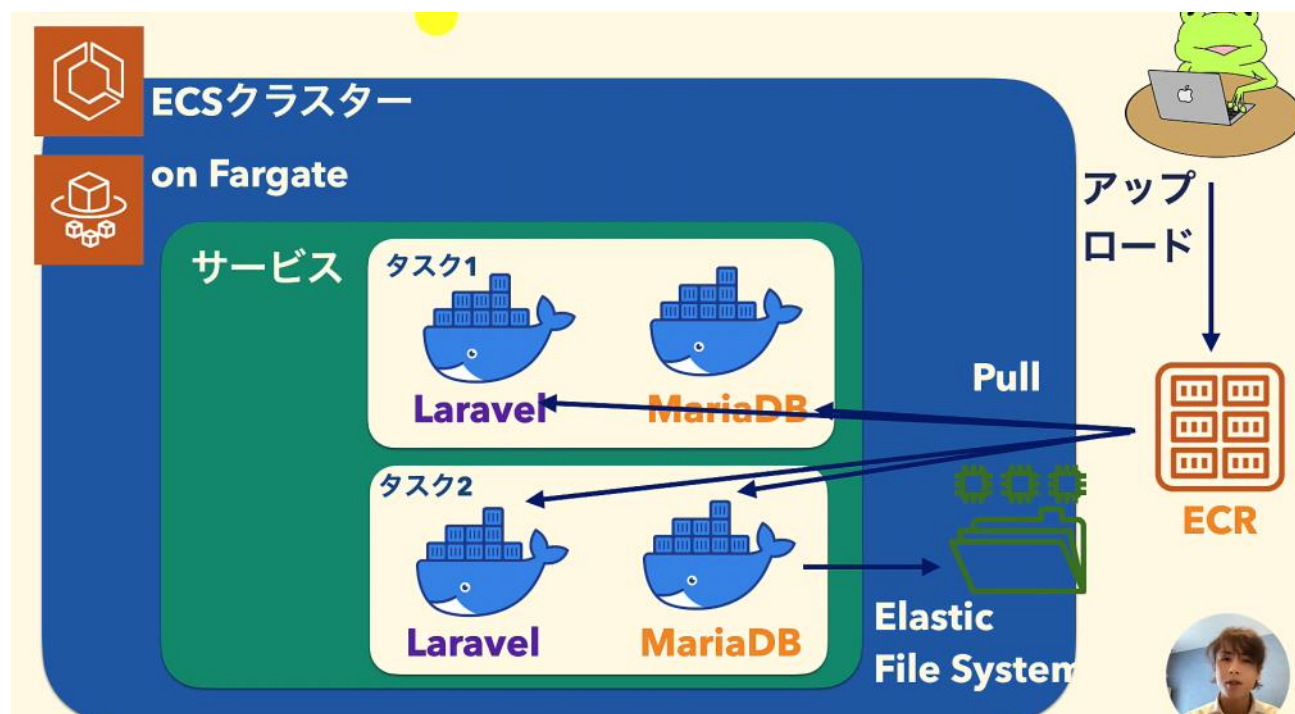
```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"
```

```
unzip awscliv2.zip
```

```
sudo ./aws/install
```

セッション5

2023年8月4日 9:33



ローカルのパソコン→ECRにアップロード→ECSクラスターにPull

Laravel・MariaDBを配置する ←お互いが通信を行いデータベースと連携する

タスクを複数割り当てる ←アクセスが多くなったら増やす（水平スケーリング）

ECS

2023年8月9日 9:13

Dockerのコンテナサービス

様々なシステムやサービスの配置/設定/管理を自動化してくれるオーケストレーション

ECS料金

2023年8月9日 10:16

ECSが立ち上げたEC2インスタンスの料金

サービスが立ち上がっていてもインスタンスが立ち上がっている時間で課金

Fargateの場合はサービスが稼働している時間で課金される

タスクを0にしておけば課金されない

同じ環境で使用する場合はFargateの方が割高

Fargate

2023年8月4日 10:04

<https://www.sunnyccloud.jp/column/20230303-01/>

AWS Fargateは、Amazon EC2インスタンスによるコンテナ実行環境をフルマネージドで自動化するサービス

[AWS Fargate](#)とは、AWS上でコンテナをサーバーレスで実行することが出来るようにする機能を提供してくれるサービスです。サーバ運用で必要となるOSのメンテナンス等の管理が不要になることでアプリケーション開発や構築に集中して取り組むことが出来るようになります。AWS Fargateは、データプレーン（コンテナの実行環境）としてECSやEKSと組み合わせて利用します。

特徴

- ・AWSマネージドで、EC2インスタンスのプロビジョニング、スケール、管理が不要
- ・ワークロード使用量によって課金
- ・自動的なコンピューティングのスケールリング
- ・他のAWSサービス（VPCネットワーキング、ELB、IAM、Cloud Watchなど）と連携可能

弱点

- ・EC2のようにサーバーへアクセスできない
- ・dockerコマンドが使えない
- ・デバッグがやりづらい(ログを確認するためにコンテナに入るのが大変だから)
- ・初心者にはイメージしづらい

Fargate・・・管理サーバの実態がない



EC2

オートスケーリングでインスタンスを立ち上げて実行環境に入れることができる

ネットワークモード：awsvpc（コンテナに自動でeniなどを紐付けてくれるが、インスタンスの場合はアタッチできるコンテナの数に限りがある）
bridge（インスタンスと同じアドレスでアクセスできるが、同じポートを使用できない）

docker run

ECSクラスター (EC2)

EC2 1台目



タスク1



Laravel



MariaDB

タスク2



Laravel



MariaDB

EC2 2台目



タスク3



Laravel



MariaDB

タスク4



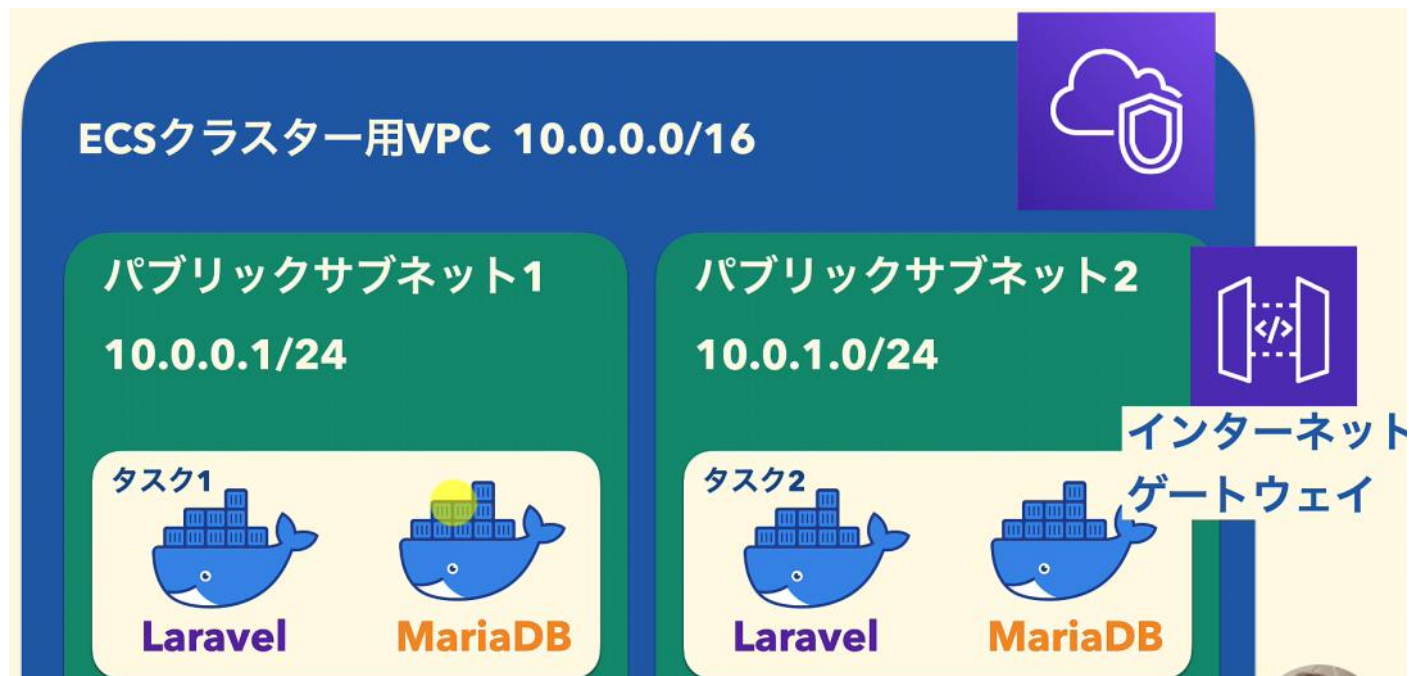
Laravel



MariaDB

コンテナ配置

2023年8月4日 10:06



①Dockerイメージを作成

ローカル環境で、本番環境用の

Dockerfile作成→ビルドでイメージ化

②ECRにアップロード

作成したDockerfileをコンテナレジストリにアップロード

③ECSにPULL

ECSタスクにPULLを行う

→後は自動的にdocker run される

Dockerfile



2023年8月4日 10:21

udemy-lara
vel8-mys...

#phpとwebサーバーが組合わせられたものを使用

FROM php:7.4.24-apache

PHPのモジュールなどをインストール

RUN apt-get update ¥

&& apt-get install -y zlib1g-dev ¥

&& apt-get install -y zip unzip ¥

&& apt-get -y install libzip-dev libonig-dev ¥

&& docker-php-ext-install pdo_mysql mysqli zip ¥

&& docker-php-ext-enable pdo_mysql mysqli ¥

&& a2enmod rewrite

タイムゾーン設定

ENV TZ=Asia/Tokyo

cronのインストール

RUN apt-get update && apt-get install -y ¥

busybox-static ¥

&& apt-get clean

composerをインストール Laravelを使うのに必要

RUN curl -sS <https://getcomposer.org/installer> | php -- --install-dir=/usr/bin/ --filename=composer

ENV COMPOSER_ALLOW_SUPERUSER 1

ENV COMPOSER_HOME /composer

ENV PATH \$PATH:/composer/vendor/bin

アプリケーションフォルダを環境変数として設定

ENV APP_HOME /var/www/html

apacheのuidとgidをdocker user uid/gidに変更。

RUN usermod -u 1000 www-data && groupmod -g 1000 www-data

#change the web_root to laravel /var/www/html/public folder

RUN sed -i -e "s/html/html¥/public/g" /etc/apache2/sites-enabled/000-default.conf

COPY ./php/vhost.conf /etc/apache2/conf-enabled/vhost.conf

apache module rewrite を有効にする

RUN a2enmod rewrite

ソースコードと.envファイルをDockerImageに埋め込む

COPY . \$APP_HOME

COPY .env.production /var/www/html/.env

初回起動時に行うスクリプトファイルをコピーして実行権限を与える

COPY ./php/start.sh /var/www/html/start.sh

RUN chmod 744 ./php/start.sh

必ずキャッシュ用のディレクトリを作っておくこと→Fargateの場合ずっとキャッシュが残ることになる

RUN mkdir bootstrap/sessions && ¥

mkdir storage/framework/cache/data

フレームワークに必要なモジュールをDockerImageにインストール

RUN composer install --no-dev --no-interaction

書き込み権限を与える

RUN chown -R www-data:www-data \$APP_HOME

RUN chmod -R 777 storage && ¥

chmod -R 777 bootstrap

CMD php artisan migrate --force

CMD ["bash", "start.sh"]

ECR

2023年8月4日 14:14

<https://dev.classmethod.jp/articles/re-introduction-2022-ecr/>

「ECS」 → 「リポジトリ」

リポジトリを作成する

可視性設定：プライベート

イメージスキャナ：有効

lab2-laravelecs のプッシュコマンド

- macOS / Linux
- Windows

AWS CLI および Docker 向けの最新バージョンがインストールされていることを確認します。詳細については、[Amazon ECR の開始方法](#) を参照してください。

次の手順を使用して、リポジトリに対してイメージを認証し、プッシュします。Amazon ECR 認証情報ヘルパーなどの追加のレジストリ認証方法については、[レジストリの認証](#) を参照してください。

1. 認証トークンを取得し、レジストリに対して Docker クライアントを認証します。
AWS CLI を使用する

```
aws ecr get-login-password --region ap-northeast-1 | docker login --username AWS --password-stdin 157094121738.dkr.ecr.ap-northeast-1.amazonaws.com
```

注意: AWS CLI の使用中にエラーが発生した場合は、最新バージョンの AWS CLI と Docker がインストールされていることを確認してください。

2. 以下のコマンドを使用して、Docker イメージを構築します。一から Docker ファイルを構築する方法については、「[こちらをクリック](#)」の手順を参照してください。既にイメージが構築されている場合は、このステップをスキップします。

```
docker build -t lab2-laravelecs .
```

3. 構築が完了したら、このリポジトリにイメージをプッシュできるように、イメージにタグを付けます。

```
docker tag lab2-laravelecs:latest 157094121738.dkr.ecr.ap-northeast-1.amazonaws.com/lab2-laravelecs:latest
```

4. 以下のコマンドを実行して、新しく作成した AWS リポジトリにこのイメージをプッシュします:

```
docker push 157094121738.dkr.ecr.ap-northeast-1.amazonaws.com/lab2-laravelecs:latest
```

手順

2023年8月4日 14:18



udemy-lara
vel8-mys...

ロールを割り当てる
put imageの権限が必要

```
$ aws ecr get-login-password --region ap-northeast-1 | docker login --username AWS --password-stdin  
157094121738.dkr.ecr.ap-northeast-1.amazonaws.com
```

```
$ sudo aws s3 sync s3://lab23-docker/udemy-laravel8-mysql-simple-memo-feature_ecs /home/ec2-user/source/
```

```
$ docker build -t laravelecs .
```

タグ付け

```
$ docker tag laravelecs:latest 157094121738.dkr.ecr.ap-northeast-1.amazonaws.com/lab2-laravelecs:latest
```

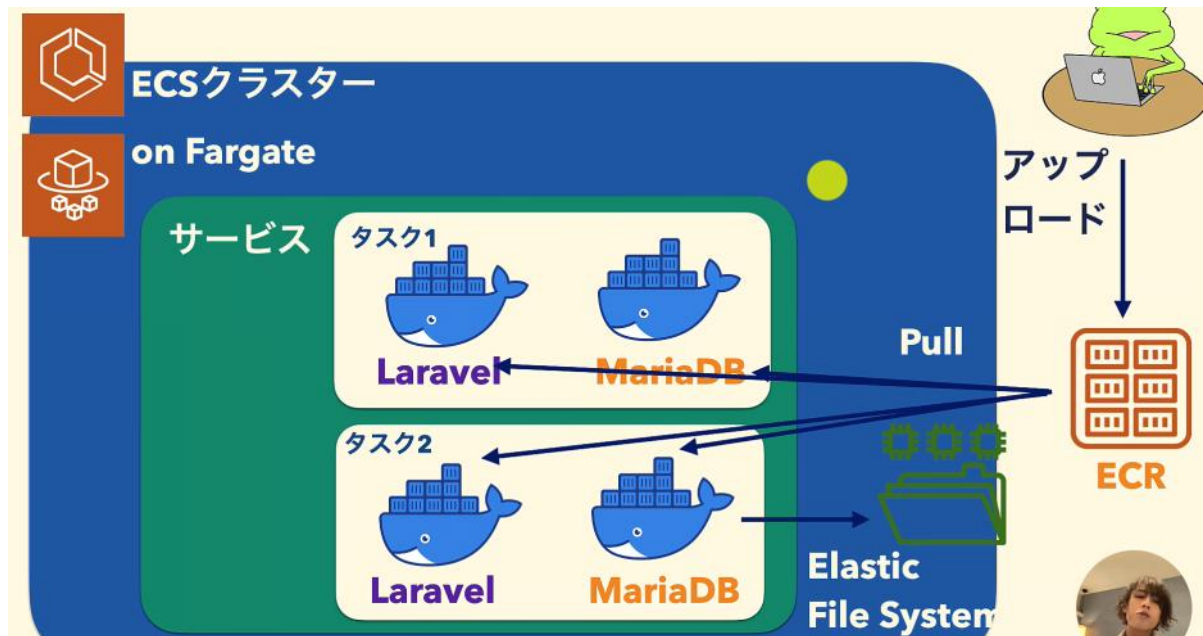
push

```
$ docker push 157094121738.dkr.ecr.ap-northeast-1.amazonaws.com/lab2-laravelecs:latest
```

クラスター作成

2023年8月4日 15:30

プライベートサブネットに配置



- ①サービスを立ち上げる
- ②サービス経由でタスクを立ち上げる

タスク定義

2023年8月4日 15:49

タスク実行ロールにecsTaskExecutionRoleを設定する

デプロイ方法

「ECS」→「クラスター」→「クラスター名」→「サービスを作成」

「クラスター」タブの「タスク」

サービス作成した際にタスクが実行されない

対策①：先にロググループを作成しておく

対策②：TaskロールにCreateの権限をアタッチしておく

コンテナ設定

必須コンテナ：そのコンテナがないと立ち上がらない

ポートマッピング：下の図の場合

コンテナ1・・・ホストポート80 コンテナポート80

コンテナ2・・・ホストポート8080 コンテナポート80

環境変数：コンテナによって必要なものは違う

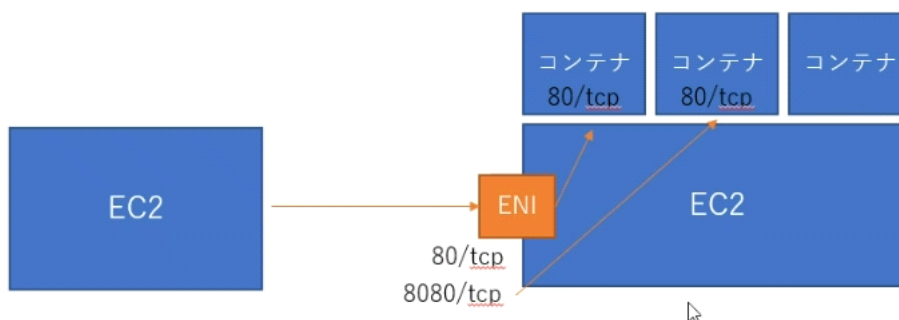
エラーの時はログを確認する

ログの収集：エラー時に使うのでチェックをいれておく

ヘルスチェック：コマンドを実行して成功するかを確認する

ストレージ：EFSのパスなどを指定する

コンテナパス：マウント先のパス（そのパスを参照すればデータをとることができる）



Session Managerプラグイン

2023年8月7日 15:05

インストール https://docs.aws.amazon.com/ja_jp/systems-manager/latest/userguide/session-manager-working-with-install-plugin.html#install-plugin-windows

\$ sudo yum install -y https://s3.amazonaws.com/session-manager-downloads/plugin/latest/linux_64bit/session-manager-plugin.rpm

確認

\$ session-manager-plugin

Fargateでコンテナに入る

2023年8月7日 15:18

トラブルシューティングの際に入ることが多い
設定を追加することは基本的にはない（コンテナを立ち上げ直したら消えてしまうので）

[ロール作成](#)のTaskにポリシーをアタッチする

Json

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ssmmessages:CreateControlChannel",
        "ssmmessages:CreateDataChannel",
        "ssmmessages:OpenControlChannel",
        "ssmmessages:OpenDataChannel"
      ],
      "Resource": "*"
    }
  ]
}
```

サービスに対して許可をだす

```
$ aws ecs update-service --region ap-northeast-1 --cluster lab2-simple-memo-ecs --
service lab2-simple-memo-service --enable-execute-command
```

Shift + G

```
"enableECSTags": true,
"propagateTags": "NONE",
"enableExecuteCommand": true
```

「スラスター」→「サービス」→「サービスの更新」（強制デプロイにチェックを入れる）

```
$ sudo aws ecs execute-command --region ap-northeast-1 --cluster lab2-simple-memo-ecs --task 12d5376d52c14c3e91fb4956c7624723 --container laravel --interactive --command "/bin/sh"
```

以下のエラーが起こるときは

An error occurred (AccessDeniedException) when calling the DescribeTasks operation: User: arn:aws:sts::157094121738:assumed-role/WS_Lab_EC2Role/i-0159a6085ecfb45d7 is not authorized to perform: **ecs:DescribeTasks** on resource: arn:aws:ecs:ap-northeast-1:157094121738:task/lab2-simple-memo-ecs/12d5376d52c14c3e91fb4956c7624723 because no identity-based policy allows the ecs:DescribeTasks action

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ssmmessages:CreateControlChannel",
        "ssmmessages:CreateDataChannel",
        "ssmmessages:OpenControlChannel",
        "ssmmessages:OpenDataChannel",
        "ecs:ExecuteCommand",
        "ecs:DescribeTasks"
      ],
      "Resource": "*"
    }
  ]
}
```

セクション6

2023年8月8日 8:15

MariaDB

2023年8月8日 8:15

ECRでリポジトリを作成

プッシュコマンドを表示から

※2.を実行するカレントディレクトリに注意する

AWS CLI および Docker 向けの最新バージョンがインストールされていることを確認します。詳細については、[Amazon ECR の開始方法](#) を参照してください。

次の手順を使用して、リポジトリに対してイメージを認証し、プッシュします。Amazon ECR 認証情報ヘルパーなどの追加のレジストリ認証方法については、[レジストリの認証](#) を参照してください。

1. 認証トークンを取得し、レジストリに対して Docker クライアントを認証します。
AWS CLI を使用する

```
aws ecr get-login-password --region ap-northeast-1 | docker login --username AWS --password stdin 157094121738.dkr.ecr.ap-northeast-1.amazonaws.com
```

注意: AWS CLI の使用中にエラーが発生した場合は、最新バージョンの AWS CLI と Docker がインストールされていることを確認してください。

2. 以下のコマンドを使用して、Docker イメージを構築します。一から Docker ファイルを構築する方法については、「[こちらをクリック](#)」の手順を参照してください。既にイメージが構築されている場合は、このステップをスキップします。

```
docker build -t lab2-mariadbecs .
```

3. 構築が完了したら、このリポジトリにイメージをプッシュできるように、イメージにタグを付けます。

```
docker tag lab2-mariadbecs:latest 157094121738.dkr.ecr.ap-northeast-1.amazonaws.com/lab2-mariadbecs:latest
```

4. 以下のコマンドを実行して、新しく作成した AWS リポジトリにこのイメージをプッシュします:

```
docker push 157094121738.dkr.ecr.ap-northeast-1.amazonaws.com/lab2-mariadbecs:latest
```

ECRをコンテナにデプロイ

2023年8月8日 8:40

「タスク定義」→「新しいリビジョンの定義」

mariadbのコンテナの追加をする

ポート：3306

ヘルスチェック：コマンドが実行して成功したらHealthyになる

環境変数追加

▼ 環境変数 - オプション 情報

個別に追加

キーと値のペアを追加して、環境変数を指定します。

キー	値のタイプ	値	
MYSQL_DATABASE	値 ▼	simplememo	削除
MYSQL_USER	値 ▼	dbuser	削除
MYSQL_PASSWORD	値 ▼	simplememodbuser	削除
MYSQL_ROOT_PASSWORD	値 ▼	password	削除

ヘルスチェック

▼ HealthCheck - オプション 情報

コマンド

コンテナが正常かどうかを判断するために実行されるコマンドのカンマ区切りのリストを入力します。リストは、タスク定義の JSON ファイルの文字列配列に自動的に変換されます。

CMD-SHELL, mysqladmin ping -u dbuser -psimplememodbuser -h 127.0.0.1 || exit 1

間隔

各ヘルスチェック検証の間の期間 (秒)。有効な値は 5～300 です。デフォルト値は 30 です。

10

秒

タイムアウト

ヘルスチェックが成功するまでの待機時間 (失敗とみなされるまでの時間) (秒)。有効な値は 2～60 です。デフォルト値は 5 です。

10

秒

laravelのスタートアップオプションを変更する

▼ スタートアップの依存関係の順序 - オプション

順序を設定 [情報](#)

タスク定義内のコンテナを開始する順序を制御します。 [ドキュメントを表示](#)

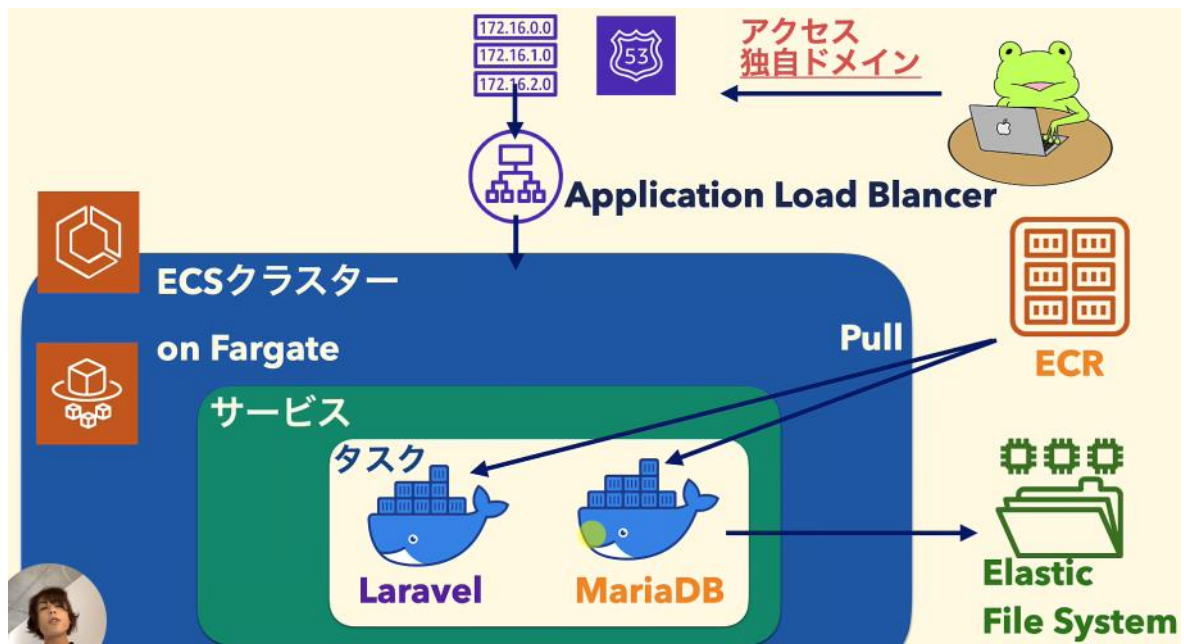
コンテナ名

条件

lab2-mariadb

Healthy

削除



データベースの永続化

2023年8月8日 9:12

コンテナは削除されたらデータは消える
→Elastic File Systemに一旦保存しておく

EFSでデータを共通化

2023年8月8日 9:20

「EFS」→ファイルシステムの作成

コンテナは削除されるとデータも消えてしまうのでEFSに出力しておく

ファイルを共有するためのシステム（nfsプロトコルを使用 AWSverだとEFSという）

マウントする

アタッチ

Linux インスタンスに Amazon EFS ファイルシステムをマウントします。 [詳細はこちら](#)

☒ DNS 経由でマウント

☐ IP 経由でマウント

EFS マウントヘルパーの使用:

```
sudo mount -t efs -o tls fs-071ff754a0d05da62:/ efs
```

NFS クライアントの使用:

```
sudo mount -t nfs4 -o nfsvers=4.1,rsize=1048576,wsiz=1048576,hard,timeo=600,retrans=2,noresvport fs-071ff754a0d05da62.efs.ap-northeast-1.amazonaws.com:/ efs
```

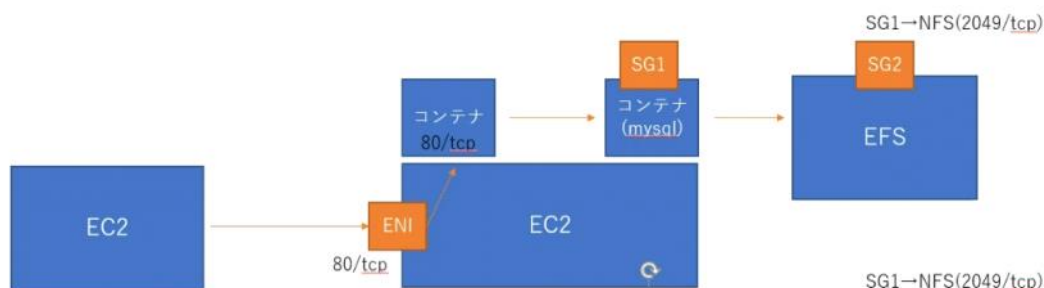
Task実行ロールに「AmazonEFSCSIDriverPolicy」を追加する

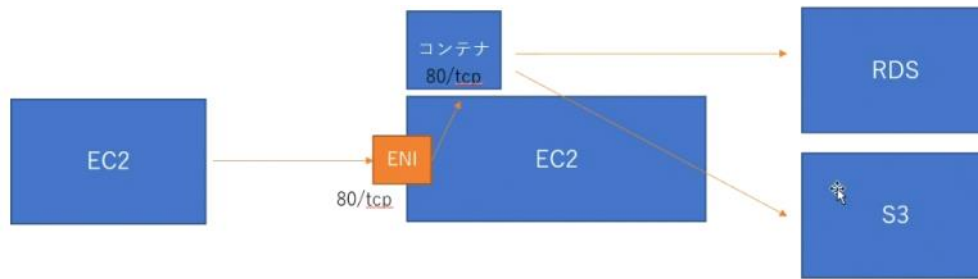
ネットワークタブからセキュリティグループを設定する

インバウンドルール 情報

セキュリティグループ ID	タイプ	プロトコル	ポート範囲	ソース	説明 - オプション
sgr-0cf08b7196bf35ae2	NFS	TCP	2049	カスタム	削除
sgr-065d2771d9b169d35	HTTP	TCP	80	カスタム	削除

ソースにサービスに割り当てているセキュリティグループを選択





```
$ aws ecs execute-command --region ap-northeast-1 --cluster lab2-simple-memo-ecs --task 424bba6cf6374eb380f9b7d89343a6d5 --container lab2-mariadb --interactive --command "/bin/sh"
```

ACM証明書



2023年8月8日 11:13

リクエスト

ゾーンのドメイン名で

CNAMEをゾーンに作成する

↓この値をゾーンでCNAMEレコードに登録する

CNAME 名	CNAME 値
 _29fdb9f9b70 653c6b219ddc 3b3627bb1.lab 2.ahaws.toyota - bibliotheca.co m.	 _e09228475c1 cf569e6acc568 4cec6315.nbgf hbpblk.acm- validations.aws .

セクション7

2023年8月8日 16:10

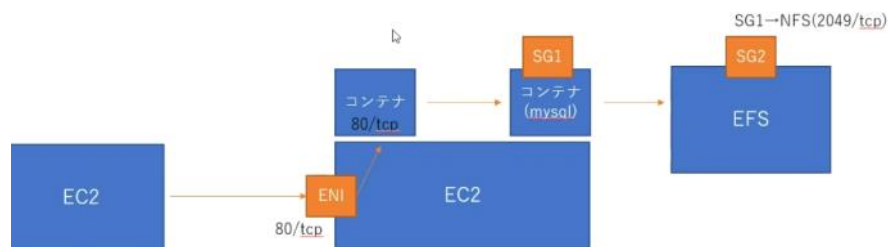
ロードバランサー

2023年8月8日 11:51

ターゲットグループの作成

基本的な設定：IPアドレス

ヘルスチェック：/login → アクセスしたときにエラーが返ってきたらやり直す



ロードバランサーの作成

※パブリックサブネットに作成する必要がある

インバウンドルール 情報

セキュリティグループルール ID	タイプ 情報	プロトコル 情報	ポート範囲 情報	ソース 情報	説明 - オプション 情報
sgr-0cf08b7196bf35ae2	NFS ▼	TCP	2049	カスタム ▼ Q	削除
				sg-0876c84847731eb3f ✕	
sgr-065d2771d9b169d35	HTTP ▼	TCP	80	カスタム ▼ Q	削除
				sg-0876c84847731eb3f ✕	
sgr-0c5e526d575fceded	HTTPS ▼	TCP	443	カスタム ▼ Q	削除
				pl-06a3ac4b9417d8ca5 ✕	

▼ リスナー HTTP:80

削除

プロトコル	ポート	デフォルトアクション	情報
HTTP ▼	: 80 1~65535	転送先 lab2-ecs-simplememo ターゲットの種類: IP, IPv4	HTTP ▼
ターゲットグループの作成			

リスナータグ - 省略可能

リスナーにタグを追加することを検討してください。タグを使用すると、AWS リソースを分類できるため、リソースをより簡単に管理できます。

リスナータグの追加

タグは最大 50 個追加できます。

▼ リスナー HTTPS:443

削除

プロトコル	ポート	デフォルトアクション	情報
HTTPS ▼	: 443 1~65535	転送先 lab2-ecs-simplememo ターゲットの種類: IP, IPv4	HTTP ▼
ターゲットグループの作成			

作成したACMを読み込ませる

サービス作成

2023年8月8日 16:39

デプロイ設定

アプリケーションタイプ [情報](#)

実行するアプリケーションのタイプを指定します。

☒ サービス

停止して再起動できる長時間のコンピューティング作業を処理するタスクグループを起動します。例としては、ウェブアプリケーションが挙げられます。

☐ タスク

実行および終了するスタンドアロンタスクを起動します。例としては、バッチジョブが挙げられます。

タスク定義

既存のタスク定義を選択します。新しいタスク定義を作成するには、[タスク定義](#) にアクセスしてください。

☐ リビジョンの手動指定

選択したタスク定義ファミリーに最新の 100 個のリビジョンを選択する代わりに、リビジョンを手動で入力します。

ファミリー

lab2-simple-memo ▼

リビジョン

14 (最新) ▼

サービス名

このサービスに一意の名前を割り当てます。

alb-service ▼

▼ ロードバランシング - オプション

ロードバランサーの種類 [情報](#)

サービスで実行されているタスク間で着信トラフィックを分散するようにロードバランサーを設定します。

Application Load Balancer ▼

Application Load Balancer

新しいロードバランサーを作成するか、既存のロードバランサーを選択するかを指定します。

☐ 新しいロードバランサーの作成

☒ 既存のロードバランサーを使用

ロードバランサー

サービスで実行されているタスク間で着信トラフィックを分散するために使用するロードバランサーを選択します。

lab2-ecs-alb ▼

ロードバランス用のコンテナの選択

lab2-laravel ecs 80:80 ▼

リスナー [情報](#)

ロードバランサーが接続リクエストをリッスンするポートとプロトコルを指定します。

ポート

80

プロトコル

HTTP

ターゲットグループ [情報](#)

ロードバランサーが、サービス内のタスクへのルートリクエストに使用するターゲットグループを、新規に作成するか、既存のものから選択するかを指定します。

☐ 新しいターゲットグループの作成

☒ 既存のターゲットグループを使用

ターゲットグループ名

lab2-ecs-simplememo ▼

ヘルスチェックパス

/login

ヘルスチェックプロトコル

HTTP

ヘルスチェックの猶予期間 [情報](#)

0

秒

セクション8

2023年8月8日 16:42

スケーリング

2023年8月8日 16:42

「ECS」→「クラスター」→「サービス」
サービスの更新

▼ サービスの Auto Scaling - オプション

CloudWatch アラームに応じてサービスの必要数を指定範囲内で調整します。アプリケーションのニーズに合わせていつでも Service Auto Scaling の設定を変更できます。

☒ サービスのオートスケーリングを使用

Service Auto Scaling の設定を変更することで、サービスの必要数を調整する

タスクの最小数

Service Auto Scaling による調整可能な下限値。

タスクの最大数

Service Auto Scaling による調整可能な上限値。

スケーリングポリシー

[削除](#)

スケーリングポリシータイプ [情報](#)

ターゲットの追跡またはstep scalingポリシーの方針を作成します。

☒ ターゲットの追跡

特定の評価メトリクスの目標値に基づいて、サービスが実行するタスクの数を増減させます。

☐ ステップスケーリング

アラーム超過のサイズに応じて変化する、ステップ調整と呼ばれる一連の scaling 調整に基づいて、サービスが実行するタスクの数を増減させます。

ポリシー名

lab2-autoScale

ECS サービスメトリクス

ECSServiceAverageCPUUtilization ▼

ターゲット値

30

スケールアウトクールダウン期間

60

スケールインクールダウン期間

60

☐ スケールインをオフにする

メトリクス：画像は平均CPU使用率

ターゲット値：しきい値

スケールアウト：1 つタスクを増やした後に次を増やすまでのクールタイム時間

スケールダウン：しきい値を下回ったときに減らすクールタイム時間

負荷のかけ方

2023年8月8日 16:46

ab -n 5000 -c 100 <http://10.18.50.171/login> でアクセスをする

※-c：同時接続数

-n：総接続数

セキュリティ関係

2023年8月8日 16:57

セキュリティグループ（アウトバウンドは全許可）

ソース

上の3つは自分のセキュリティグループ（同じセキュリティグループが割り当たっているところからの許可）

一番下はccoc-tmc-intra

EC2インスタンス・EFSに割り当てる

セキュリティグループ ID	タイプ	プロトコル	ポート範囲	ソース	説明・オプション
sgr-0c108b7196b155ac2	NFS	TCP	2049	カスタム	削除
sgr-065d2771d9b169d55	HTTP	TCP	80	カスタム	削除
sgr-0d882c12946a9a390	HTTPS	TCP	443	カスタム	削除
sgr-0c5e526d575faced	HTTPS	TCP	443	カスタム	削除

ロール・・・最小の権限のみを与える











(EC2インスタンス)

- ・ lab2-ecsFargateExecRole・・・要らない
- ・ AmazonS3FullAccess・・・バケットからファイルなどを持ってくるときに使用
- ・ ECS関連のロール・・・必要
- ・ AmazonSSMManagedInstanceCore・・・セッションマネージャーで接続する際に必要
- ・ EC2InstanceProfileForImageBuilderECRContainerBuikds・・・ECRにpushコマンドの際に必要

ポリシー名	タイプ
lab2-ecsFargateExecRole	カスタマー管理
AmazonS3FullAccess	AWS 管理
AmazonEC2ContainerServiceforEC2Role	AWS 管理
AmazonEC2ContainerServiceRole	AWS 管理
AmazonEC2ContainerServiceAutoscaleRole	AWS 管理
AmazonECSTaskExecutionRolePolicy	AWS 管理
AmazonSSMManagedInstanceCore	AWS 管理
EC2InstanceProfileForImageBuilderECRContainerBuilds	AWS 管理

(タスク実行ロール)

- ・ CloudWatchLogsFullAccess・・・サービスの実行時にログをだすので必要（先にロググループを作成しておけば必要ない）
- ・ AmazonECSTaskExecutionRolePolicy・・・タスク実行に必要
- ・ AmazonSSMManagedInstanceCore・・・いらない？
- ・ AmazonEFSCSIDriverPolicy・・・Elastic File Systemを使用する際は必要

ポリシー名 	タイプ
 lab2-ecsFargateExecRole	カスタマー管理
  CloudWatchLogsFullAccess	AWS 管理
  AmazonECSTaskExecutionRolePolicy	AWS 管理
  AmazonSSMManagedInstanceCore	AWS 管理
  AmazonEFSCSIDriverPolicy	AWS 管理

lab2-ecsFargateExecRole

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ssmmessages:CreateControlChannel",
        "ssmmessages:CreateDataChannel",
        "ssmmessages:OpenControlChannel",
        "ssmmessages:OpenDataChannel",
        "ecs:ExecuteCommand",
        "ecs:DescribeTasks"
      ],
      "Resource": "*"
    }
  ]
}
```

プッシュコマンドでログインできないときの処理

2023年11月21日

15:50

Cloud9の状況

プッシュコマンドを入力するもエラー発生

```

NG0Reserved50:DeveloperAccess_765787770023a3b79:~/Documents$ aws ecr get-login-password --region ap-northeast-1 | docker login --username NG0 --password stdin 608728620263.dkr.ecr.ap-northeast-1.amazonaws.com
An error occurred (UnrecognizedClientException) when calling the GetAuthorizationToken operation: The security token included in the request is invalid
Error: Cannot perform an interactive login from a non TTY device

```

※ 原因（予想） → シングルサインオンの関係でaccess_keyやsecret_access_keyの認証が失敗している

封端方法


Cloud9で .aws/credentials を編集する。 → vi ../aws/credentials


```
lanReserveSSO_DeveloperAccess_76578770823a3c791~/environment $ vi ../aws/credentials
```

↓ 四角で囲われたところを編集する


[illegible]

ログインの画面を開く → Command line or programmatic accessをクリック



[keigo](#)
[MFA devices](#)
[Sign out](#)



AWS Account (2)



GitLab




Prj-AH-WorldSkillsCloud
#608728620263 | al_aws+WorldSkillsCloud@toyota-tokyo.tech

DeveloperAccess

Management console

Command line or programmatic access



Sandbox-VV-TOROHandsOn

枠線で囲われた場所をコピーする (※[~~~_DeveloperAccess]の部分もコピーされるがいない)

Get credentials for DeveloperAccess

Create access for the account **Prj-AH-WorldSkillsCloud (6087-2862-0263)** with **DeveloperAccess**

Use any of the following options to access AWS resources programmatically or from the AWS CLI. You can retrieve new credentials as often as needed.

[macOS and Linux](#) | [Windows](#) | [PowerShell](#)

▼ AWS IAM Identity Center credentials (Recommended)

To extend the duration of your credentials, we recommend you configure the AWS CLI to retrieve them automatically using the `aws configure sso` command. [Learn more](#)

SSO Start URL `https://d-956705c220.awsapps.com/start#`

SSO Region `ap-northeast-1`

▼ Option 1: Set AWS environment variables (Short-term credentials)

Run the following commands in your terminal. [Learn more](#)

```
export AWS_ACCESS_KEY_ID="ASIAV3OYNTTTUROYPO6"
export AWS_SECRET_ACCESS_KEY="SUBk0tVr/yA2a1wPPPOkXGumw3wJhD0uuhz5Wpa"
export AWS_SESSION_TOKEN="TQ0b3pZ2luX2VjELt////////wEaDmFwLW5vcnRoZWZdC0dkYwRAIgaVmw1THI"
```

▼ Option 2: Manually add a profile to your AWS credentials file (Short-term credentials)

Paste the following text in your AWS credentials file (typically located in `~/.aws/credentials`). [Learn more](#)

```
[608728620263_DeveloperAccess]
aws_access_key_id = ASIAV3OYNTTTUROYPO6
aws_secret_access_key = SUBk0tVr/yA2a1wPPPOkXGumw3wJhD0uuhz5Wpa
aws_session_token = TQ0b3pZ2luX2VjELt////////wEaDmFwLW5vcnRoZWZdC0dkYwRAIgaVmw1THI
```

▼ Option 3: Use individual values in your AWS service client (Short-term credentials)

貼り付ける

```
# Do not modify this file, if this file is modified it will not be updated. If the file is deleted, it will be recreated on Tue May 21 2025 06:40:09 GMT+0900 (Coordinated Universal Time).
# a76079f88041c0a0b3c7a988861e440d1e4e821944f34c332a571128a #

[default]
aws_access_key_id=ASIAV3OYNTTTUROYPO6
aws_secret_access_key=SUBk0tVr/yA2a1wPPPOkXGumw3wJhD0uuhz5Wpa
aws_session_token=TQ0b3pZ2luX2VjELt////////wEaDmFwLW5vcnRoZWZdC0dkYwRAIgaVmw1THI

region=ap-northeast-1
```

8/1 All

保存して終了 完成！

```
aws ecr get-login-password --region ap-northeast-1 | docker login --username AWS --password stdin 608728620263.dkr.ecr.ap-northeast-1.amazonaws.com
WARNING! Your password will be stored unencrypted in /home/ec2-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store
```

Login Successful

fargate トラブルシューティング

2023年11月21日 17:47

①docker run をしてみる

②アプリの場合はcloud9からコンテナに入って確認する

https://www.google.com/search?q=aws+ecs+fargate+execute+command&rlz=1C1TKQJ_jaJP1047JP1055&oq=aws+ecs+fargate+exec&gs_lcrp=EgZjaHJvbWUqCQgBEAAYExiABDIGCAAQRRg5MgkIARAAGBMYgAQyCggCEAAYDRgTGB4yDAgDEAAYCBgNGBMYHjIMCAQQABglGA0YExgeMg4IBRAAGAgYChgNGBMYHtIBCjEyODExajBqMTWoAgCwAgA&sourceid=chrome&ie=UTF-8

https://www.google.com/search?q=aws+ecs+update-service+enable-execute-command&rlz=1C1TKQJ_jaJP1047JP1055&oq=aws+ecs+update-service+&gs_lcrp=EgZjaHJvbWUqBwgCEAAYgAQyBggAEEUYOTIHCAEQABiABDIHCAIQABiABDIHCAMQABiABDIHCAQQABiABDIHCAUQABiABDIGCAYQABgeMgYIBxAAGB4yBggIEAAYHjIGCAkQABge0gEJNzQyMGowajE1qAIAAsAIA&sourceid=chrome&ie=UTF-8