令和5年度文部科学省委託
「専門職業人材の最新技能アップデートのための専修学校リカレント教育推進」事業

コンテナ技術基礎教材資料

情報技術者の技能アップデートのためのリカレント教育推進事業

令和5年度文部科学省委託
「専門職業人材の最新技能アップデートのための専修学校リカレント教育推進」事業

コンテナ技術基礎教材資料

情報技術者の技能アップデートのためのリカレント教育推進事業

目 次

コンテナ技術基礎教材資料

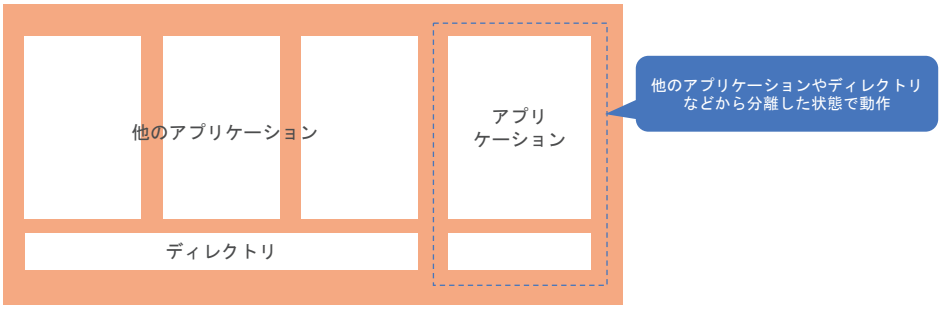
コンテナ技術の教育プログラム開発
2.1 コンテナとは

2.1 コンテナとは

2.1.1 コンテナ型仮想化とは

コンテナ型仮想化：「コンテナ」というアプリケーションと実行環境をまとめて隔離するしくみを利用し、OS 単位ではなくアプリケーション単位で仮想化する技術

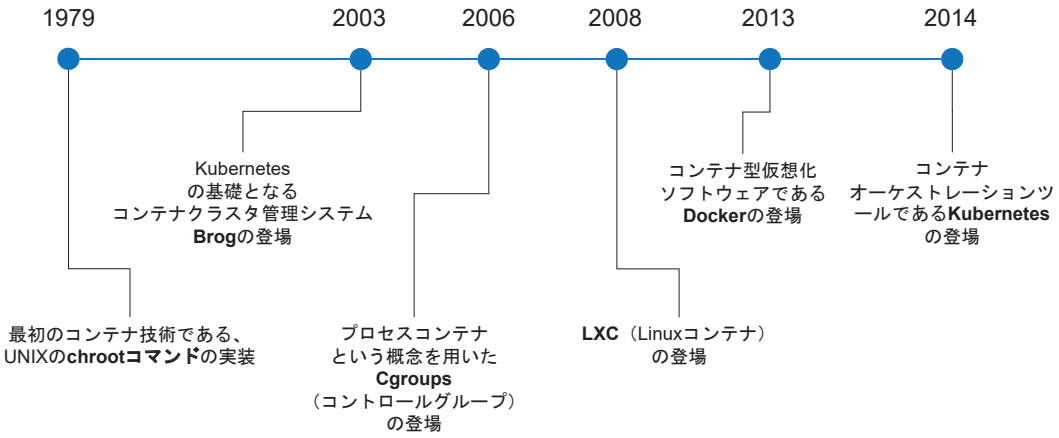
コンテナの基本的な原理は以下の通りです。



2.1 コンテナとは

2.1.2 コンテナ技術の歴史

コンテナ技術の歴史は以下の通りです。



3

2.1 コンテナとは

2.1.3 コンテナ技術のメリット

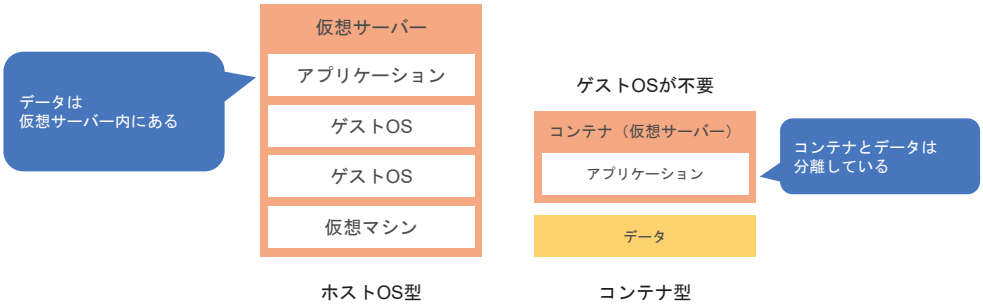
ホストOS型とコンテナ型を比較してみてみましょう。

4

2.1 コンテナとは

2.1.4 コンテナ技術のデータ管理

コンテナ技術では、可能な限りデータをコンテナに含めないのが基本です。
そのため、アプリケーション用のコンテナを破棄してもデータが失われることはなく、
データを新しいコンテナで使用することもできます。



5

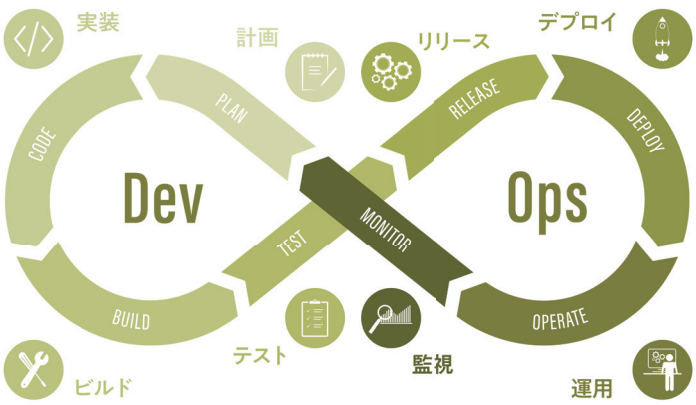
コンテナ技術の教育プログラム開発
2.2 DevOpsとは

1

2.2 DevOpsとは

2.2.1 DevOpsとは

DevOps：開発者と運用者が協力してサービスを提供する手法のこと

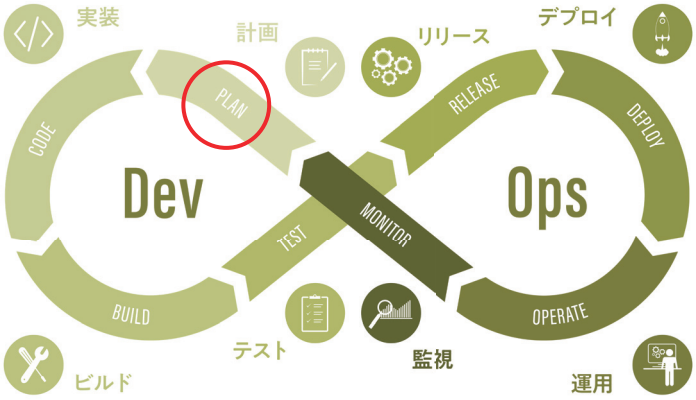


2

2.2 DevOpsとは

2.2.2 DevOps：Plan（計画）

Planはソフトウェア開発と運用の全体的な戦略や方針を策定するプロセス

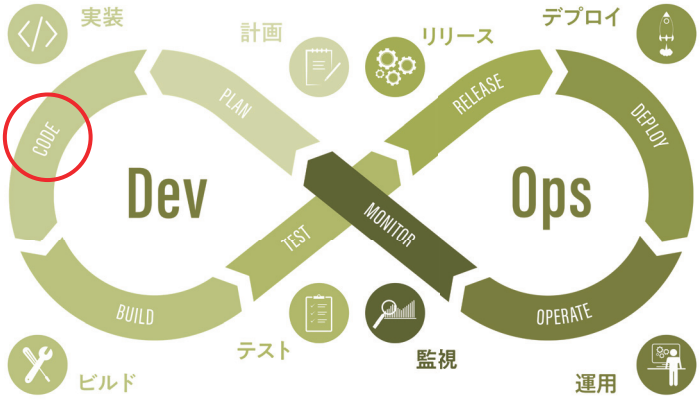


3

2.2 DevOpsとは

2.2.3 DevOps：Code（実装）

Codeはプログラミングによってアプリケーションやシステムの機能を実装するプロセス

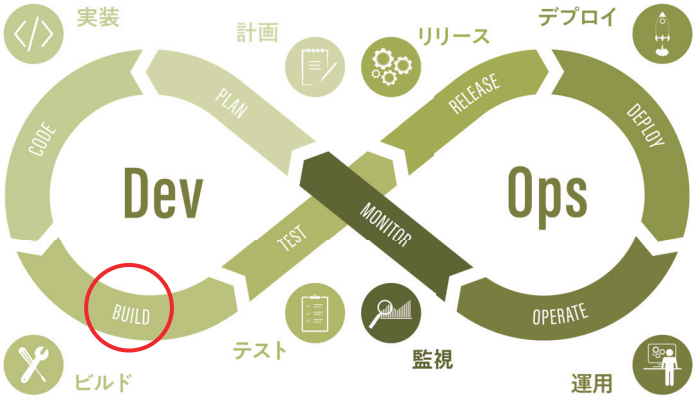


4

2.2 DevOpsとは

2.2.4 DevOps：Build（ビルド）

Buildはソースコードから実行可能なアプリケーションやコンポーネントを生成するプロセス

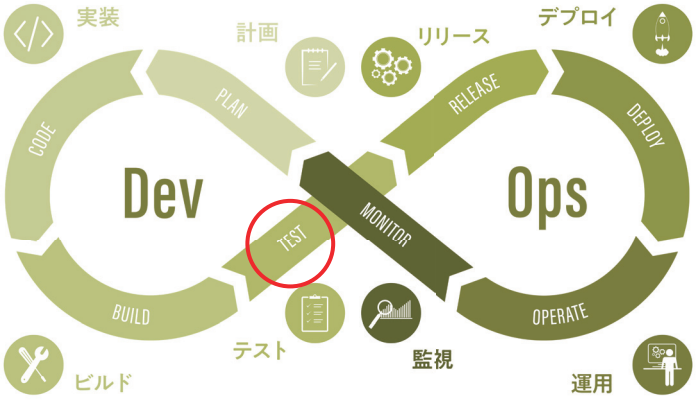


5

2.2 DevOpsとは

2.2.5 DevOps：Test（テスト）

Testはソフトウェアの品質を確認し、問題を発見して修正するためのプロセス

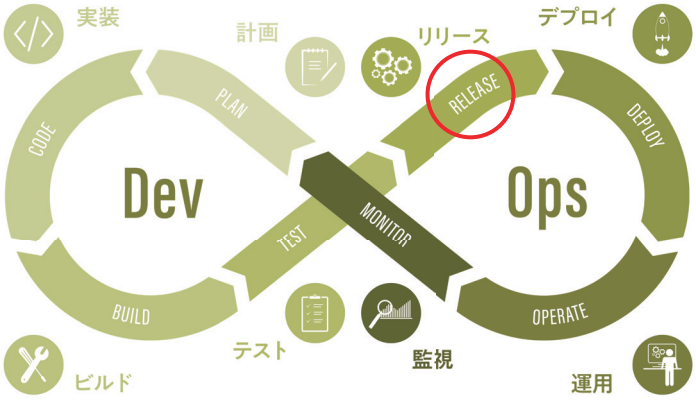


6

2.2 DevOpsとは

2.2.6 DevOps：Release（リリース）

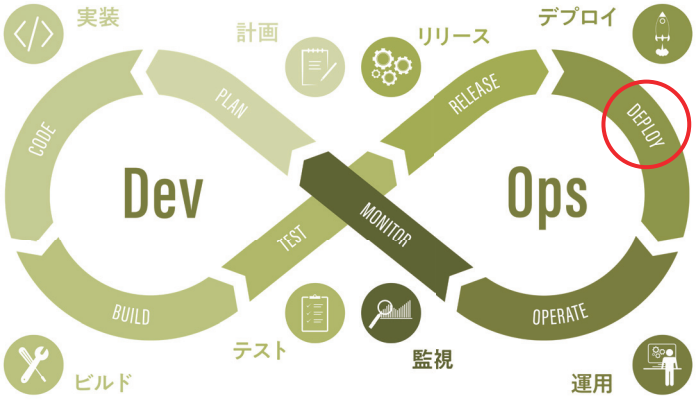
Releaseはソフトウェアなどの新しいバージョンや変更点をユーザーに提供するプロセス



2.2 DevOpsとは

2.2.7 DevOps：Deploy（デプロイ）

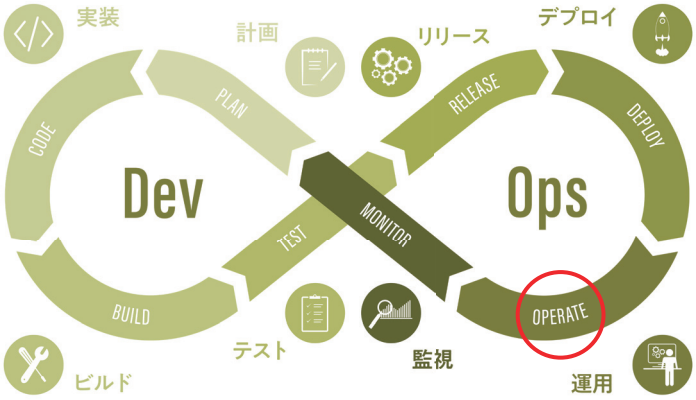
Deployはソフトウェアなどを実行環境に配置し、利用可能な状態にするプロセス



2.2 DevOpsとは

2.2.8 DevOps：Operate（運用）

Operateはソフトウェアなどが運用環境で実行され正常に機能し続けるためのプロセス

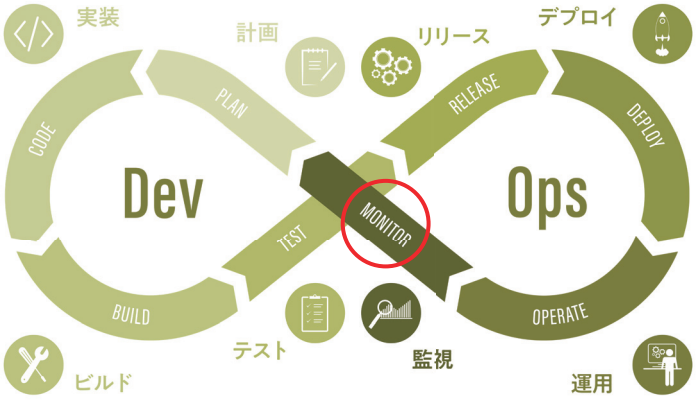


9

2.2 DevOpsとは

2.2.9 DevOps：Monitor（監視）

Monitorはソフトウェアなどのパフォーマンスを監視し問題や異常を検出するプロセス

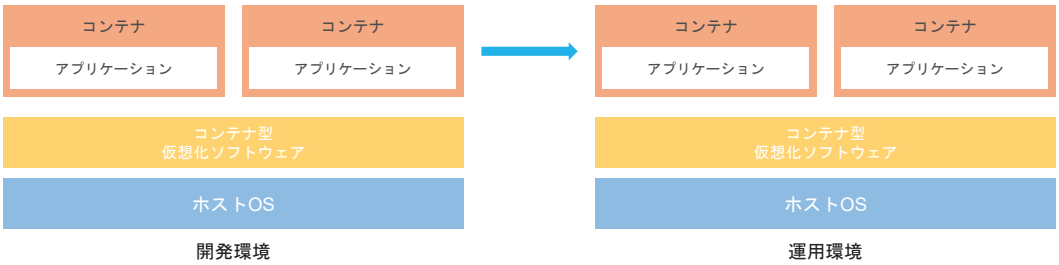


10

2.2 DevOpsとは

2.2.10 DevOpsとコンテナ技術

コンテナ技術はアプリケーションの実行に必要なプログラムやライブラリを
1つのパッケージにまとめるので、あるコンテナをそのまま他のサーバーに移しても動作します



11

コンテナ技術の教育プログラム開発
2.3 コンテナ技術とそれ以外の技術

1

2.3 コンテナ技術とそれ以外の技術

2.3.1 Docker以外のコンテナ技術

コンテナといえばDockerといえるほどDockerは広く普及しています。
しかし、コンテナ型の仮想化を実現できるDocker 以外の技術もあります。

概要	プラットフォーム
LXC（Linux Containers）	Linuxカーネルが持つコンテナ機能を利用したコンテナ型仮想化ソフトウェア。1つのLinuxカーネル上で複数のLinuxを動作させることができる
libvirt	Red Hat社を中心としたオープンソースプロジェクトで、仮想マシンの管理を提供するAPI
libcontainer	Dockerを構成するLinuxカーネルの仮想化ライブラリ
systemd-nspawn	chrootコマンドの強化版。ディレクトリ構造のコンテナ化だけでなく、プロセスツリーやさまざまなホスト・ドメイン名も仮想化することができる
cgroups	Linuxカーネルに含まれる機能で、プロセス管理による利用制限とアクセス権制御を行う
CRIU	作成したチェックポイントまでリストアするための機能を提供する

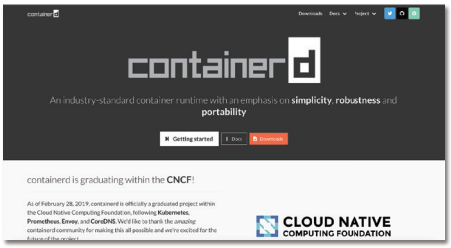
2

2.3 コンテナ技術とそれ以外の技術

2.3.2 Docker以外のコンテナランタイム

コンテナランタイム：コンテナの実行や管理を行うソフトウェアのこと

Docker以外のコンテナランタイムを紹介します。



containerd

https://containerd.io/

もともとDockerの一部であった業界標準のコンテナランタイムであり、Dockerも内部ではcontainerdを使用

cri-o

https://cri-o.io/

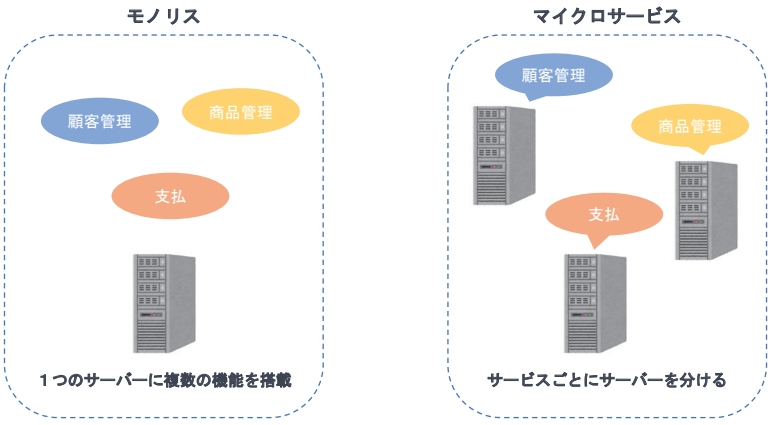
軽量なコンテナランタイムであり、コンテナを管理するためのツールである「Kubernetes」に特化

3

2.3 コンテナ技術とそれ以外の技術

2.3.3 マイクロサービスとモノリス

マイクロサービス：依存関係のない複数の小さいサービスを組み合わせて大きなサービスとして提供する設計手法

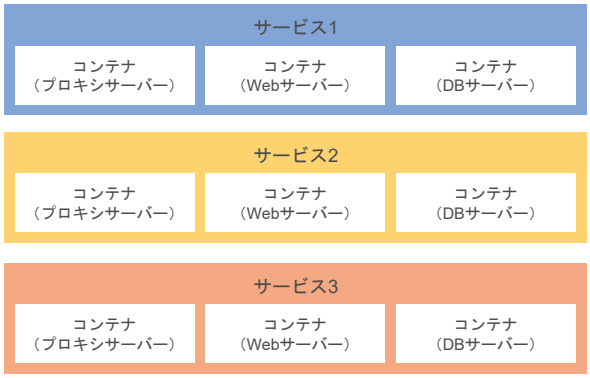


4

2.3 コンテナ技術とそれ以外の技術

2.3.4 コンテナと相性のいいマイクロサービス

コンテナ技術は、1つのアプリケーションごとに1つのコンテナを作成し、
コンテナを連携させて1つの大きなサービスを作ることができます。
そのため、マイクロサービスとの相性は非常にいいといえます。

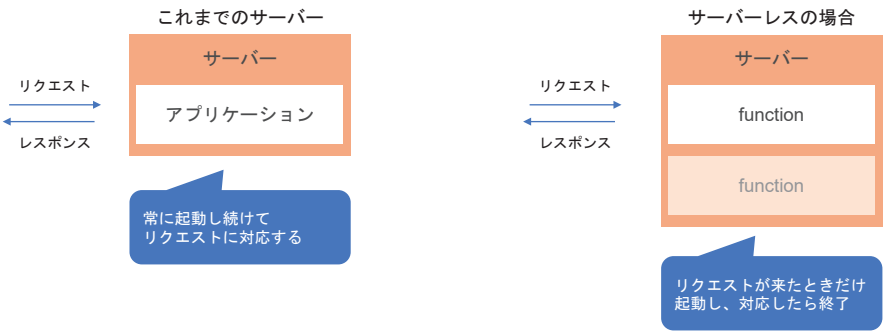


5

2.3 コンテナ技術とそれ以外の技術

2.3.5 サーバーレス

サーバーレス(Serverless) ：主にクラウドで提供されており、サーバーが存在しない環境を指す
「server」（仕える人）を「less」（欠く）という意味



6

2.3 コンテナ技術とそれ以外の技術

2.3.6 AWS Lambda

サーバーレスを提供するサービスでは、AWSのAWS Lambda が有名です。

AWS Lambda
https://aws.amazon.com/jp/lambda/
Amazon Web Services（AWS）が提供するサーバーレス
コンピューティングサービスです。

＜特徴＞

・従量制課金モデルのためコストを最適化できる
・多くのプログラミング言語をサポート
・高いセキュリティ

2.3 コンテナ技術とそれ以外の技術

2.3.7 サーバーレスのメリットとデメリット

サーバーレスのメリットとデメリットは以下の通りです。

＜メリット＞

•使っていない時間帯のコストが発生しない
•サーバーの管理が不要
•開発リソースの省略が可能

＜デメリット＞

•自由度や可搬性が低い
•リソースの制約があり長時間処理に向かない
•標準化がされていない

サーバーレスは比較的短時間の処理を実行するのに使われています。
開発したいサービスの特徴にあわせてどの技術を使用すべきかを十分に検討しましょう。

コンテナ技術の教育プログラム開発
2.4 Dockerとは

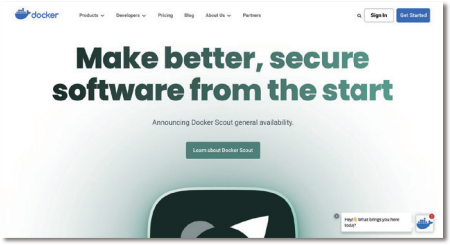
1

2.4 Dockerとは

2.4.1 Docker とは

Docker：コンテナの実行やコンテナイメージの作成・配布を行うためのプラットフォーム
※コンテナイメージ：コンテナを実行するためのテンプレート

Docker はその使いやすさから、
デファクト・スタンダードとして広い支持を集めています。
コンテナ技術といえばDockerといえるツールです。



Docker
https://www.docker.com/

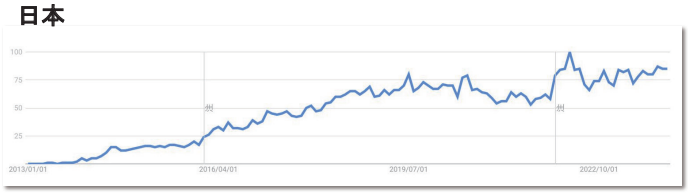
2

2.4 Dockerとは

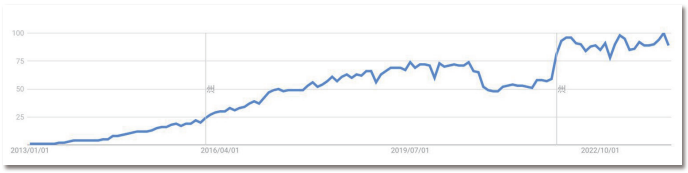
2.4.2 Dockerの注目度

Google Trendsで「Docker」をキーワードとして検索した結果です。右肩上がりに注目度が増しています。

日本



全ての国



3

2.4 Dockerとは

2.4.3 Dockerの歴史

2013年03月：Python Conference のライトニングトークでSolomon Hykes氏によって紹介

2014年06月：Docker エンジンのバージョン1.0 が一般に利用可能

2014年12月：Docker のダウンロード回数が1 億回を突破

2016年06月：コンテナオーケストレーション機能であるSwarmモードが追加

2017年10月：コンテナオーケストレーションツールのKubernetesがDocker Enterpriseに統合

2018年04月：Docker Enterprise 2.0 がアナウンス

2019年04月：Docker Enterprise 3.0 がアナウンス

4

2.4 Dockerとは

2.4.4 コンテナ技術の発展を支えるOCI

Open Container Initiative（OCI）は、
コンテナ技術の標準化とオープンソースのコンテナランタイムおよび
イメージフォーマットの仕様策定を目的とするプロジェクトです。
以下の2つの使用標準を定めています。

・**OCI Runtime Specification**
コンテナの実行時に必要な機能を規定する仕様

・**OCI Image Format Specification**
コンテナイメージの定義とパッケージングに関する仕様

OCIのメンバー

5

2.4 Dockerとは

2.4.5 Dockerのコンセプト

Docker は次のようなコンセプトの上に築かれています。

柔軟性	疎結合
プログラミング言語などが制約されない	システムを独立したコンポーネントへ分解できる
軽量	**スケーラブル**
効率的にリソースを活用する	需要に応じてリソースを増減できる
ポータブル	**セキュア**
異なる実行環境への移行が容易	コンテナ同士を分離できる

6

コンテナ技術の教育プログラム開発
2.5 Dockerコンテナ

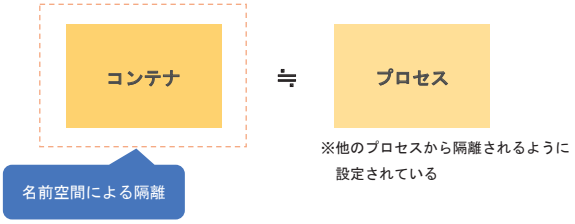
1

2.5 Dockerコンテナ

2.5.1 コンテナ とは

コンテナ：OS 上で実行されているプロセスのこと

※プロセスはプログラムの実行単位のことであり、「実行中のプログラム」ともいえます。



コンテナは他のプロセスから隔離されるよう設定

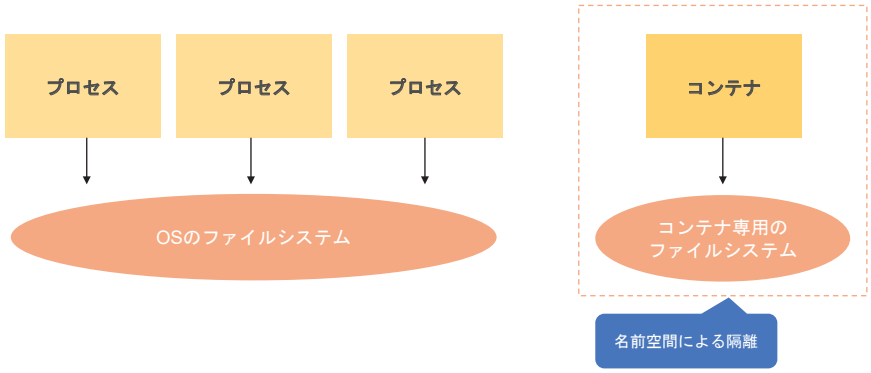
→外部から影響を受けない環境でプロセスを実行可能

2

2.5 Dockerコンテナ

2.5.2 ファイルシステムの隔離

名前空間のしくみによって実現されていることのうち、
とくに重要なことは「ファイルシステムの隔離」です。



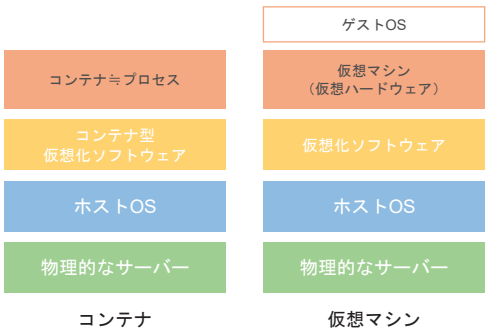
3

2.5 Dockerコンテナ

2.5.3 コンテナと仮想マシンの違い

コンテナと仮想マシンは、以下の点で決定的な違いがあります。

＜コンテナ＞
・他のコンテナやプロセスとOSを
共有する
・コンテナは瞬時に起動できる
・共有カーネルを利用するため、
リソース消費が少なく、多数の
コンテナをホストできる



＜仮想マシン＞
・他の仮想マシンや物理マシンとOS
を共有せず独自のゲストOSを持つ
・起動には時間がかかる
・独自のゲストOSを持つため、リソ
ース消費が大きく、ハードウェア
資源をより多く必要とする

→アプリケーションの要件に応じ、どちらかを選択する必要があります。

4

2.5 Dockerコンテナ

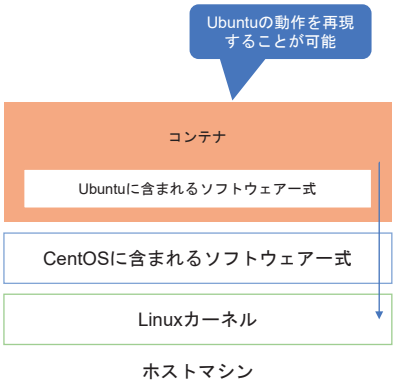
2.5.4 なぜコンテナにOS を載せることができるのか

コンテナには、CentOS やUbuntu などのOS を載せることが可能

CentOS やUbuntu などのOS は、より正確には
ディストリビューションと呼ばれます。

・Ubuntuに含まれるソフトウェアー式をコンテナが
備えている場合、そのコンテナ内で実行されるプ
ロセスをあたかもUbuntu 上で実行されているよう
に見せることができます。

・それにより、仮想マシンのようなゲストOS を使
うことなく複数のOS、つまりディストリビューシ
ョンを共存させることが可能です。



コンテナはホストマシンのOS
（カーネル）を呼び出して動
作している

5

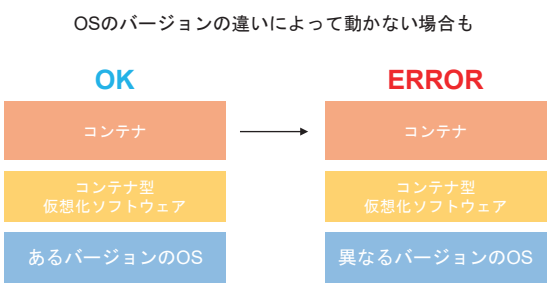
2.5 Dockerコンテナ

2.5.5 コンテナがあれば仮想マシンは不要か

コンテナでは、OSのバージョンの違いによって動かない場合もありますが、
仮想マシンではOSも含めて同じ環境を再現できます。

コンテナ
OS を共有しているため、OS のバージョンの違
い
や互換性の影響を受ける

仮想マシン
それぞれにOS をインストール

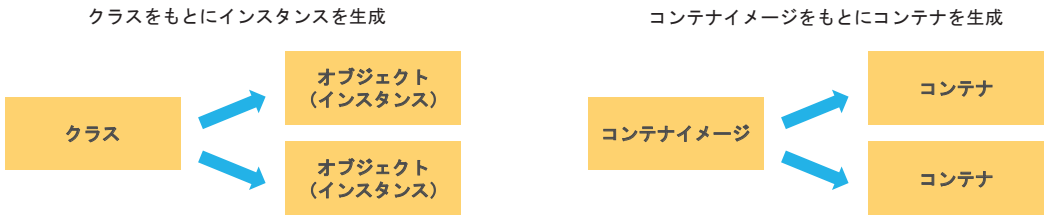


6

2.5 Dockerコンテナ

2.5.6 コンテナイメージ

コンテナイメージ：コンテナを実行するためのテンプレート

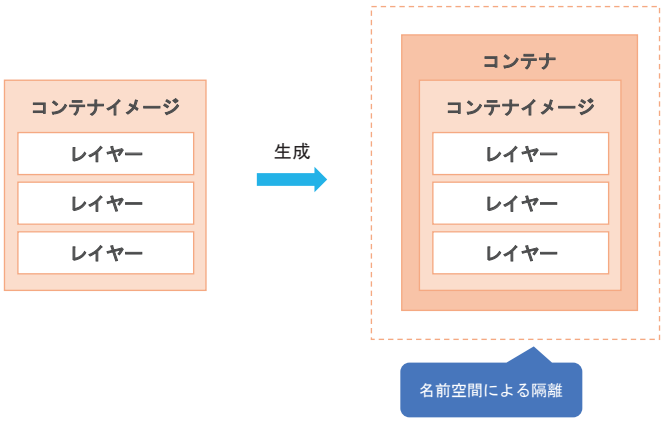


7

2.5 Dockerコンテナ

2.5.7 コンテナの生成

コンテナはコンテナイメージから生成されます。

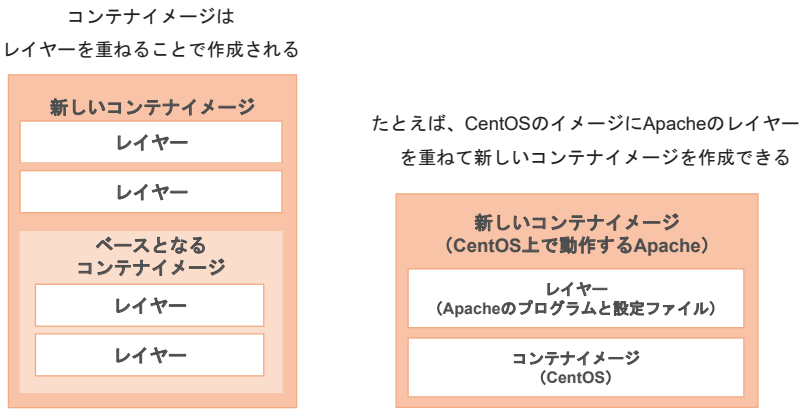


8

2.5 Dockerコンテナ

2.5.8 コンテナイメージの作成

コンテナイメージは、ベースとなるコンテナイメージのファイルシステムに
新たなレイヤーを重ねることによって作成されます。

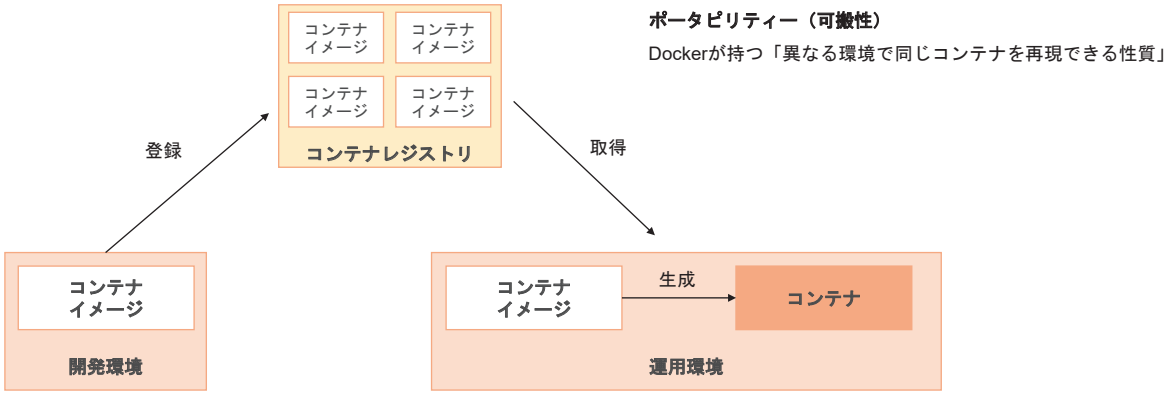


9

2.5 Dockerコンテナ

2.5.9 コンテナイメージの配布

コンテナイメージはデータなので配布することができます。



ポータビリティー（可搬性）
Dockerが持つ「異なる環境で同じコンテナを再現できる性質」

10

2.5 Dockerコンテナ

2.5.10 コンテナのライフサイクル

コンテナのライフサイクル：コンテナが作成されてから削除されるまでの「状態の流れ」

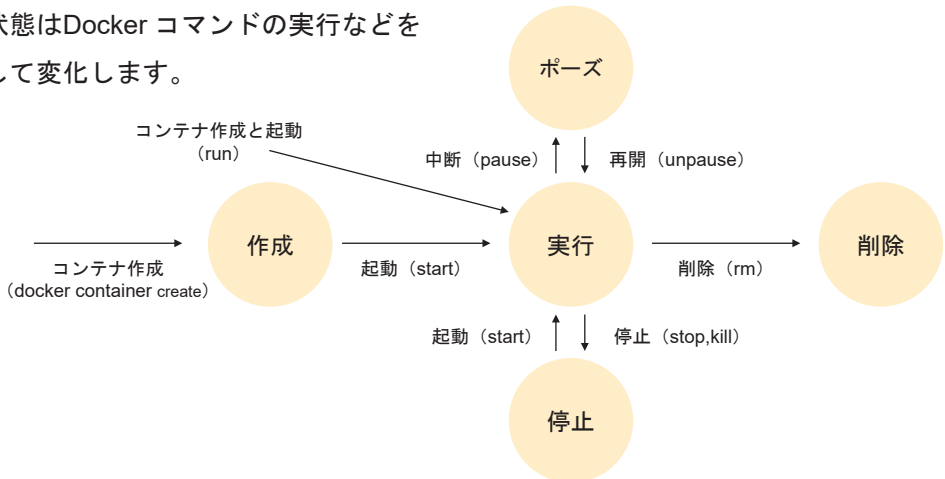
状態	説明
作成	コンテナレイヤーが作成された状態
実行	コンテナ内でプロセスが実行している状態
停止	コンテナ内でプロセスが終了している状態
ポーズ	コンテナ内でプロセスが一時停止している状態
削除	コンテナレイヤーが削除された状態

11

2.5 Dockerコンテナ

2.5.11 コンテナの状態変化

コンテナの状態はDocker コマンドの実行などを
トリガーにして変化します。



12

コンテナ技術の教育プログラム開発
2.6 Dockerのアーキテクチャ

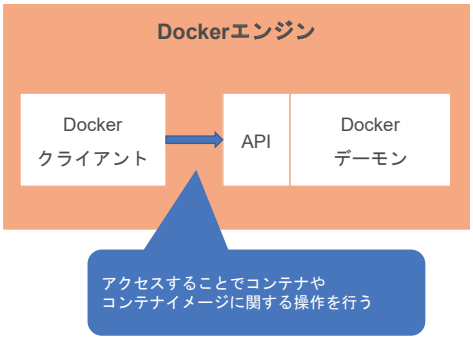
1

2.6 Dockerのアーキテクチャ

2.6.1 Docker エンジンとは

Docker エンジン：コンテナやコンテナイメージを管理するためのアプリケーション

Docker エンジンはクライアント・サーバー型のアプリケーションであり、クライアントであるDocker クライアントからサーバーであるDocker デーモンのAPI へアクセスすることによって、コンテナやコンテナイメージに関するさまざまな操作を行えます。



2

2.6 Dockerコンテナ

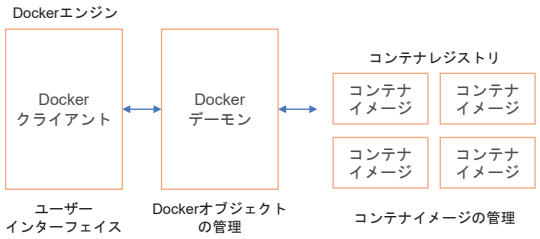
2.6.2 Dockerエンジンのコンポーネント

Docker エンジンはDocker クライアント、Docker デーモン、コンテナレジストリの3 つから構成されています。

・Dockerクライアント
Docker デーモンのUIとなるコンポーネント

・Dockerデーモン
Docker クライアントからの要求に応じDockerオブジェクトを管理するコンポーネント

・コンテナレジストリ
コンテナイメージを保管するためのコンポーネント



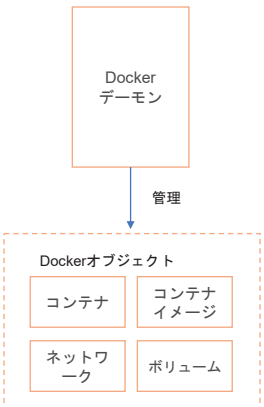
3

2.6 Dockerコンテナ

2.6.3 Docker オブジェクトとは

Docker オブジェクト：Docker デーモンの管理の対象となるもの

Docker オプジェクトにはさまざまなものが含まれています。
・コンテナ
・コンテナイメージ
・ネットワーク
・ボリューム



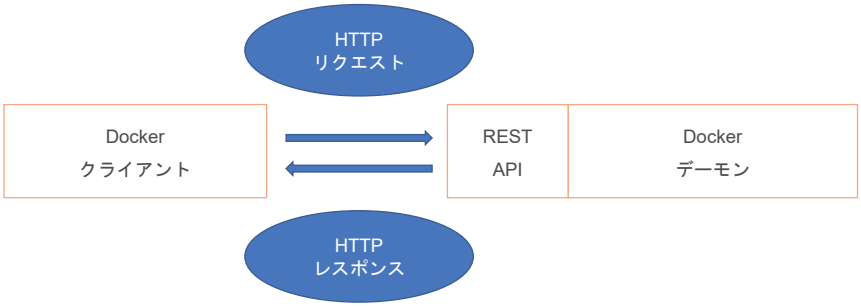
4

2.6 Dockerコンテナ

2.6.4 Docker デーモンとの通信

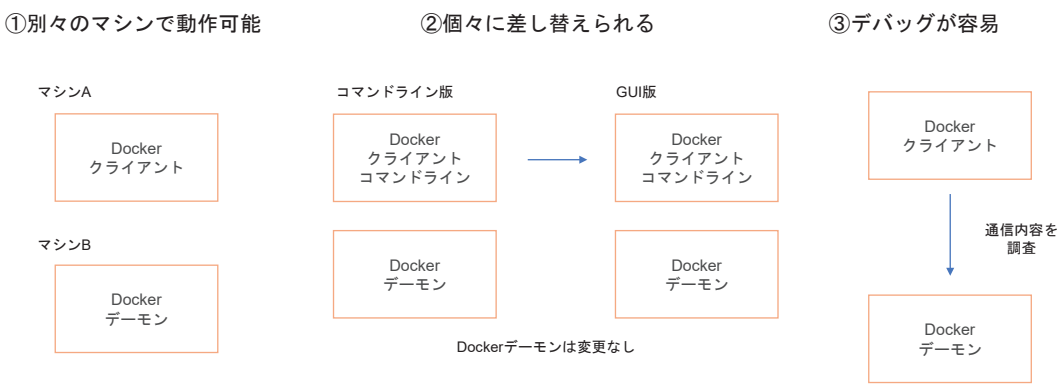
Docker デーモンはDocker クライアントからの要求を受け入れるためのAPIを備えています。

Dockerデーモンが備えるAPI ではREST と呼ばれるHTTP をベースとした通信プロトコルが使用されています。
HTTPリクエストの送信とHTTPレスポンスの受信ができるプログラムなら、Docker デーモンと通信可能です。



2.6 Dockerコンテナ

2.6.5 複数コンポーネントに分かれていることのメリット



コンテナ技術の教育プログラム開発
2.7 Dockerを使う上での基本的な考え方

1

2.7 Dockerを使う上での基本的な考え方

2.7.1 Dockerの基本

基本は1コンテナに1プロセス

Docker ではプロセスごとにコンテナを分けることが推奨されています

プロセスごとに3つのコンテナに分ける

コンテナは本質的にプロセスと同じであるため、
アプリケーションを同じコンテナに含むべきかどうか判断する場合には、
同じプロセスに含んだ場合に自然かどうかを基準にできます。

2

2.7 Dockerを使う上での基本的な考え方

2.7.2 プロセスごとにコンテナを分けるメリット

ボトルネックとなるコンテナを複数起動し処理能力の低下を改善

Docker ではプロセスの実行によるオーバーヘッドがほぼゼロのため性能の劣化はほぼなく、
ボトルネックとなるコンテナを複数起動することで負荷分散が可能になります。

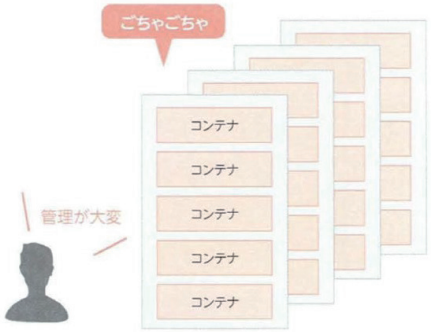
3

2.7 Dockerを使う上での基本的な考え方

2.7.3 プロセスごとにコンテナを分けるデメリット

コンテナが増えるほど管理が複雑に

この問題を解決するためのアプローチの1つがコンテナオーケストレーション



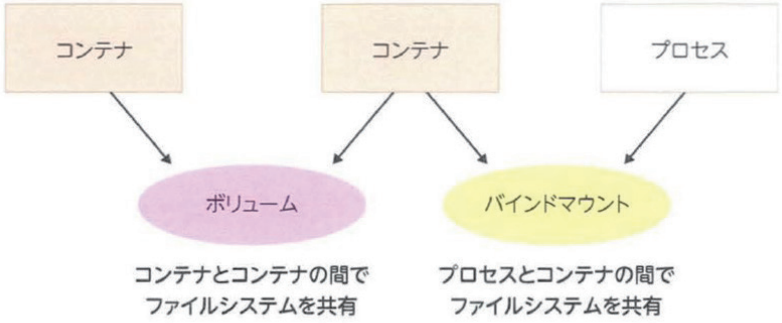
4

2.7 Dockerを使う上での基本的な考え方

2.7.4 ボリュームとは

ボリューム：コンテナによって読み書きされるデータを永続化するためのしくみ

ホストOS のファイルシステムをコンテナにマウントするバインドマウントという機能もあります。



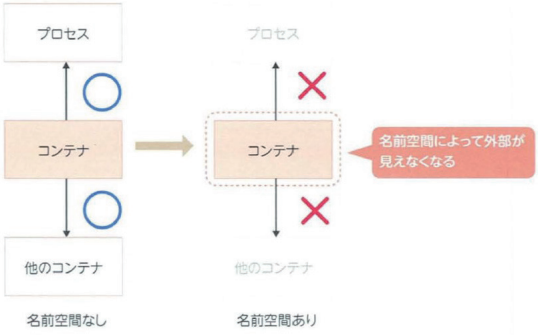
5

2.7 Dockerを使う上での基本的な考え方

2.7.5 名前空間とは

名前空間：内部のプロセスに対して外部のプロセスから隔離されているように見せるしくみ

名前空間のしくみを一言で表すと「見えないなら、ないのと同然」ということです。

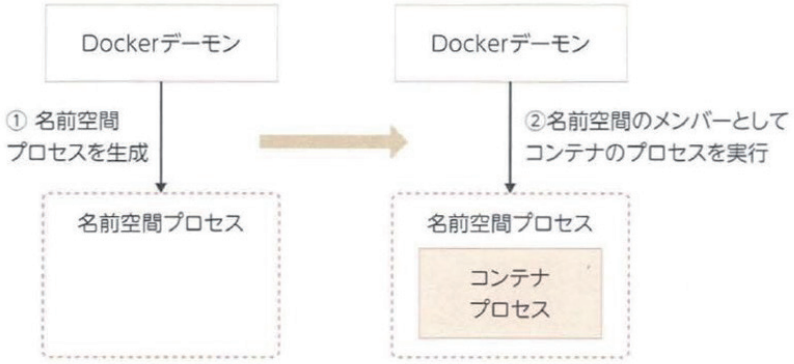


6

2.7 Dockerを使う上での基本的な考え方

2.7.6 名前空間とコンテナ

Docker では名前空間のしくみを使ってコンテナを他のコンテナから隔離



7

2.7 Dockerを使う上での基本的な考え方

2.7.7 名前空間によって隔離されるリソース

名前空間はさまざまなリソースを隔離することができる

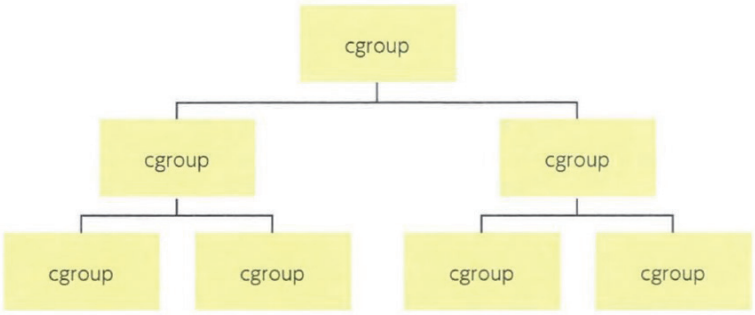
名前空間	隔離することができるリソース
PID名前空間	プロセスID
ネットワーク名前空間	ネットワークインターフェイス、ポート番号
IPC名前空間	プロセス間通信
マウント名前空間	ファイルシステム
UTS名前空間	ホスト名

8

2.7 Dockerを使う上での基本的な考え方

2.7.8 control groups とは

control groups ：ハードウェアリソースの使用量を制限するしくみ



9

2.7 Dockerを使う上での基本的な考え方

2.7.9 差分管理

差分管理：ベースとの相違点を記録していくことによって変更を管理する方法のこと

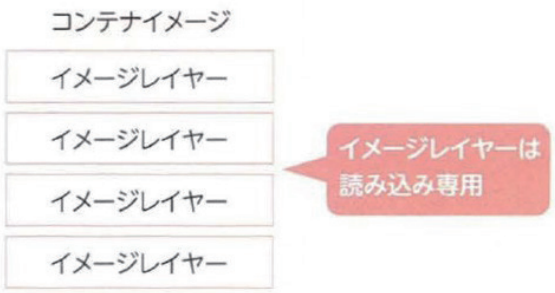
コンテナイメージに含まれるファイルシステムに、差分管理のしくみを使用しています。

10

2.7 Dockerを使う上での基本的な考え方

2.7.10 イメージレイヤーの特徴

イメージレイヤー：コンテナイメージに含まれるレイヤー

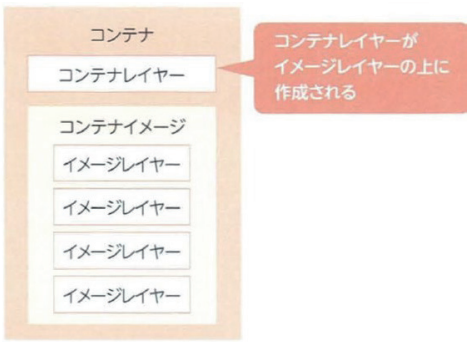


11

2.7 Dockerを使う上での基本的な考え方

2.7.11 コンテナレイヤーの特徴

コンテナレイヤー：イメージレイヤーの上に作成されたコンテナ内のプロセスによる
ファイルの追加や変更を記録するためのレイヤー

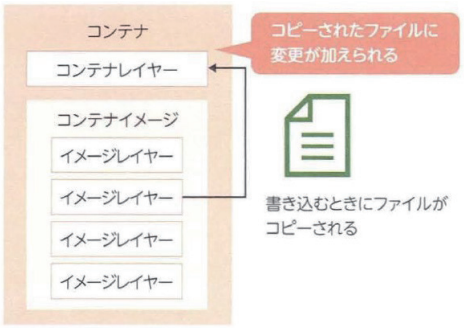


12

2.7 Dockerを使う上での基本的な考え方

2.7.12 差分管理のメリット

差分管理のおかげでコンテナを起動するたびにコンテナイメージに含まれるファイルを
コンテナレイヤーにすべてコピーする必要がなく、高速なデプロイが実現



13

2.7 Dockerを使う上での基本的な考え方

2.7.13 Swarm mode

Swarm mode ：Docker エンジンに設けられたクラスタ管理の機能のこと

クラスタ管理とは、複数のマシンなどをまとめて管理することです。

14

2.7 Dockerを使う上での基本的な考え方

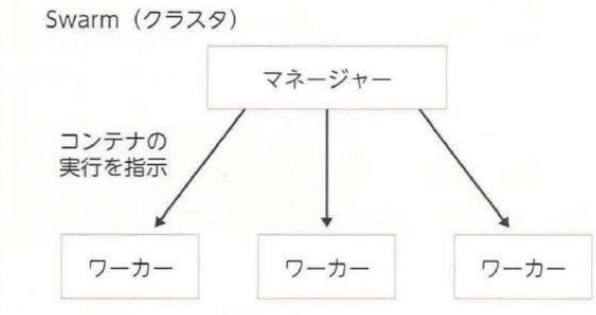
2.7.14 マネージャーとワーカー

Swarm 内のノードには3つの役割から1つが与えられる

マネージャー
Swarm全体を管理し、
ワーカーにタスクの実行を指示します。

ワーカー
マネージャーからの指示に従って
コンテナを実行する役割を担います。

マネージャー兼ワーカー
マネージャーとワーカー両方の役割を
兼ねます。

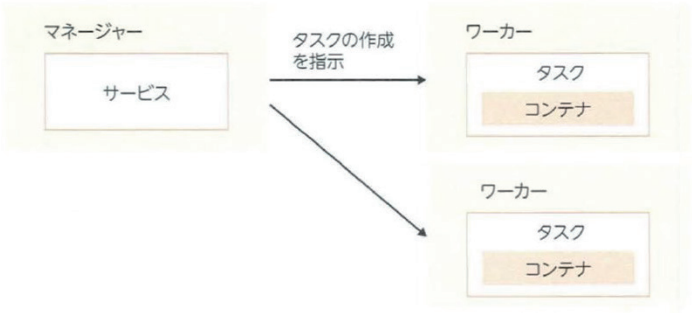


15

2.7 Dockerを使う上での基本的な考え方

2.7.15 サービスとタスク

ワーカーで実行されるコンテナはタスクと呼ばれ、サービスを管理するための最小単位となります。



16

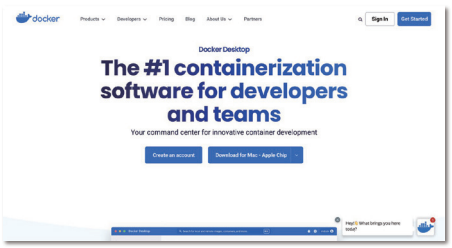
コンテナ技術の教育プログラム開発
2.8 Dockerの基本的な使い方

1

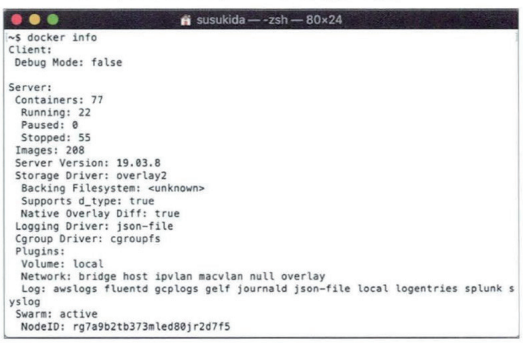
2.8 Dockerの基本的な使い方

2.8.1 主要なDockerコマンド

Docker でコンテナやコンテナイメージを操作する際はコマンドを使用します。



Docker Desktop
https://www.docker.com/products/docker-desktop/



2

2.8 Dockerの基本的な使い方

2.8.2 Docker コマンドの基本

AWS にてコンテナを扱う際にいくつか基本的なDockerのコマンドを利用します。
ここでは、最低限押さえておくべきDocker のコマンドを紹介します。

Dockerコマンド

Docker + info. + build + - &英字1文字
version ls --&英単語
image rm
container tag
. .
. .
. .

動作	コマンド（新体系）
イメージの作成	docker image build
イメージ情報の表示	docker image ls
イメージの削除	docker image rm
イメージタグの追加	docker image tag
レジストリへのイメージ保存	docker image push
レジストリからのイメージ取得	docker image pull
コンテナの実行	docker container run
ログの確認	docker container logs
実行中のコンテナに対するコマンド実行	docker container exec
コンテナの停止	docker container stop

3

2.8 Dockerの基本的な使い方

2.8.3 イメージの作成(docker image build)

アプリケーションのソースコードとDockerfile からイメージを作成する場合は、
docker image build コマンドを利用します。

次の例は、Dockerfile とアプリケーションコードがカレントディレクトリに配置されている状況下において、
「myapp」という名前のイメージを作成します。

◇イメージの作成

```
$ docker image build -t myapp:v1 .
```

ここでは、イメージのバージョン管理等を行えるようにタグ情報として「v1」を付与しています。

4

2.8 Dockerの基本的な使い方

2.8.4 イメージ情報の表示(docker image ls)

作成•取得したイメージ情報の一覧を表示するには、
docker image ls コマンドを実行します。

◇イメージ情報の表示

```
$ docker image ls
REPOSITORY   TAG        IMAGE ID       CREATED              SIZE
myapp        v1         57150e8e03d9   About a minute ago   10.5MB
```

5

2.8 Dockerの基本的な使い方

2.8.5 イメージの削除(docker image rm)

不要になったイメージは、**docker image rm** コマンドで削除可能です。

以下は、作成した「myapp」イメージを削除する例です。削除するイメージはIDで指定しています。
また、削除後にイメージ情報を確認しています。

◇イメージの削除

```
$ docker image rm 57150e8e03d9
Untagged: myapp:v1
Deleted: sha256:57150e8e03d977f29463fa7826ed4c894d5c24166d1501d21e
b7d9302d7066ed

$ docker image ls
REPOSITORY   TAG        IMAGE ID   CREATED   SIZE
```

6

2.8 Dockerの基本的な使い方

2.8.6 イメージタグの追加(docker image tag)

作成したイメージに対して**docker image tag**コマンドを使うことでタグ情報を追加できます。

次の例は、「v1 」タグが付与されている「myapp」イメージに対して、新たに「20210701 」タグを追加します。

◇イメージタグの追加

```
$ docker image tag myapp:v1 myapp:20210701

$ docker image ls
REPOSITORY   TAG        IMAGE ID       CREATED              SIZE
myapp        20210701   57150e8e03d9   About a minute ago   10.5MB
myapp        v1         57150e8e03d9   About a minute ago   10.5MB
```

7

2.8 Dockerの基本的な使い方

2.8.7 レジストリへのイメージの保存(docker image push)

作成したイメージは、**docker image push** コマンドでレジストリに保存が可能です。

次の例は、「v1 」とタグを付けた「myapp」イメージをレジストリに保存します。

◇レジストリへのイメージの保存

```
$ docker image push myapp:v1
```

実際の運用にて「docker image push」を利用する場合は、コマンド実行前にレジストリへのログインや特定のルールに従ったイメージ名の付与等が必要となります。

8

2.8 Dockerの基本的な使い方

2.8.8 レジストリからのイメージの取得(docker image pull)

レジストリに保存したイメージは、**docker image pull** コマンドで取得できます。

次の例は、レジストリに保存された「v1」とタグを付けた「myapp」イメージを取得します。

◇レジストリからのイメージの取得

```
$ docker image pull myapp:v1
```


2.8 Dockerの基本的な使い方

2.8.9 コンテナの実行(docker container run)

コンテナを実行するには、**docker container run** コマンドを実行します。

次の例では、デタッチドモード（バックグラウンドでコンテナを実行）かつ80番のポート番号をマッピングしてコンテナを実行しています。

◇コンテナの実行

```
$ docker container run -d -p 80:80 myapp:v1
101b5647431d206752564b6b987aacb77c4030096fc948106a07a5b9cbb47e83
```

実行したコンテナの起動状態は、docker container ls コマンドで確認できます。

次の例では、出力する情報を指定して実行しています。

◇コンテナの起動状況の確認

```
$ docker container ls --format  'table {{.ID}}\t{{.Image}}\t{{.
Status}}'
CONTAINER ID   IMAGE      STATUS
101b5647431d   myapp:v1   Up 8 minutes
```

2.8 Dockerの基本的な使い方

2.8.10 ログの確認(docker container logs)

コンテナ内のアプリケーションのログを確認するには、
docker container logs コマンドが便利です。

コマンド実行時には「docker container ls 」で表示された「CONTAINER_ID 」指定します。
「-f」オプションを付けることで、をログを表示し続けることもできます。

◇ログの確認

```
$ docker container logs 101b5647431d
```

11

2.8 Dockerの基本的な使い方

2.8.11 実行中のコンテナに対するコマンド実行 (docker container exec)

docker container exec コマンドを実行することで、
実行中のコンテナに対してコマンドを発行できます。

次の例は、稼働中のコンテナに対して「/bin/sh」を実行することで、コンテナ内のシェルに切り替える例です。

◇実行中のコンテナに対するコマンド実行

```
$ docker container exec -it 101b5647431d /bin/sh
/ #
```

12

2.8 Dockerの基本的な使い方

2.8.12 コンテナの停止(docker container stop)

実行中のコンテナを停止するには、**docker container stop**コマンドを実行します。

コマンド実行時には「docker container ls」で表示された「CONTAINER_ID 」を指定します。

◇コンテナの停止

```
$ docker container stop 101b5647431d

$ docker container ls --format  'table {{.ID}}\t{{.Image}}\t{{.
Status}}'
CONTAINER ID   IMAGE     STATUS
```

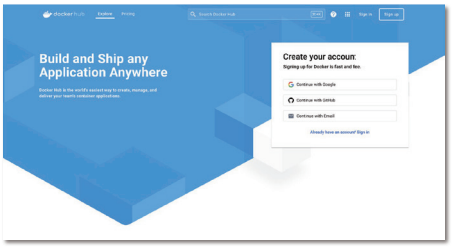
13

2.8 Dockerの基本的な使い方

2.8.13 Docker Hub

Docker Hub：世界最大のコンテナレジストリサービス

さまざまなソフトウェアのコンテナイメージが登録されており、ユーザーはDocker Hub からコンテナイメージを取得して実行することで、ソフトウェアを利用できます



Docker Hub
https://hub.docker.com/

14

2.8 Dockerの基本的な使い方

2.8.14 コンテナレジストリ

コンテナレジストリ：コンテナイメージの保管と配布の2つの役割を担うDockerのコンポーネント

コンテナレジストリの実体は、レジストリサーバー上で動作するアプリケーションです。

プッシュ
コンテナレジストリへコンテナイメージをアップロードすること

プル
コンテナレジストリからコンテナイメージをダウンロードすること

15

2.8 Dockerの基本的な使い方

2.8.15 リポジトリとタグ

リポジトリ：コンテナイメージの保管場所

コンテナレジストリが複数のリポジトリを含むのに対し、リポジトリは複数のコンテナイメージを含みます。

16

2.8 Dockerの基本的な使い方

2.8.16 Docker composeとは

Docker Compose：複数のコンテナを定義し実行する Docker アプリケーションのためのツール

Compose は YAML ファイルを使い、アプリケーションのサービスを設定します。
コマンドを１つ実行するだけで、設定内容に基づいた全てのサービスを生成・起動します。

抜粋: Docker Compose 概要
https://docs.docker.jp/compose/index.html

17

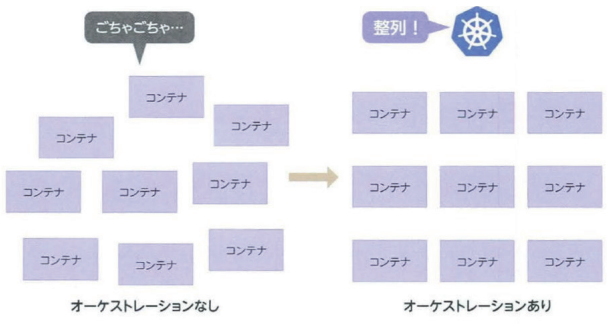
コンテナ技術の教育プログラム開発
2.9 コンテナオーケストレーションとは

1

2.9コンテナオーケストレーションとは

2.9.1 オーケストレーションとは

オーケストレーション：デプロイ、スケーリング、運用を自動化および管理する技術

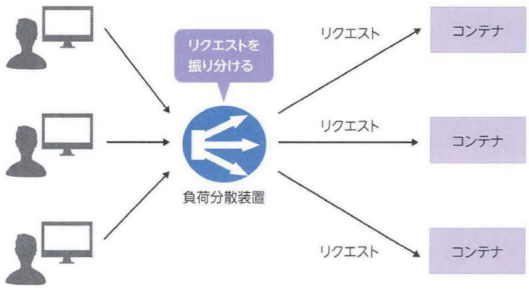


2

2.9コンテナオーケストレーションとは

2.9.2 負荷分散

負荷分散：複数のコンテナへリクエストを振り分けて負荷を分散すること

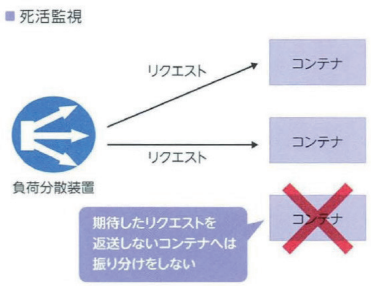


3

2.9コンテナオーケストレーションとは

2.9.3 死活監視

死活監視：システムを構成するコンテナが正常に稼働しているかとうかを監視すること

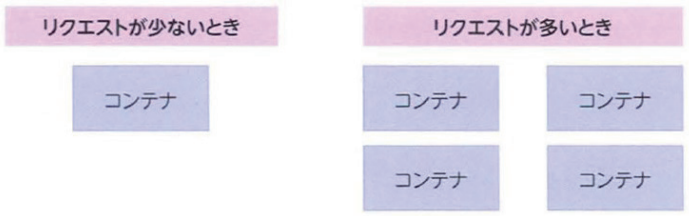


4

2.9コンテナオーケストレーションとは

2.9.4 スケーリング

スケーリング：リクエストの規模に応じてコンテナを増減させること



5

コンテナ技術の教育プログラム開発
2.10 Kubernetesとは

1

2.10 Kubernetesとは

2.10.1 Kubernetes

Kubernetes：オーケストレーション（コンテナの管理・運用を自動化する）を行うツール

Kubernetes はコンテナの管理・運用を自動化することができるので、コンテナを活用したアプリケーションの運用に要する負荷を、軽減することができます。

コンテナオーケストレーションツールはKubernetes だけではありませんが、Kubernetes はコンテナオーケストレーションツールのデファクトスタンダードとしての地位を築いています。

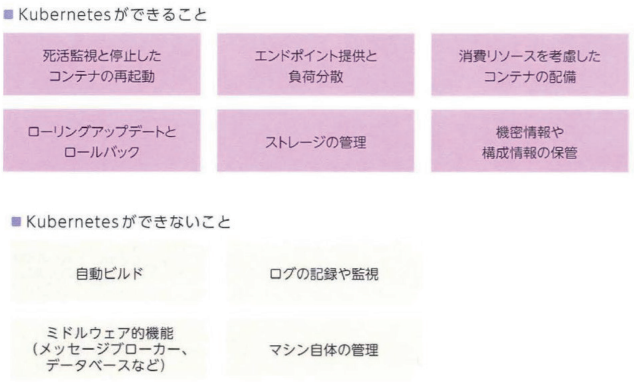
Kubernetes（https://kubernetes.io/）

2

2.10 Kubernetesとは

2.10.2 Kubernetes ができること

Kubernetes はコンテナの管理・運用を自動化することができるので、
コンテナを活用したアプリケーションの運用に要する負荷を、軽減することができます。

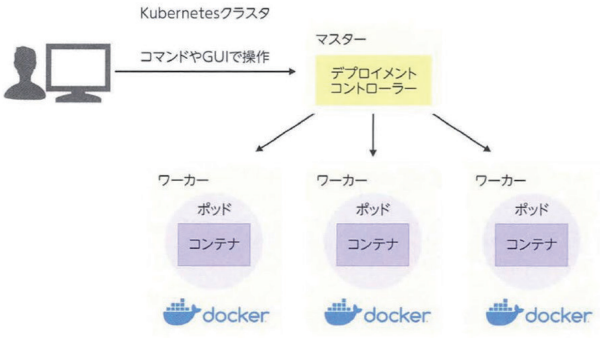


3

2.10 Kubernetesとは

2.10.3 Kubernetes クラスタとは

Kubernetesクラスタ：Kubernetesによって管理されるクラスタ

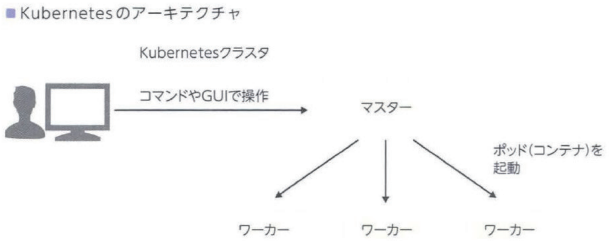


4

2.10 Kubernetesとは

2.10.4 Kubernetesのアーキテクチャ

Kubernetes にはマスターとワーカーの2 種類のノードがあります。
ユーザーは、kubectl コマンドなどを使ってマスターに命令を送り、
マスターはその命令に従って、ワーカーを操作します。
このマスターとワーカーではさまざまなコンポーネントが動作しており、
そのコンポーネントが連携し合うことで、オーケストレーションは実行されています。



5

2.10 Kubernetesとは

2.10.5 Kubernetesクラスタを操作する方法

ユーザーがKubernetes クラスタを操作する方法は以下の3つです。

kubectl コマンド
　キーボードからコマンドを入力して実行すると、Kubernetes クラスタを操作できます。

Dashboard （ダッシュボード）
　Kubernetes クラスタを操作したり、ポッドのCPU 使用率などの統計データをグラフでわかりやすく表示できます。

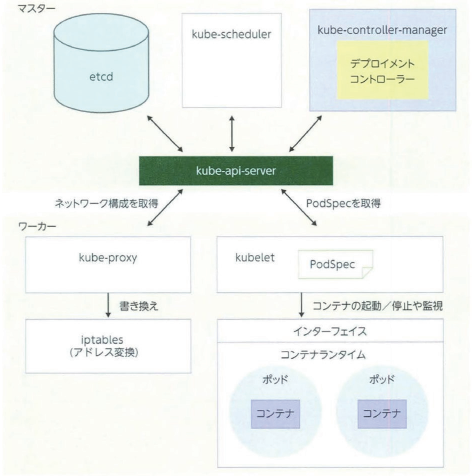
Kubernetes API
　このAPI を用いてマスターにリクエストを送信すると、Kubernetes クラスタを操作できます。

6

2.10 Kubernetesとは

2.10.6 Kubernetesのアーキテクチャ①

マスターでは以下のようなコンポーネントが実行されます。



kube-api –server
ユーザーやワーカーがKubernetes クラスタとやりとりするためのエンドポイントを提供します。

etcd
Kubernetes クラスタに関する設定などのデータをなどの統計データをグラフでわかりやすく表示です。

kube-scheduler
ポッドを起動するノードを計画するスケジューラーです。

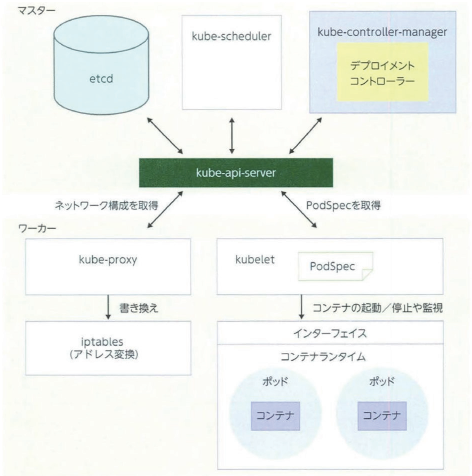
kube-controllermanager
Kubernetes クラスタを期待する状態に維持します。

7

2.10 Kubernetesとは

2.10.7 Kubernetesのアーキテクチャ②

ワーカーでは以下のようなコンポーネントが実行されます。



コンテナを起動するために必要なコンテナランタイムが実行されるとともに、kubelet とkube-proxyと呼ばれる2 つのエージェントプログラムが実行されます。

Kubelet
マスターからPodSpec を取得し、コンテナランタイムのインターフェイスへアクセスして、コンテナの起動／停止や監視を行います。

kube-proxy
マスターからネットワーク構成を取得し、ポッドとポッドが通信するときに、アクセスの中継やアドレス変換ルールの更新を行います。

8

コンテナ技術の教育プログラム開発
2.11 ポッドとデプロイメントコントローラ

1

2.11 ポッドとデプロイメントコントローラ

2.11.1 ポッド(Pod)とは

ポッド(Pod)：Kubernetes がコンテナを管理するための最小単位

ポッドは1 つ以上のコンテナから構成されます。

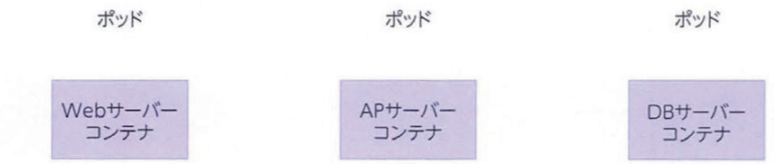
2

2.11 ポッドとデプロイメントコントローラ

2.11.2 同じポッドに含めるかどうかの判断基準

まったく無関係なコンテナ同士を同じポッドに含めることは可能ですが、推奨されません。
以下の判断基準に該当するコンテナは別々のポッドにしましょう。

＜判断基準＞
・まったく無関係なコンテナ同士
・関係はあるが同じポッド内で起動する必要がないコンテナ同士



3

2.11 ポッドとデプロイメントコントローラ

2.11.3 ポッドの必要性

1 つのコンテナ内で複数のプロセスを起動する場合は、プロセスの管理をユーザー自身で行う必要があります。
ポッド内で複数のコンテナを起動すればコンテナの管理をKubernetesに任せられ、
運用の負荷やリソース消費量が減ります。

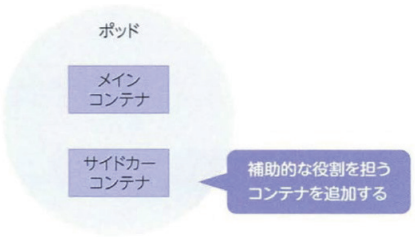
4

2.11 ポッドとデプロイメントコントローラ

2.11.4 ポッドのデザインパターン

サイドカーコンテナ：補助的な役割を担うコンテナ

1つのポッドごとに1つのコンテナとするのが一般的なポッドの使い方です。
補助的な役割を担うコンテナを同じポッドに追加すると、メインとなるコンテナの機能を拡張することができます。



＜補助的な役割＞
ログ収集やキャッシュ
プロキシ／アダプタなどのヘルパープログラム

5

2.11 ポッドとデプロイメントコントローラ

2.11.5 サイドカーコンテナの使い所

共通の機能を1つにまとめて再利用する方法はサイドカーコンテナだけではありません。
パッケージとしてプロセスに組み込むこともできますし、別々のポッドに含めてポッド間で通信することも可能です。

■ Envoy と Istio

envoy

Envoy（https://www.envoyproxy.io/）

Istio（https://istio.io/）

6

2.11 ポッドとデプロイメントコントローラ

2.11.6 デプロイメントコントローラ

レプリカセット(Replica Set)：Kubernetesでポッドの複製を管理し、指定した数だけ維持する機能

対象とするポッドの複製（レプリカ）を指定した数だけ維持する

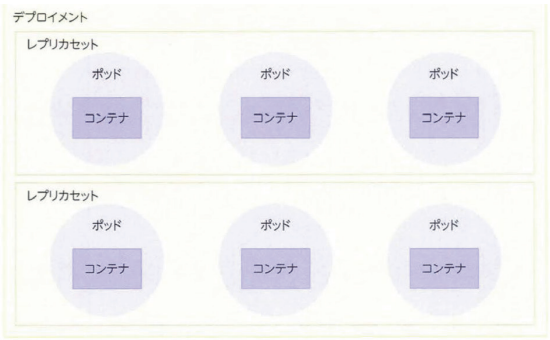
7

2.11 ポッドとデプロイメントコントローラ

2.11.7 デプロイメントとは

デプロイメント(Deployment)：レプリカセットをさらにグループにして管理する単位

レプリカセットが維持するポッド数の増減やレプリカセットの作成を行う

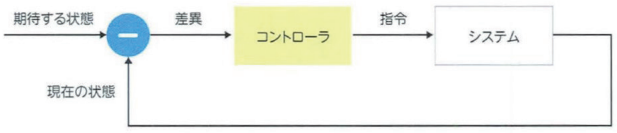


8

2.11 ポッドとデプロイメントコントローラ

2.11.8 コントローラとは

コントローラ(controller) ：現在の状態を期待する状態にするためのメカニズムのこと

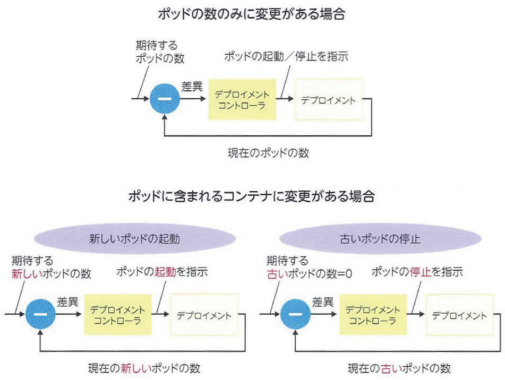


9

2.11 ポッドとデプロイメントコントローラ

2.11.9 デプロイメントコントローラとは

デプロイメントコントローラ：テプロイメントをあるべき状態に保つためのコントローラ



10

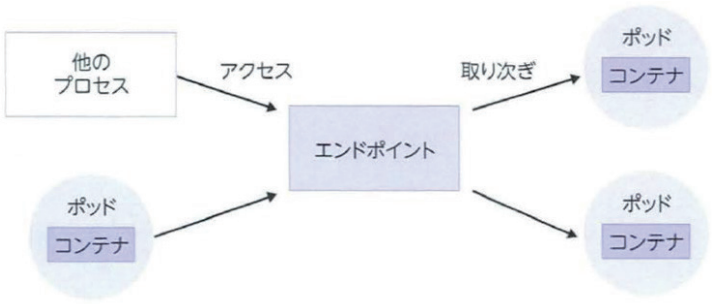
コンテナ技術の教育プログラム開発
2.12 Kubernetesを使う上での基本的な考え方

1

2.12 Kubernetesを使う上での基本的な考え方

2.12.1 サービス(Service)

サービス(Service)：コンテナへアクセスするための窓口となるエンドポイントを提供するしくみ

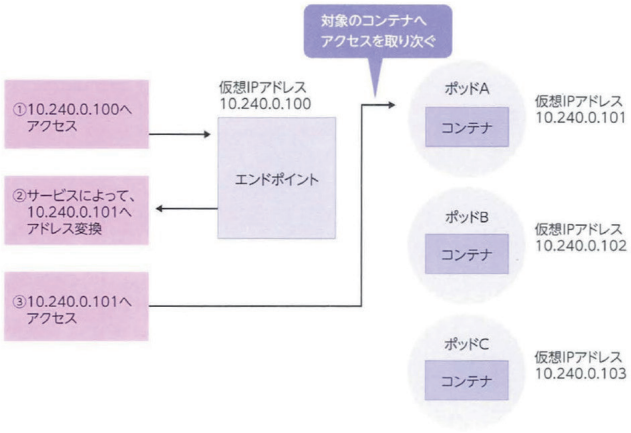


2

2.12 Kubernetesを使う上での基本的な考え方

2.12.2 サービスのしくみ

Kubernetes では、仮想IPアドレスとネットワークアドレス変換を組み合わせて、
サービスのしくみを実現しています。



3

2.12 Kubernetesを使う上での基本的な考え方

2.12.3 サービスのタイプ

エンドポイントを提供する方法の違いによって、サービスは4 種類に分類されます。

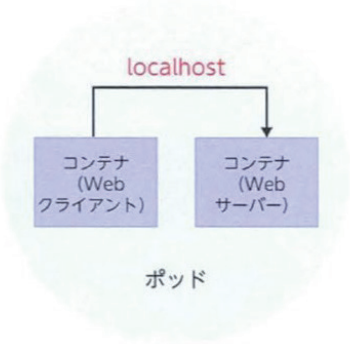
サービスのタイプ	特徴	使い分け
ClusterIP	クラスタ内部からのみアクセス可能な仮想IPアドレスが発行される	クラスタの内部からアクセスできれば十分な場合
NodePort	クラスタ内の全ノードの指定ポートへのアクセスがコンテナへ転送される。クラスタ外部からもアクセス可能	クラスタの内部からのアクセスがメインである一方でクラスタの外部から動作確認などのために時々アクセスする場合 クラウドサービスの負荷分散装置では要件を満たせないため自前で負荷分散のしくみを整える場合
LoadBalancer	クラウドサービスのロードバランサーへのアクセスがコンテナへ転送される。クラスタ外部からもアクセス可能	クラスタの外部からアクセスする必要がある場合
ExternalName	サービスへのアクセスが指定のホストへ転送される	アクセスを取り次ぐ先としてクラスタ外部のホストなどを指定する場合

4

2.12 Kubernetesを使う上での基本的な考え方

2.12.4 ポッド内の通信

同じポッドに含まれるコンテナは他のコンテナとネットワークの名前空間を共有しているため、
ポッド内のコンテナはlocalhostで通信できます。

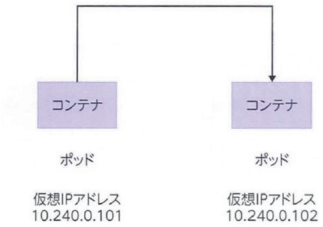


5

2.12 Kubernetesを使う上での基本的な考え方

2.12.5 ポッド間の通信

ポッドにはユニークなIP アドレスが割り当てられているため、
そのIP アドレスへアクセスすると外部からでもポッドに含まれるコンテナへアクセスできます。

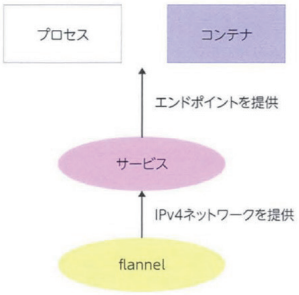


6

2.12 Kubernetesを使う上での基本的な考え方

2.12.6 flannelとサービスの違い

flannel：Kubernetes のネットワーク実装のひとつで、
クラスタを構成する複数のマシン同士をつなぐオーバーレイネットワークを構築します。



・flannel

クラスタ内の異なるマシンにあるポッド間の通信を可能にするしくみ

・サービス

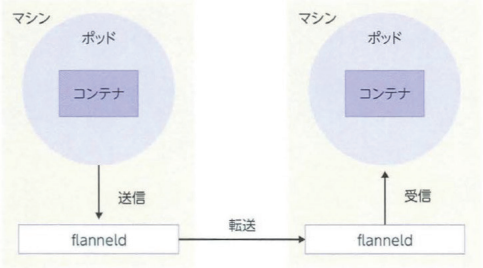
ポッドへアクセスするためのエンドポイントを提供するしくみ

7

2.12 Kubernetesを使う上での基本的な考え方

2.12.7 flanneld とは

flanneld：flannel がIPv4 ネットワークを構築するために、
Kubernetesクラスタを構成するすべてのマシン上で起動するエージェントプログラム



8

2.12 Kubernetesを使う上での基本的な考え方

2.12.8 負荷分散と死活監視

負荷分散：複数のコンテナへリクエストを振り分けて負荷を分散すること
死活監視：システムを構成するコンテナが正常に稼働しているかどうかを監視すること

9

2.12 Kubernetesを使う上での基本的な考え方

2.12.9 可用性とは

可用性：ユーザーがシステムやサービスを利用したいときに利用できる性質です。

可用性を評価する指標の1 つに稼働率があります。
稼働率は以下の計算式で求められます。

$$\text{稼働率} = \frac{\text{システムが正常に稼働していた時間}}{\text{システムが稼働していた時間}}$$

10

2.12 Kubernetesを使う上での基本的な考え方

2.12.10 可用性を高める手段の例

可用性を高める手段の1 つとして単一障害点の排除があります。

単一障害点：システムの構成要素であり、それが故障するとシステム全体が停止するもの

単一障害点を排除するための1 つの方法として冗長化があります。

冗長化：予備を用意しておくこと。故障発見時に交換することでシステム全体の停止を防ぐ

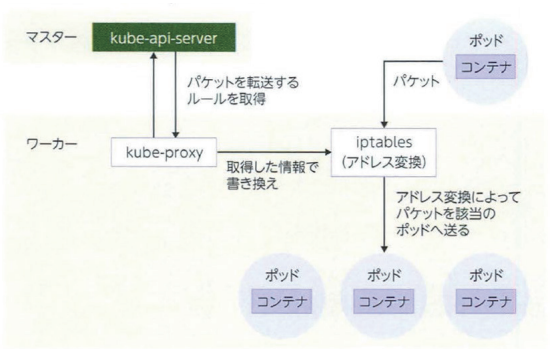
11

2.12 Kubernetesを使う上での基本的な考え方

2.12.11 Kubernetes における負荷分散

Kubernetes ではサービスというしくみで負荷分散を行います。

具体的な分散方法：kube-proxy による転送ルールに従い、iptables のネットワークアドレス変換機能を使用して
該当のポッドへパケットを送る

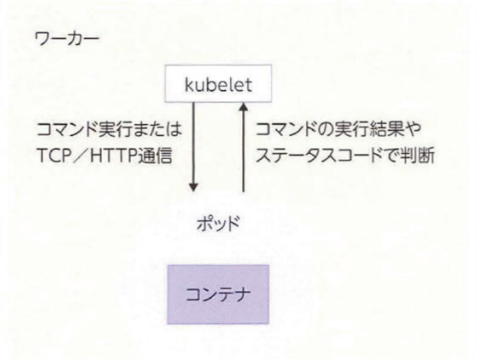


12

2.12 Kubernetesを使う上での基本的な考え方

2.12.12 Kubernetes における死活監視

Kubernetes では、kubeletというエージェントプログラムでポッドの状態を定期的にチェックします。



13

2.12 Kubernetesを使う上での基本的な考え方

2.12.13 スケーリング

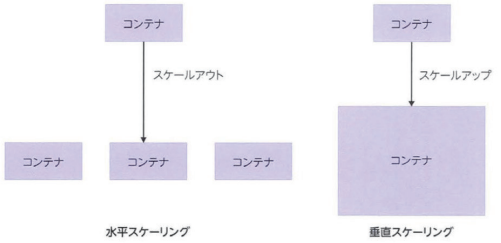
スケーリング：コンテナ技術においてはリクエストの規模に応じてコンテナを増減させること

14

2.12 Kubernetesを使う上での基本的な考え方

2.12.14 スケーリングを行う方法

スケーリングには2つの種類があります。スケーリングを行うのは以下のような場合です。
・リクエスト件数などのデータを定期的に取得し、取得したデータが一定のしきい値を上回る／下回る場合
・過去数回分のデータを参照し、連続して一定のしきい値を上回る／下回る場合



水平スケーリング：リソースの数を増減させる
垂直スケーリング：リソースの性能を上下させる

15

2.12 Kubernetesを使う上での基本的な考え方

2.12.15 水平スケーリング

Horizontal Pod Autoscaler を使用してポッドの水平スケーリングを行います。
コントローラは定期的にポッドのメトリクスを取得し、取得したメトリクスに基づいて必要なポッド数を算出し、レプリカセットやデプロイメントのポッド数を更新します。

メトリクスを取得する頻度：デフォルトでは15 秒おき
メトリクス：CPU 使用率（必要に応じて他の指標をカスタムメトリクスとして使用可能）

$$\text{①必要なポッド数} = \text{現在のポッド数} \times \frac{\text{現在のCPU使用率}}{\text{期待するCPU使用率}}$$

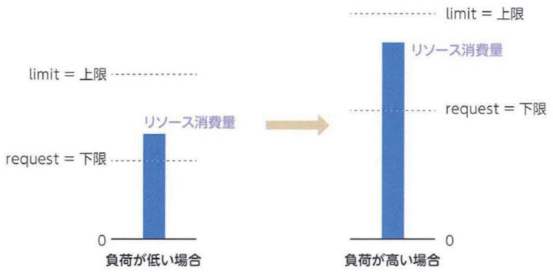
②必要なポッド数の小数点以下を切り上げ

16

2.12 Kubernetesを使う上での基本的な考え方

2.12.16 垂直スケーリング

Horizontal Pod Autoscaler を使用してポッドの垂直スケーリングを行います。
ポッドに割り当てるリソースの数量はrequestとlimitの2つのパラメーターによって決まります。
Request：下限、limit：上限



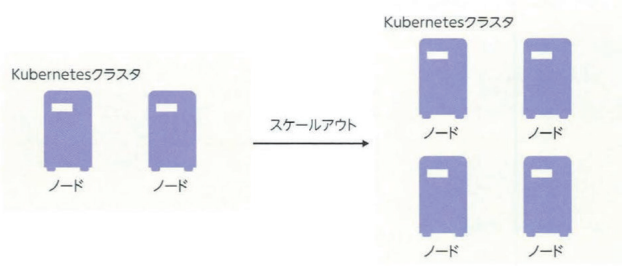
17

2.12 Kubernetesを使う上での基本的な考え方

2.12.17 クラスタ自体の水平スケーリング

Cluster Autoscalerを使用してクラスタ自体の水平スケーリングを行います。
以下のような場合にスケーリングを行います。

・リソース不足でポッドの起動に失敗した場合：クラスタを構成するマシンの数を増やします。
・リソースの使用率が低い状態が一定時間にわたり継続した場合：そのマシンを破棄してマシンの数を調整



18

コンテナ技術の教育プログラム開発
2.13 Kubernetesの基本的な使い方

1

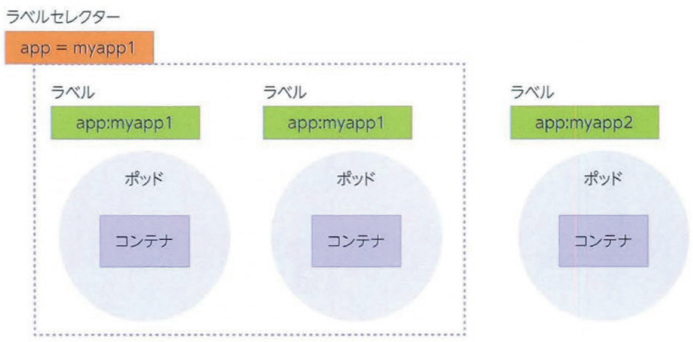
2.13 Kubernetesの基本的な使い方

2.13.1 Kubernetesの基本的な使い方

マニフェストと呼ばれるテキストファイルを使用してクラスタを操作します。

ラベルセレクター：オブジェクトを選択するために指定されるラベルに関する条件

ラベル：オブジェクトに割り当てられたキーとバリューのペアのこと



2

2.13 Kubernetesの基本的な使い方

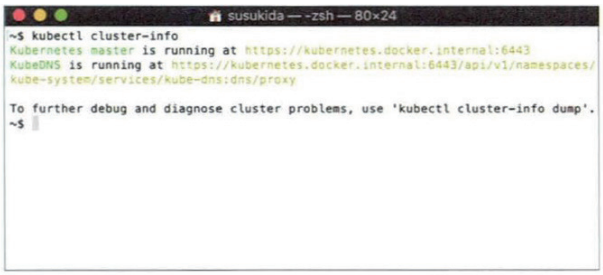
2.13.2 Kubernetes コマンドを利用できる環境

Docker コマンドと同様、Kubernetes コマンドはコマンドライン操作

Linux：ターミナル

Mac：ターミナル

Windows：コマンドプロンプト、PowerShell



3

2.13 Kubernetesの基本的な使い方

2.13.3 ポッドの作成

ポッドを作成するには、**kubectl apply** コマンドを実行します。

```
~$ kubectl apply -f pod.yaml
```

pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  labels:
    app: my-app
spec:
  containers:
    - name: my-container
      image: nginx:stable
```

マニフェストに「pod.yam|」を使用し、「my-pod」という名称のポッドを作成。
作成されるポッドにはキーが「app」、バリューが「my-app」であるラベルが割り当てられるのに加え、「nginx:stable」をイメージとするコンテナが実行されます。
「-f」に指定するマニフェストの内容によって、何を作成／更新するのかが変わります。

4

2.13 Kubernetesの基本的な使い方

2.13.4 ポッドの確認

作成されたポッドを確認するには、**kubectl get pods** コマンドを実行します。

```
~$ kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
my-pod    1/1     Running   0          3s
```

ポッドの状態が「Running（実行中）」であることを確認できます。

5

2.13 Kubernetesの基本的な使い方

2.13.5 ポッドの確認／更新／削除

作成されたポッドを確認するには、**kubectl exec** コマンドを実行します。

```
~$ kubectl exec -it my-pod -- echo Hello Kubernetes
```

作成されたポッドを更新するには、マニフェストに変更を加えてから、
ポッドを作成する時と同じ、**kubectl apply** コマンドを実行します。

```
~$ kubectl apply -f pod.yaml
```

実行中のポッドを削除するには、**kubectl delete** コマンドを実行します。

```
~$ kubectl delete -f pod.yaml
```

6

2.13 Kubernetesの基本的な使い方

2.13.6 レプリカセットの作成

レプリカセットを作成するには、**kubectl apply** コマンドを実行します。

```
~$ kubectl apply -f replicaset.yaml
```

replicaset.yaml

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: my-replicaset
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-container
          image: nginx:stable
```

マニフェストに「replicaset.yaml」を使用し、「my-replicaset」という名称のレプリカセットを作成しています。
このマニフェストを見ると、templateの内容はポッドを作成するときと似ていることがわかります。

作成されるレプリカセットには3つのポッドが含まれ、それぞれのポッドにはキーが「app」であってバリューが「my-app」であるラベルがポッドへ割り当てられるのに加え、「nginx:stable」をイメージとするコンテナが実行されます。

7

2.13 Kubernetesの基本的な使い方

2.13.7 レプリカセットの確認／更新／削除

作成されたレプリカセットを確認するには、**kubectl get replicasets** コマンドを実行します。

```
~$ kubectl get replicasets
NAME            DESIRED   CURRENT   READY   AGE
my-replicaset   3         3         3       10s
```

レプリカセットの更新と削除は、ポッドの場合と同じです。

8

2.13 Kubernetesの基本的な使い方

2.13.8 デプロイメントの作成

デプロイメントを作成するには、**kubectl apply** コマンドを実行します。

```
~$ kubectl apply -f replicaset.yaml
```

replicaset.yaml

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: my-replicaset
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-container
          image: nginx:stable
```

マニフェストに「deployment.yaml」を使用し、「my-deployment」という名称のレプリカセットを作成しています。
このマニフェストを見ると、レプリカセットを作成するマニフェストの内容とほぼ同一であることがわかります。

作成されるデプロイメントには1 つのレプリカセットが含まれます。
また、このレプリカセットには3 つのポッドが含まれ、それぞれのポッドにはキーが「app」であってバリューが「my-app」であるラベルがポッドへ割り当てられるのに加え、「nginx:stable」をイメージとするコンテナが実行されます。

2.13 Kubernetesの基本的な使い方

2.13.9 デプロイメントの確認／更新／削除

作成されたデプロイメントを確認するには、**kubectl get deployments** コマンドを実行します。

```
~$ kubectl get deployments
NAME            DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
my-deployment   3         3         3            3           20s
```

デプロイメントの更新と削除は、ポッドの場合と同じです。

2.13 Kubernetesの基本的な使い方

2.13.10 サービスの作成

サービスを作成するには、**kubectl apply** コマンドを実行します。

```
~$ kubectl apply -f service.yaml
```

service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: ClusterIP
  ports:
    - name: http
      protocol: TCP
      port: 8080
      targetPort: 80
  selector:
    app: my-app
```

ここでは、マニフェストに「service. yaml」を使用し、「my-service」という名称のサービスを作成しています。

作成されるサービスでは、TCP/8080 へのアクセスがポッドのTCP/80へ転送されるルールが含まれ、このルールは「http」と命名されます。

また、アクセスの転送先には、キーが「app」であってバリューが「my-app」であるラベルが割り当てられたポッドが、選択されます。

11

2.13 Kubernetesの基本的な使い方

2.13.11 サービスの確認／更新／削除

作成されたサービスを確認するには、**kubectl get services** コマンドを実行します。

```
~$ kubectl get services
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
my-service   ClusterIP   1.2.3.4      <none>        8080/TCP   15s
```

サービスの更新と削除は、ポッドの場合と同じです。

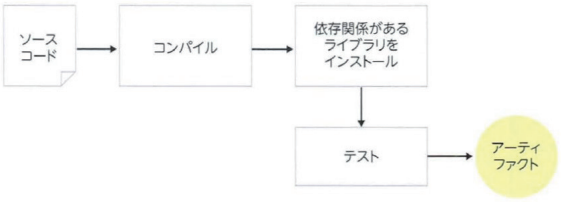
12

2.13 Kubernetesの基本的な使い方

2.13.12 自動ビルドとは

自動ビルド：プログラムなどを利用可能な状態にする手順を自動化すること

＜プログラムをリリース可能な状態にするために一般的に必要な手順＞
プログラムをバイナリコードへ変換するコンパイル、プログラムが外部のライブラリを必要とする場合、依存関係があるライブラリをあらかじめインストール、テストの実施 など



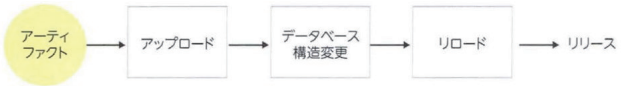
13

2.13 Kubernetesの基本的な使い方

2.12.13 自動デプロイとは

自動デプロイ：プログラムなとをリリースする手順を自動化すること

＜プログラムをリリースするために一般的に必要な手順＞
サーバーにプログラムをアップロードする、データベースの構造変更などの作業が必要　など



14

2.13 Kubernetesの基本的な使い方

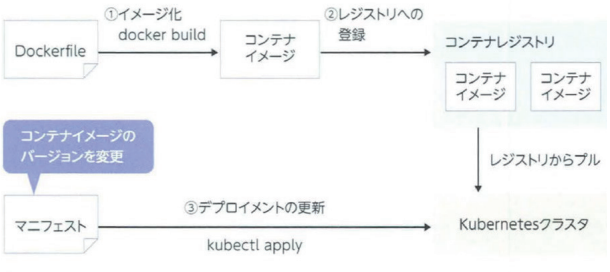
2.13.14 イメージ化からデプロイまでの手順

変更をリリースするまでに「イメージ化」「レジストリへの登録」「デプロイメントの更新」の3 つの手順が必要です。

①イメージ化
Dockerfileを利用するなどしてコンテナのイメージを作成

②レジストリへの登録
イメージをレジストリへプッシュしプルするための準備

③デプロイメントの更新
デプロイメントのマニフェストに記述されたイメージのバージョンを変更
その後にkubectlコマンドを利用して変更を適用
変更を適用したら古いコンテナを含むポッドを停止すると同時に、
新しいコンテナを含むポッドを起動



15

2.13 Kubernetesの基本的な使い方

2.13.15 新しいポッドに置き換える方法

デプロイメントの更新では、古いポッドを新しいポッドに置き換える処理が実施されます。
このポッドを置き換える方法は「更新戦略」と呼ばれ、「Recreate」と「Rolling Update」の2つがサポートされます。

Recreate
新しいポッドが起動する前に古いポッドがすべて停止

Rolling Updat
古いポッドの数を減らしていくと同時に
新しいポッドの数を増やしていく

16

2.13 Kubernetesの基本的な使い方

2.13.16 マルチクラウド

マルチクラウド：複数のクラウドを組み合わせて構成された環境のこと

クラウドは、以下の2つに分けられます。

パブリッククラウド：一般向けに提供されるクラウド

プライベートクラウド：特定の個人や組織向けに提供されるクラウド

17

2.13 Kubernetesの基本的な使い方

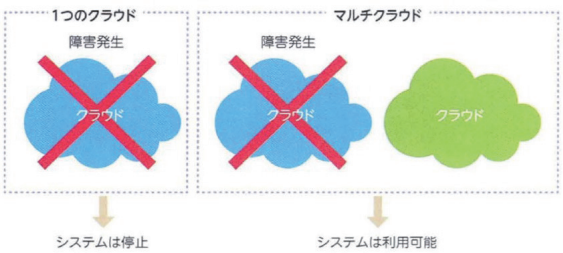
2.13.17 マルチクラウドの必要性

＜1つのクラウド上で構成されたシステムの場合＞

利用しているクラウドで障害が発生するとシステムが利用できなくなるなどの事態を招く恐れがあります。

＜複数のクラウドを組み合わせて構成されたシステムの場合＞

あるクラウドに障害が発生してもそのクラウドを切り離せば、システムが利用できる状態を維持できます。

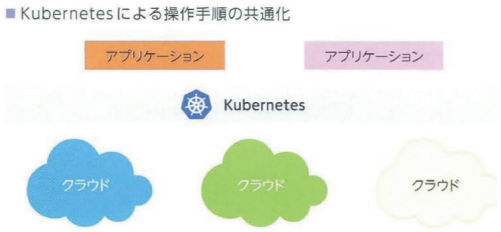


18

2.13 Kubernetesの基本的な使い方

2.13.18 Kubernetes によるマルチクラウド

Kubernetes を使うと、異なるクラウドを共通の手順で操作することが可能となり、
クラウド間の相違点を吸収して開発や運用の複雑化を低減することができます。



19

コンテナ技術の教育プログラム開発
2.14 AWSとGCPとAzure

1

2.14 AWSとGCPとAzure

2.14.1 主要なクラウドコンピューティングプロバイダー

クラウドコンピューティングプロバイダーはオンデマンドでリソースを提供し、アプリケーションやサービスの実行に必要なインフラストラクチャを提供しています。
ここでは、主要な3つのプロバイダーが提供するクラウドサービスについて紹介します。

サービス名	提供元	特徴	活用例
AWS (Amazon Web Services)	Amazon.com	世界最大のクラウドプロバイダー。 多彩なクラウドサービスと高い可用性を提供。	Netflix、Airbnb、NASAなど
GCP (Google Cloud Platform)	Google	機械学習とデータ分析に強み。 スケーラビリティと高速なクラウドサービスを提供。	Spotify、Snapchat、HSBCなど
Azure (Microsoft Azure)	Microsoft	Windowsベースのアプリケーションに最適化。 幅広いサービスとMicrosoft製品との統合。	BMW、Samsung、GEなど

2

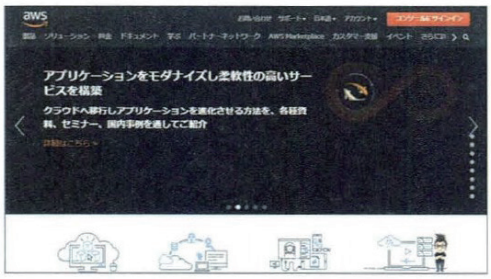
2.14 AWSとGCPとAzure

2.14.2 AWS

AWS はIaaSおよびPaaS市場で圧倒的なシェアを持つAmazon社のクラウドサービスです。
他の2 つのクラウドサービスと比べてもっとも早く、クラウドサービスの提供を開始しました。

AWS (Amazon Web Services)
提供元：Amazon.com
特徴：世界最大のクラウドプロバイダー。
　　多彩なクラウドサービスと高い可用性を提供。
利用例：Netflix、Airbnb、NASAなど

AWS



3

2.14 AWSとGCPとAzure

2.14.3 GCP

GCP は、Kubernetes を開発したGoogle 社が提供しているクラウドサービスです。
もっとも早くKubernetes 関連のクラウドサービスを提供し始めました。

GCP (Google Cloud Platform)
提供元：Google
特徴：機械学習とデータ分析に強み。
　　スケーラビリティと高速なクラウドサービスを提供。
利用例：Spotify、Snapchat、HSBCなど

GCP

4

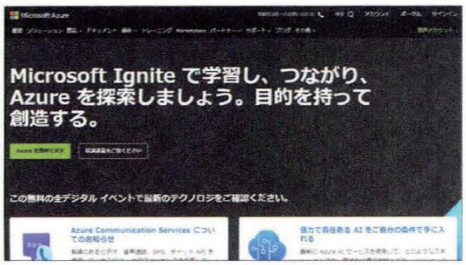
2.14 AWSとGCPとAzure

2.14.4 Azure

Azureは、Saas市場で高いシェアを持つクラウドサービスです。
Windows Server を使用したオンプレミスの環境から連携や移行がしやすいのが特徴です。

Azure (Microsoft Azure)
提供元：Microsoft
特徴：Windowsベースのアプリケーションに最適化。
　　幅広いサービスとMicrosoft製品との統合。
利用例：BMW、Samsung、GEなど

Azure

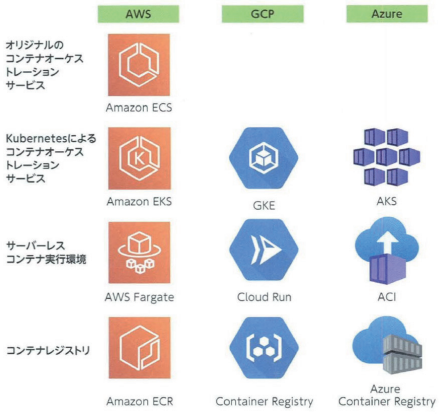


5

2.14 AWSとGCPとAzure

2.14.5 各クラウドで提供されているコンテナサービス

AWS 、GCP、Azureでは、互いによく似たコンテナサービスが提供されています。

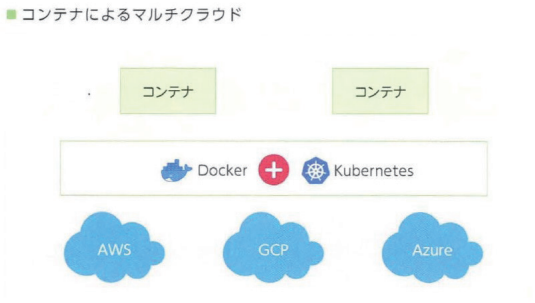


6

2.14 AWSとGCPとAzure

2.14.6 どのクラウドサービスを使うべきか

コンテナサービスの内容や価格だけではなく、情報セキュリティ管理システム認証などコンテナ以外の点にも着目し、総合的に判断することが必要です。
どれか一つではなく、複数のクラウドサービスを併用したマルチクラウドという選択肢も検討できます。



7

コンテナ技術の教育プログラム開発
2.15 AWSとは

1

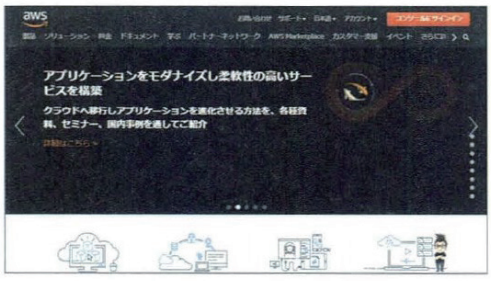
2.15 AWSとは

2.15.1 AWS

AWS はIaaSおよびPaaS市場で圧倒的なシェアを持つAmazon社のクラウドサービスです。
他の2 つのクラウドサービスと比べてもっとも早く、クラウドサービスの提供を開始しました。

AWS (Amazon Web Services)
提供元：Amazon.com
特徴：世界最大のクラウドプロバイダー。
　　多彩なクラウドサービスと高い可用性を提供。
利用例：Netflix、Airbnb、NASAなど

AWS



2

2.15 AWSとは

2.15.2 AWSが提供するサービス

AWS では、Kubenetes をベースとしたサービスや、AWS独自のオーケストレータ、
マネージドなリポジトリ等、コンテナに関連した多数のサービスが存在します。

＜AWSのサービス＞

・Amazon Elastic Container Service (ECS)

・Amazon Elastic Kubernetes Service (EKS)

・AWS Fargate

3

2.15 AWSとは

2.15.3 Amazon Elastic Container Service (ECS)

Amazon ECS は、コンテナのオーケストレーションを行うAWS のサービスです。

主な特徴

・AWSオリジナルのコンテナオーケストレーションサービス

・コンテナイメージを管理する機能を備えていないが、Amazon ECRを利用すればコンテナイメージの登録や配布を行うことができる

・FargateまたはEC2という起動タイプを選択できる

4

2.15 AWSとは

2.15.4 ECSのしくみ

Kubernetesと比較しながら、Amazon ECSのしくみについてみていきましょう。

・サービスが複数のタスクを含み、タスクが最大10 個のコンテナを含む3 層構造
・サービスは、Kubernetes におけるデプロイメントとレプリカセットの2つの役割を担い、タスクのローリングアップデートやタスクの死活監視などを行える
・タスクはKubernetes におけるポッドの役割を担っている
・サービス定義とタスク定義はサービスやタスクの仕様をJSON形式で記述したテキストファイルで、Kubernetesにおけるマニフェストと同様

■ Amazon ECSのしくみ

5

2.15 AWSとは

2.15.5 ECSのメリットとデメリット

ECSを活用することにはメリットがありますが、一方でデメリットもあります。
メリットとデメリットは表裏一体の関係にあるため、状況に応じて両面から検討しましょう。

＜メリット＞
・インフラストラクチャの管理・運用をAWS に任せることができる
・他のAWSサービスとの連携が容易

＜デメリット＞
・インフラストラクチャの管理・運用の権限をAWS が持つことになるため、情報セキュリティマネージメントなどの要件によってはデメリット
・AWS に依存してしまうとマルチクラウドへの展開が困難

■ Amazon ECSはAWSサービスとの連携が容易

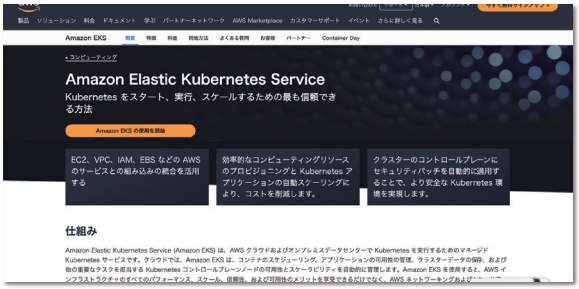
6

2.15 AWSとは

2.15.6 Amazon Elastic Kubernetes Service (EKS)

Amazon EKS は、コンテナのオーケストレーションを行うAWS のサービスです。
Amazon ECSと並び、AWS で利用できるもう一つのコンテナオーケストレーションサービスです。

主な特徴
・Kubernetes クラスタの管理・運用を自動化できる
・自らインフラストラクチャを管理・運用するのと比べて負荷を軽減することができる
・FargateまたはEC2という起動タイプを選択できる



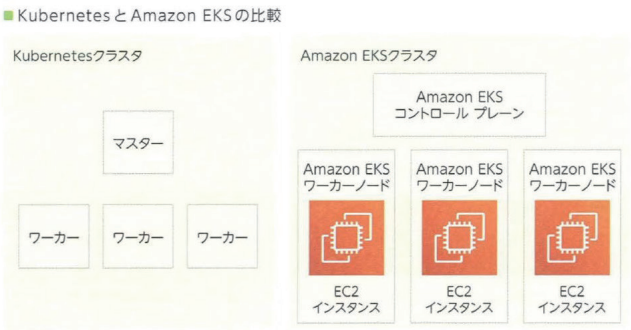
7

2.15 AWSとは

2.15.7 EKSのしくみ

Amazon EKSのしくみについてみていきましょう。

・Amazon EKS におけるクラスタはAmazon EKS コントロールプレーンとAmazon EKS ワーカーノードから構成される
・Amazon EKS コントロールプレーンはKubernetes クラスタにおけるマスターにあたり、Amazon EKS によって管理・運用される
・AWS ではデータセンターをリージョン（複数のアベイラビリティゾーンから構成される）という単位で地理的に分割して運用しているため、可用性が高い



8

2.15 AWSとは

2.15.8 リージョンとアベイラビリティゾーン

AWS ではデータセンターをリージョンという単位で地理的に分割して運用しています。
リージョンは複数のアベイラビリティゾーンから構成されています。

・リージョンは、物理的なデータセンターの集合体で、
AWSのサービスが提供される地域
・アベイラビリティゾーンは、リージョン内の独立したデータセンターまたはデータセンターグループ
・AWS は2023年12月現在、全世界に32 のリージョンと、
102のアベイラビリティーゾーンを運用

9

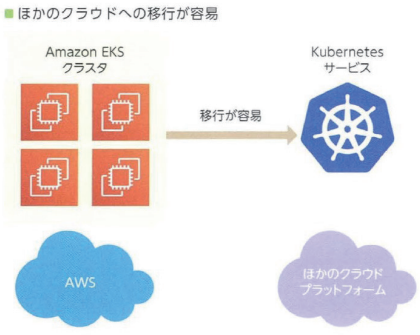
2.15 AWSとは

2.15.9 EKSのメリットとデメリット

EKSを活用することにはメリットがありますが、一方でデメリットもあります（ECSと類似）。
メリットとデメリットは表裏一体の関係にあるため、状況に応じて両面から検討しましょう。

＜メリット＞
・インフラストラクチャの管理・運用をAWS に任せることができる
・他のAWSサービスとの連携が容易
・AWS からAWS 以外で運用しているKubernetes クラスタへの移行について容易になる場合がある

＜デメリット＞
・インフラストラクチャの管理・運用の権限をAWS が持つことになるため、情報セキュリティマネージメントなどの要件によってはデメリット
・AWS に依存してしまうとマルチクラウドへの展開が困難



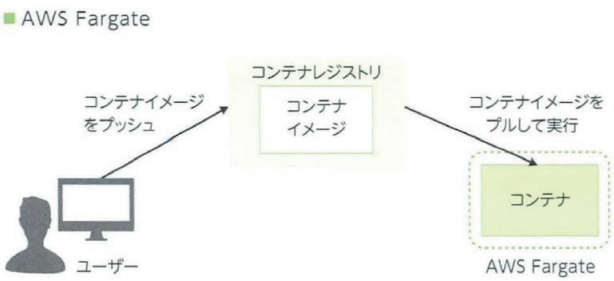
10

2.15 AWSとは

2.15.10 AWS Fargate

AWS Fargate は、サーバーレスなコンテナ実行環境を提供するAWS のサービスです。
コンテナイメージを指定するだけで、コンテナを実行できます。

主な特徴
・クラスタ管理が不要であり、コンテナを運用する負荷を軽減できる
・利用料金はリソースの使用量と使用時間をベースに計算され、コンテナを実行していない場合は利用料金は発生しない



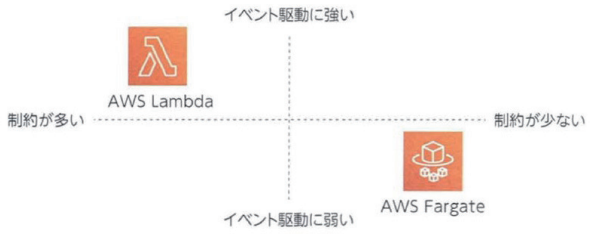
11

2.15 AWSとは

2.15.11 AWS FargateとAWS Lambdaとの違い

AWS はサーバーレスなプログラム実行環境サービスとしてAWS Lambdaを提供しています。
AWS Fargate とは起動時に指定するものがソースコードかコンテナイメージかという点が異なりますが、実行環境を提供するサービスという点で役割が共通しています。

＜使い分けるポイント＞
・AWS Lambdaでは実行時間の上限が15 分であり、実行時間が長い場合はAWS Fargate が適している
・AWS Lambda ではデプロイパッケージサイズの上限が250MB であり、サイズが大きい場合はAWS Fargate が適している
・HTTPS リクエストを受信したときにプログラムを実行したい場合、イベントが発生したときだけ課金されるAWS Lambda が適している



12

コンテナ技術の教育プログラム開発
2.16 AWSの主要なサービス

1

2.16 AWSの主要なサービス

2.16.1 サービスの多様性と選択

AWSには多様なサービスが存在しています。
同じ目的のために複数のサービスが存在していますが、それぞれ異なるニーズや要件に対応しています。

＜サービス選択時に考慮すべき要素＞

パフォーマンス要件：必要な処理能力やスケーラビリティを満たせるか。

コスト：予算に合わせたサービス選択が可能か。

セキュリティとコンプライアンス：業界の規制やセキュリティ要件に適合しているか。

運用の容易さ：管理や運用が簡単か、あるいは専門的な知識が必要か。

統合性：既存のシステムや他のAWSサービスとの連携が容易か。

2

2.16 AWSの主要なサービス

2.16.2 サービスの組み合わせの必要性

AWSの各サービスは、それぞれ特定の機能や役割を担っています。
それらは単体で使用するよりも、他のサービスと組み合わせることによって最大の効果を発揮します。

＜ビジネスニーズに応じたサービスの組み合わせ例＞
・ウェブアプリケーション
Amazon EC2 でホストされるウェブサーバーと、バックエンドのデータストレージとしての Amazon RDS、ユーザー通知用の Amazon SES の組み合わせ。

・eコマースプラットフォーム
顧客データの管理に Amazon RDS を、商品カタログのホスティングに Amazon EC2 を使用し、顧客へのメールマーケティングやトランザクションメールに Amazon SES を利用。

3

2.16 AWSの主要なサービス

2.16.3 AWSの主要なサービス

AWS は世界最大のクラウドプロバイダーで、多彩なクラウドサービスを提供しています。

＜AWSの主要なサービス＞

メール：Amazon Simple Email Service（SES）

データベース：Amazon Relational Database Service (RDS)

サーバー：Amazon Elastic Compute Cloud (EC2)

4

2.16 AWSの主要なサービス

2.16.4 Amazon Simple Email Service (SES)

Amazon SESは、電子メール送信を簡単かつ効率的に行うためのクラウドベースのメールサービスです。

主な機能

・メール配信
　マーケティング、トランザクショナル、通知メールの配信に対応。
・スケーラビリティ
　大規模なメールキャンペーンや急増するメール需要に対応するためのスケールアップが可能。
・従量課金
　セキュリティ対策として、SPF、DKIM、DMARCをサポートし、データの保護や規制遵守を支援。
・分析機能
　配信率、バウンス率、開封率などのメトリクスを提供し、メールキャンペーンの効果測定を支援。

5

2.16 AWSの主要なサービス

2.16.5 Amazon Relational Database Service (RDS)

Amazon RDSはクラウドでデータベースを簡単に設定、運用、スケーリングするためのサービスです。

主な機能

・サポートされるデータベースエンジン
　MySQL、PostgreSQL、MariaDB、Oracle、SQL Serverなど。
・管理とメンテナンス
　バックアップ、パッチ管理、レプリケーションを自動化。
・パフォーマンス
　高可用性、高耐久性を実現し、負荷の高いアプリケーションにも対応。
・セキュリティ
　VPCでのネットワーク分離、暗号化、IAMによるアクセス制御を提供。

6

2.16 AWSの主要なサービス

2.16.6 Amazon Elastic Compute Cloud (EC2)

Amazon EC2は、スケーラブルなクラウドコンピューティング容量を提供するサービスです。

主な特徴

・インスタンスタイプ
　さまざまなコンピューティングニーズに対応する多様なインスタンスタイプ。
・スケーラビリティ
　必要に応じて容量を増減可能。オートスケーリングによる効率的なリソース管理。
・高い耐障害性
　複数のロケーションにインスタンスを配置することができ、他のアベイラビリティーゾーンで発生した障害の影響を受けない。

7

コンテナ技術の教育プログラム開発
2.17 コンテナを利用したAWSアーキテクチャ

1

2.17 コンテナを利用したAWSアーキテクチャ

2.17.1 Well-Architectedフレームワークの活用

AWSのアーキテクチャとは、システムやアプリケーションの設計構造を指します。
多様なAWSサービスと技術を組み合わせて、特定のビジネスニーズや要件に合わせて最適化されます。

＜6つの柱＞


運用上の優秀性


セキュリティ


信頼性


パフォーマンス効率


コスト最適化


持続可能性

➡ 柱ごとに固有の設計原則が定義されています。

2

2.17 コンテナを利用したAWSアーキテクチャ

2.17.2 一つ目の柱「運用上の優秀性」

目的：新機能とバグ修正を迅速かつ確実にユーザーに提供すること

＜設計原則＞

・運用をコードとして実行する
・小規模かつ可逆的な変更を頻繁に行う
・運用手順を頻繁に改善する
・障害を予想する
・運用上の障害すべてから学ぶ

3

2.17 コンテナを利用したAWSアーキテクチャ

2.17.3 二つ目の柱「セキュリティ」

目的：データとシステムのセキュリティを保護すること

＜設計原則＞

・強力なアイデンティティ基盤を実装する
・トレーサビリティの維持
・すべてのレイヤーでセキュリティを適用する
・セキュリティのベストプラクティスを自動化する
・伝送中および保管中のデータを保護する
・データに人の手を入れない
・セキュリティイベントに備える

4

2.17 コンテナを利用したAWSアーキテクチャ

2.17.4 三つ目の柱「信頼性」

目的：システムが意図したとおりに正確かつ一貫して機能し続ける能力を確保すること

＜設計原則＞

・障害から自動的に復旧する
・復旧手順をテストする
・水平方向にスケールして集合的なワークロードの
可用性を向上する
・キャパシティーを勘に頼らない
・オートメーションで変更を管理する

5

2.17 コンテナを利用したAWSアーキテクチャ

2.17.5 四つ目の柱「パフォーマンス効率」

目的：リソースを最適に利用して、最大限のパフォーマンスを達成すること

＜設計原則＞

・最新テクノロジーの標準化
・わずか数分でグローバル展開する
・サーバーレスアーキテクチャを使用する
・より頻繁に実験する
・システムに対する精通の程度を考慮する

6

2.17 コンテナを利用したAWSアーキテクチャ

2.17.6 五つ目の柱「コスト最適化」

目的：クラウドリソースのコストを効率的に管理し、全体的な費用対効果を最大化すること

＜設計原則＞

・クラウド財務管理を実装する
・消費モデルを導入する
・全体的な効率を測定する
・差別化につながらない高負荷の作業に費用をかけるのをやめる
・費用を分析し帰属関係を明らかにする

7

2.17 コンテナを利用したAWSアーキテクチャ

2.17.7 六つ目の柱「持続可能性」

目的：環境に配慮したクラウドコンピューティングの実践を通じて、
環境への影響を最小限に抑えつつ、効率的かつ責任あるリソース使用を促進すること

＜設計原則＞

・影響を理解する
・持続可能性の目標を設定する
・使用率を最大化する
・より効率的なハードウェアやソフトウェアの新製品を予測して採用する
・マネージドサービスを使用する
・クラウドワークロードのダウンストリームの影響を軽減する

8

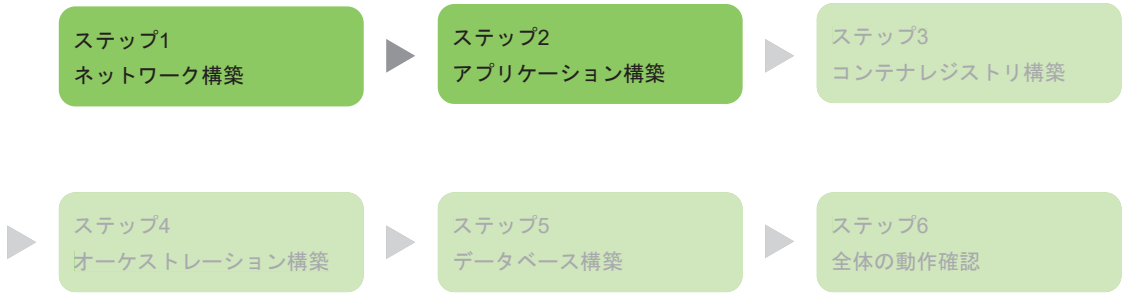
コンテナ技術の教育プログラム開発
2.18 AWSの使い方①

1

2.18 AWSの使い方①

2.18.1 Web システム構築の流れ

コンテナとオーケストレータをメインにAWS上でWeb システムを構築する流れは以下の通りです。
ここでは、ステップ1からステップ2までの工程について解説します。

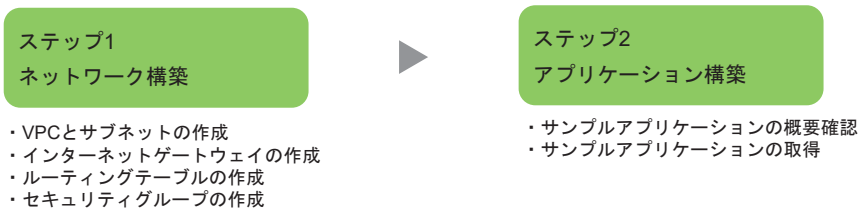


2

2.18 AWSの使い方①

2.18.2 ネットワーク構築～アプリケーション構築

ネットワーク構築とアプリケーション構築は、それぞれ以下のような工程となっています。



3

2.18 AWSの使い方①

2.18.3 ネットワーク構築

ネットワーク構築は以下の流れで進めていきます。

① AWS の東京リージョンに新規VPCを作成（マルチAZ構成が前提）

②AZ（ Availability Zone ） ごとに「パブリックサブネット」と「プライベートサブネット」を作成

③インターネットゲートウェイを作成

④ルートテーブルを設定

⑤セキュリティグループを設定

⑥CloudFormationを利用してリソースを作成

4

2.18 AWSの使い方①

2.18.4 CloudFormation

CloudFormation：AWSリソースのプロビジョニングと管理を自動化するためのサービス
AWSリソースのセットアップと構成をコードで容易に行うことができます。

＜主な特徴＞

テンプレートベースの構成：リソースの構成情報をJSONまたはYAML形式のテンプレートファイルに記述

自動化と一貫性：テンプレートを使用してリソースをデプロイすることで、迅速なデプロイメントが実現

簡単なアップデートとロールバック：テンプレートを更新するだけでアップデートやロールバックが可能

リソースの依存関係の管理：リソース間の依存関係を処理し、指定された順序で自動的に作成および削除する

コストとセキュリティの管理：テンプレートを使用することで、コストとセキュリティ管理を最適化できる

5

2.18 AWSの使い方①

2.18.5 アプリケーション構築

フロントエンドとバックエンドのアプリケーションをGitHubから取得する流れを解説します。
なお、ローカル環境ではなく、AWS 上で実行可能なWebIDEであるAWS Cloud9 を利用する想定です。

①Cloud9の名称設定／インスタンスタイプ選択

②Cloud9 の起動／ロール／ネットワーク設定

③Cloud9の環境作成／セキュリティグループの設定

④アプリケーションのソースコード取得（ GitHubからCloud9へ）

⑤アプリケーションの取得結果の確認

6

2.18 AWSの使い方①

2.18.6 AWS Cloud9

AWS Cloud9： AWSが提供しているクラウドベースの統合開発環境（IDE）

ブラウザ上で直接コードを記述、実行、デバッグし、AWSリソースの管理が可能になります。

＜主な特徴＞

クラウドベース：インストールが不要で、インターネットに接続された任意のブラウザからアクセス可能

組み込みのAWSツールサポート：AWSリソースへの簡単なアクセスと管理が可能

コラボレーション機能：複数の開発者がリアルタイムで同じコードベース上で協力し作業することが可能

サポート言語とツール：JavaScript、Python、PHPなど複数のプログラミング言語に対応

直接のコード実行とデバッグ：コードを直接実行し、ステップごとにデバッグすることが可能

7

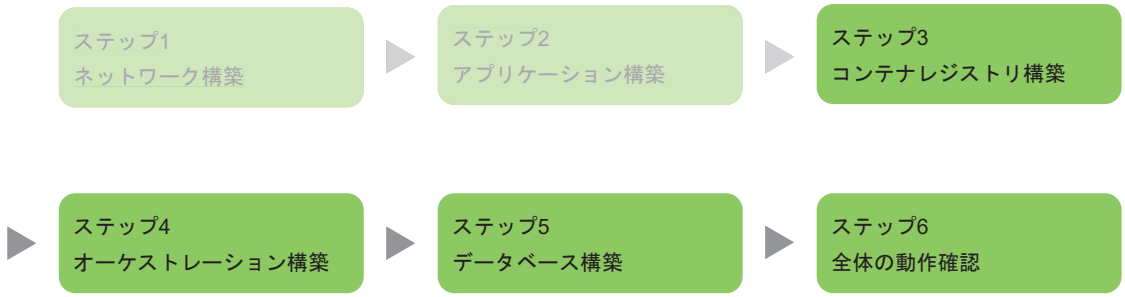
コンテナ技術の教育プログラム開発
2.19 AWSの使い方②

1

2.19 AWSの使い方②

2.19.1 Web システム構築の流れ

コンテナとオーケストレータをメインにAWS上でWeb システムを構築する流れは以下の通りです。
ここでは、ステップ3からステップ6までの工程について解説します。

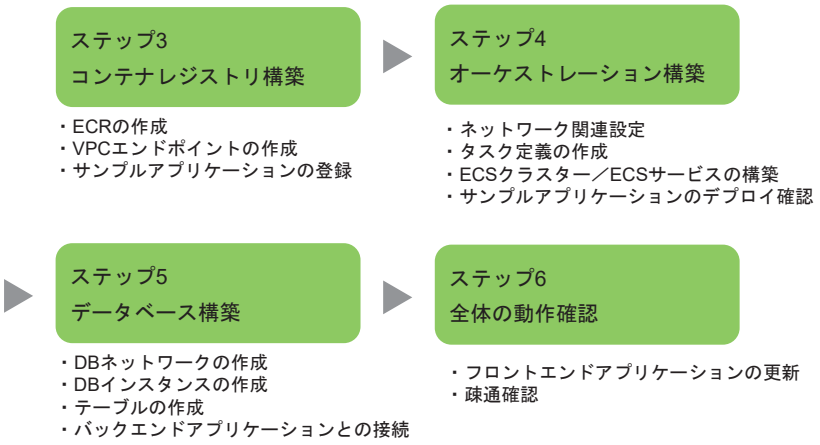


2

2.19 AWSの使い方②

2.19.2 コンテナレジストリ構築～全体の動作確認

コンテナとオーケストレータをメインにAWS上でWeb システムを構築する流れは以下の通りです。



3

2.19 AWSの使い方②

2.19.3 コンテナレジストリ構築

コンテナレジストリ構築は以下の流れで進めていきます。

①コンテナレジストリの作成

②コンテナレジストリへのネットワーク作成

③アプリケーションのビルド

④コンテナビルド（コンテナイメージの作成）

⑤コンテナイメージをコンテナレジストリに登録（プッシュ）

⑥コンテナレジストリからイメージを取得（プル）後にデプロイ

4

2.19 AWSの使い方②

2.19.4 オーケストレータ構築

オーケストレータ構築は以下の流れで進めていきます。

①コンテナからログを転送するために、CloudWatch用のインタフェース型VPCエンドポイントを作成

②フロントエンドアプリケーションからリクエストを受け付けるロードバランサーである内部向けALB を作成

③ ECS の作業としてタスク定義、ECS クラスター、ECS サービスの順番に設定を追加

④最後に、ECS がECR からコンテナを取得してFargate上にテプロイするところまでを確認

その後、正しくデプロイされているかは以下の手順で確認します。

①バックエンドアプリケーションへ直接HTTP リクエストを送信することで応答を確認

②フロントエンドアプリケーションからHTTP リクエストを送信することで応答を確認

5

2.19 AWSの使い方②

2.19.5 データベース構築

データベース構築は以下の流れで進めていきます。
ここでは、Cloud9からAmazon Auroraに接続する想定で流れをみていきます。

①データベース用セキュリティグループの作成

②Auroraインスタンスの作成

③テーブルとサンプルデータの作成

④Secrets Managerの設定と、アクセスする情報をIAM ロールへ設定

⑤コンテナの環境変数の設定

⑥Secrets Manager用のVPC エンドポイントの作成

⑦データベースへの接続確認

6

2.19 AWSの使い方②

2.19.6 アプリケーション間の疎通確認

疎通確認：ネットワーク上でのデバイス間やシステム間の通信が正しく行われているかを
確認するプロセス

ここでは、フロントエンドアプリケーションからバックエンドアプリケーションに接続できているかどうか、バックエンドアプリケーションからデータベースに接続できているかどうか、について確認します。

どちらも問題なく接続が確認できれば、アプリケーションを無事に稼働させられています。

7

コンテナ技術の教育プログラム開発
2.20 AWSの使い方③

1

2.20 AWSの使い方③

2.20.1 実践的なコンテナ構成

ここまで、アプリケーションを動かすための基本的なコンテナ構成について解説してきました。
しかし、実践的なコンテナ構成として知っておくべき知識はまだまだ多くあります。
実践する必要があるかどうかはプロダクトによりますが、知っておいた方が良いいくつかのコンテナ構成について紹介します。

・水平スケールによる可用性向上

AWS では需要に応じて自動でリソースのスケールが可能なAuto Scalingが利用できます。

・アプリケーションへの不正アクセス防止

AWS WAFを使用すると、脆弱性を狙った攻撃などからアプリケーションを保護できます。

2

2.20 AWSの使い方③

2.20.2 水平スケールによる可用性向上

水平スケールによる可用性向上には、Auto Scalingの機能を利用します。
ECS上で設定する流れは以下となります。

＜Auto Scaling設定の流れ＞

ECS ダッシュボードよりサービス設定を更新します。値は以下が参考になります。

デプロイメントの設定：変更なし
ネットワーク構成：変更なし
スケーリングポリシータイプ：「ターゲットの追跡」を設定
ECS サービスメトリクス：CPU の平均値を対象メトリクスに指定
ターゲット値：CPU負荷のしきい値を設定（80 %など）

設定を完了したらレビュー画面が表示されるため、更新内容を確認してサービスを更新します。

2.20 AWSの使い方③

2.20.3 Auto Scalingの挙動確認

サービスのエンドポイントへ負荷をかけて、Auto Scalingが実際に動作することを確認します。
スケールアウトが発生する負荷をフロントエンドアプリケーションからかけることは困難であるため、
負荷ツールとしてメジャーなabコマンドを利用します。

ab （Apache Bench）コマンド：Apache HTTP server benchmarking tool の略で、
Webサーバー向けのベンチマークツールとして広く用いられています。

＜確認の流れ＞

①ECS ダッシュボードよりサービス設定を更新してAuto Scalingを追加（前スライドの内容）

②abコマンドを実行して一定時間待機

③CPU負荷が高まったこと、ECS クラスターが起動しているタスク数が増加していることを確認

2.20 AWSの使い方③

2.20.4 アプリケーションへの不正アクセス防止

ネットワークセキュリティを意識して、Application Load BalancerへAWS WAFの設定を行います。

AWS WAF：AWS ウェブアプリケーションファイアウォールといい、
一般的な攻撃からウェブアプリケーションを保護

＜AWS WAF設定により対処できる問題＞
・OWASP の出版物に記載されている高リスクの脆弱性や一般的な脆弱性への対策
・クロスサイトスクリプティングへの対策等
・不正なIP アドレスや怪しいIP アドレスに対する対策
・不正なパラメータに対する対策
・SQLインジェクションへの対策

2.20 AWSの使い方③

2.20.5 AWS WAFの構成要素

WAF では複数のルールを組み合わせて、トラフィックがアプリケーションに到達する方法を制御できます。

ルール

特定の条件に基づいてウェブリクエストを許可、ブロック、またはカウントするための条件のセット

ルールグループ

複数のルールをまとめたもので、関連するルールを一つのグループとして管理

ウェブACL

ルールやルールグループを組み合わせ特定のAWSリソースへのアクセスを制御するためのポリシー

確認テスト・演習問題

2.1 コンテナとは

確認テスト

01.コンテナ型の説明として適切なものはどれか。
A.「コンテナ」を利用しアプリケーション単位で仮想化する技術
B. コンテナの最大のメリットはパフォーマンスだが、仮想サーバーの用意が煩雑である
C. コンテナごとに独自のハイパーバイザが必要
D. 仮想マシンやゲストOS を動かす必要がない代わりに、パフォーマンスは低くなる

02.コンテナ技術の歴史について述べた以下の文章で適切なものはどれか。
A.コンテナ技術はここ数年に現れた技術で、2022年に登場した
B.大量のコンテナを同時に使用するようになるにつれ、コンテナの管理が煩雑になるという課題が浮き彫りとなったが、未だ解消できていない
C.コンテナ技術の成長期を経て、2013年に、現在も代表的なコンテナ型仮想化ソフトウェアであるDocker が登場した
D.1979 年に登場したUNIX の「altroot」コマンドは、アプリケーションによるアクセスを指定したディレクトリに限定するというもの

2.1 コンテナとは

確認テスト

03.コンテナのメリットについて述べた以下の文章で適切なものはどれか。
A. アプリケーション導入が大変である一方、パフォーマンスが高まる点にある
B. コンテナ内のアプリケーションはホストOS の機能を利用して動いており、ファイルも共有される
C. コンテナ技術は、仮想マシンは必要としないがゲストOSは必要とする
D. Docker Hubを利用することにより、かんたんなサーバーならコマンドを1つ実行するだけで、仮想サーバーのコンテナを立ち上げることができる

04.コンテナ技術のデータ管理について述べた文章のうち、適切なものはどれか。
A. コンテナ技術では、可能な限りデータを同じコンテナに含め管理しやすくするのが基本
B. コンテナ技術では、仮想サーバーがクラッシュしてもデータは失われない
C.コンテナ技術においてコンテナにデータを含めることはできない
D. データの保管先としてホストOS のディレクトリを指定することはできない

2.2 DevOpsとは

確認テスト①

DevOpsについて、Plan（計画）とMonitor（監視）の間を適切に並び替えてください。

12

2.2 DevOpsとは

確認テスト

01.DevOpsについて説明した文章として適切なものはどれか。

A. 開発（ Development)と運用(Operations) の略語をあわせて作られた造語で、開発者と運用者が協力してサービスを提供する手法のこと

B. 開発者と運用者はそれぞれ担当業務が異なるので、完全に役割分担をしてサービスの提供を迅速に行っていこうというのがDevOpsの考え方

C. DEV のサークルは開発者側で、サービスの開発過程が、「プラン」、「ドゥ」、「チェック」、「アクション」の順に進められることを表す

D. OPS のサークルは、サービスの運用過程が「オープン」、「プレイ」、「スタート」、「オペレーション」の順に進められ、その結果をもとに、サービスの開発へとつなげるための「計画」が実施されることを表す

13

2.2 DevOpsとは

確認テスト

02. DEV のサークルについて説明した文章として適切なものはどれか。

A.Planフェーズでは、ビジョンと目標の設定、スケジュール、要件定義、リスク評価などの活動を通じて全体のプランニングを行う

B.Codeフェーズでは、プログラミングは行わず、Planフェーズで行ったプランニングに沿ってソフトウェアのソースコードを開発するプランを計画する

C.Buildフェーズでは、ソフトウェアのソースコードを作成するため、最適なチームビルディングを行う

D.Testフェーズでは、ソフトウェアが正常に動作することを確認するため、バグなどの修正はそれ以前のフェーズでしっかりと行われていなければならない

14

2.2 DevOpsとは

確認テスト

03.OPS のサークルについて説明した文章として適切なものはどれか。

A.Releaseフェーズでは、開発やテストを経て、新しいバージョンをリリースするが、問題が発生した場合のバックアップはないため、Devでどれだけ慎重に作業できるかが重要

B.Deployフェーズでは、ソフトウェアやアプリケーションを実行環境に配置するが、プランニングは別のフェーズで行う

C.Operateフェーズでは、システムのモニタリング、保守、パフォーマンスの最適化、エラーの検出と修正、セキュリティの維持などを行う

D.Monitorフェーズでは、運用チームが適切に機能するかを定期的に監視し、運用上の人的な問題や異常を早期に検出する

15

2.3 コンテナ技術とそれ以外の技術

確認テスト

01.コンテナの実行や管理を行うソフトウェアのことを総称してなんというか。
A.Kubernetes
B.Docker
C.コンテナランタイム
D.モノリス

02.依存関係のない複数の小さいサービスを組み合わせて大きなサービスとして提供する設計手法をなんというか。
A.コンテナ
B.モノリス
C.マイクロサービス
D.サーバーレス

9

2.3 コンテナ技術とそれ以外の技術

確認テスト

03.主にクラウドで提供されており、サーバーが存在しない環境を指すサービス開発の手法をなんというか。
A.ヘッドレス
B.サーバーレス
C.コンテナレス
D.サービスレス

04.サーバーレスのメリットに関する文章として適切なものはどれか。
A.リソースに制約がなく、長時間処理に向く
B.自由度や可搬性が高い
C.使った分だけの課金でよく使っていない時間帯のコストが発生しない
D.利用料金は定額制だが格安で使いやすい

10

2.4 Dockerとは

確認テスト

01.Dockerに関する文章として適切なものはどれか。
A.コンテナの実行やコンテナイメージの作成を行い、配布は別のプラットフォームを用いる
B.アプリケーションを実行環境ごとパッケージ化したコンテナイメージを作成できる
C.実行環境をまるごとパッケージ化するが、環境によって細かな設定変更が必要
D.世界初のコンテナ技術であり、デファクトスタンダードとして広い支持を集めている

02.コンテナ技術の標準化とオープンソースのコンテナランタイムおよびイメージフォーマットの仕様策定を目的とするプロジェクトをなんというか。
A.Open Container Inception
B.Open Containers Initiative
C.Open Container Incubator
D.Open Container Interaction

2.4 Dockerとは

確認テスト

03.OCIに関する文章として適切なものはどれか。
A.クローズドソースであり、限られたコミュニティの協力によって進行している
B.コンテナの実行に使用されるコンテナランタイムおよびコンテナイメージのフォーマットを標準化している
C.OCIの仕様に準拠したコンテナランタイムやイメージを使用することで、コンテナの移植性を向上させているが、多く使用されるが故のセキュリティ面でのリスクがある
D.コンテナの実行環境やコンテナイメージなどに関する仕様について細かく規定し、ドキュメントの数は20個以上にものぼる

04.Dockerのコンセプトに関する文章として適切なものはどれか。
A.柔軟性：使用するプログラミング言語を限定することで誰でも扱えるようになった
B.セキュア：コンテナ同士の分離や結合を制限することでセキュアな環境を提供
C.ポータブル：事前に登録した実行環境ならどこでも扱える
D.スケーラブル：需要に応じてリソースを増減できる

2.5 Dockerコンテナ

確認テスト②

Dockerコンテナに関する説明として正しいものをすべて選択してください。

1.コンテナイメージは、ベースとなるコンテナイメージのファイルシステムに新たなレイヤーを重ねることによって作成される。

2.コンテナには、CentOS やUbuntu などのOS を載せることができない。

3. Docker のコンテナライフサイクルは、「作成」「実行」「中止」「ポーズ」「再開」「削除」の6つがある。

4.コンテナと仮想マシンはどちらも物理ハードウェアを仮想的に分割し、複数の独立した実行環境を提供することで、異なるアプリケーションやサービスを同じ物理サーバー上で実行できる点で似ている。

13

2.5 Dockerコンテナ

確認テスト

01.プログラムの実行単位のことを指す言葉はどれか。
A.コンテナ
B.プロセス
C.ファイル
D.ライブラリ

02.名前空間とはどのような仕組みですか？
A. ファイルシステムの隔離を提供する機能
B. プロセス間通信を行うための仕組み
C. プロセスのリソースを隔離するためのメカニズム
D. ネットワークの通信を制御する仕組み

14

2.5 Dockerコンテナ

確認テスト

03.コンテナと仮想マシンに関する文章として適切なものはどれか。
A.コンテナは他のコンテナやプロセスとOSを共有する
B.コンテナは共有カーネルを利用するため、リソース消費が多くなりやすい
C.仮想マシンは起動に時間がかかりにくい
D.仮想マシンは独自のゲストOSを持つためリソース消費が効率的である

04.コンテナイメージに関する文章として適切なものはどれか。
A. コンテナの実行時環境を定義する設定ファイル
B. コンテナ内で実行されるアプリケーションのバイナリファイル
C. コンテナの状態や動作を記録したログファイル
D. コンテナのファイルシステムと実行環境を固めたファイルやディレクトリ

15

2.5 Dockerコンテナ

確認テスト

05.コンテナのライフサイクルに関する文章として適切なものはどれか。
A. Docker のコンテナライフサイクルは、「作成」「実行」「中止」「ポーズ」「再開」「削除」の6つ
B.コンテナが作成されてから削除されるまでの「状態の流れ」のことでDocker コマンドの実行などによって変化
C.ポーズはコンテナ内でプロセスが一時停止している状態で、メモリなどの記録は保持されていない
D.停止はコンテナ内でプロセスが一時停止している状態で、メモリなどの記録を保持したまま再開される

16

2.6 Dockerコンテナ

確認テスト

01.コンテナやコンテナイメージを管理するためのアプリケーションをなんというか。
A.Docker デーモン
B.コンテナレジストリ
C.Docker クライアント
D.Docker エンジン

02.Dockerエンジンのコンポーネントに関する文章として適切なものはどれか。
A.DockerデーモンはDocker クライアントからの要求に応じDockerオブジェクトを削除するコンポーネント
B.Docker エンジンは6つのコンポーネントから構成されている
C.コンテナレジストリはコンテナイメージを検証するためのコンポーネント
D.DockerクライアントはDocker デーモンのUIとなるコンポーネント

2.6 Dockerコンテナ

確認テスト

03.Docker オブジェクトに含まれていないものはどれですか。
A.プロセス
B.コンテナイメージ
C.ネットワーク
D.ボリューム

04.複数コンポーネントに分かれていることのメリットに関する文章として適切ではないものはどれか。
A.別々のマシンで動作可能
B.個々に差し替えられる
C.デバッグが容易
D.個々の差し替えは難しいがセキュリティが高い

2.7 Dockerを使う上での基本的な考え方

確認テスト③

Dockerについて説明した文章のうち、正しいものをすべて選択してください。

1. Swarm内のノードに与えられる役割は、マネージャー、ワーカーまたはマネージャー兼ワーカーの3つのうち1つ。

2. Docker ではプロセスの実行によるオーバーヘッドがほぼゼロのため性能の劣化はほぼなく、ボトルネックとなるコンテナを複数起動することで負荷分散が可能になる。

3.永続化とは、データが失われないことを目的として、プログラムが終了しないように省力で稼働し続けられるようにすることである。

4. ファイルが更新されるたび、つまり差分が発生するたびに全てのファイルを保存することによって変更を管理する方法を差分管理という。

2.7 Dockerを使う上での基本的な考え方

確認テスト

01.Dockerについて、プロセスごとにコンテナを分けるメリットとデメリットに関する文章として適切ではないものはどれか。
A.ボトルネックとなるコンテナを複数起動し処理能力の低下を改善できる
B.コンテナが増えるほど管理が複雑になってしまう
C他のソフトウェアと同じリソースを共有することを前提として設計されている場合、プロセスことにコンテナを分けることによって不具合が発生する恐れがある
D.プロセスごとにコンテナを分けると著しく性能が劣化する恐れがある

02.コンテナによって読み書きされるデータを永続化するためのしくみをなんというか。
A.バインドマウント
B.control groups
C.ボリューム
D.名前空間

2.7 Dockerを使う上での基本的な考え方

確認テスト

03.内部のプロセスに対して外部のプロセスから隔離されているように見せるしくみをなんというか。
A.バインドマウント
B.control groups
C.ボリューム
D.名前空間

04.名前空間の種類と隔離することができるリソースについて、正しい組み合わせはどれか。
A.PID名前空間：プロセスID、ポート番号
B.ネットワーク名前空間：ホスト名
C.IPC名前空間：ネットワークインターフェイス
D.マウント名前空間：ファイルシステム

19

2.7 Dockerを使う上での基本的な考え方

確認テスト

05.ハードウェアリソースの使用量を制限するしくみをなんというか。
A.バインドマウント
B.control groups
C.ボリューム
D.名前空間

06.ベースとの相違点を記録していくことによって変更を管理する方法のことをなんというか。
A.バインドマウント
B.control groups
C.ボリューム
D.差分管理

20

2.7 Dockerを使う上での基本的な考え方

確認テスト

07.コンテナイメージに含まれるレイヤーに関する文章として適切なものはどれか。
A.バインドマウント
B.control groups
C.ボリューム
D.名前空間

08.Swarm内のノードの役割に関する文章として適切ではないものはどれか。
A.マネージャーはSwarm全体を管理し、ワーカーにタスクの実行を指示する
B.ワーカーは、マネージャーからの指示に従ってコンテナを実行する役割を担う
C.マネージャー兼ワーカーは、マネージャーとワーカー両方の役割を兼ねるとともに、全ノードのうち1つにだけ役割が割り振られ全マネージャーノードの管理を行う
D.DockerではSwarm内のノードに、マネージャー、ワーカーまたはマネージャー兼ワーカーの3つの役割から1つが与えられる

2.8 Dockerの基本的な使い方

確認テスト

01.Dockerコマンドについて、イメージの作成はどれか。
A.docker image build
B.docker image ls
C.docker image push
D.docker image pull

02.Dockerコマンドについて、イメージ情報の表示はどれか。
A.docker image build
B.docker image ls
C.docker image push
D.docker image pull

18

2.8 Dockerの基本的な使い方

確認テスト

03.Dockerコマンドについて、イメージの削除はどれか。
A.docker image build
B.docker image ls
C.docker image rm
D.docker image pull

04.Dockerコマンドについて、イメージタグの追加はどれか。
A.docker image build
B.docker image tag
C.docker image rm
D.docker image pull

19

2.8 Dockerの基本的な使い方

確認テスト

05.Dockerコマンドについて、レジストリへのイメージ保存はどれか。
A.docker image build
B.docker image tag
C.docker image push
D.docker image pull

06.Dockerコマンドについて、レジストリからのイメージ取得はどれか。
A.docker image build
B.docker image tag
C.docker image push
D.docker image pull

2.8 Dockerの基本的な使い方

確認テスト

07.Dockerコマンドについて、コンテナの実行はどれか。
A.docker image build
B.docker container run
C.docker image push
D.docker image pull

08.Dockerコマンドについて、ログの確認はどれか。
A.docker image build
B.docker container logs
C.docker image push
D.docker image pull

2.8 Dockerの基本的な使い方

確認テスト

09.Dockerコマンドについて、実行中のコンテナに対するコマンド実行はどれか。
A.docker image build
B.docker container logs
C.docker image push
Ddocker container exec

10.Dockerコマンドについて、コンテナの停止はどれか。
A.docker image build
B.docker container stop
C.docker image push
Ddocker container exec

2.8 Dockerの基本的な使い方

確認テスト

11.コンテナイメージの保管場所をなんというか。
A. コンテナ
B.コンテナレジストリ
C.リポジトリ
D.タグ

2.8 Dockerの基本的な使い方

演習①：事前準備

VirtualBoxを利用して仮想環境内にUbuntuをインストールする演習

演習の実施に先駆けて、以下の事前準備をお願いします。

1.1 Docker Desktop for Windowsをインストールしてください
https://docs.docker.jp/docker-for-windows/install.html

1.2 WordPress 6.4.2（ソースコード）をダウンロードしてください
https://ja.wordpress.org/download/#download-install
https://ja.wordpress.org/latest-ja.zip

1.3 nginx.conf をダウンロードしてください
https://github.com/ykeisuke/wp_practice1/blob/master/docker/nginx.conf

※ 各ソフトウェアインストール時に再起動を求められた場合は再起動してください。

2.8 Dockerの基本的な使い方

演習①

VirtualBoxを利用して仮想環境内にUbuntuをインストールする演習

2.1 ダウンロードしたWordPress 6.4.2を解凍してください

2.2 Dockerコンテナにマウントするディレクトリを作成してください

2.3 2.2を利用したdocker-compose.ymlファイルを作成してください

2.4 ダウンロードしたnginx.confを使用してください

2.5 docker compose を使い、コンテナを起動してください

2.8 Dockerの基本的な使い方

演習①：ヒント

VirtualBoxを利用して仮想環境内にUbuntuをインストールする演習

https://github.com/ykeisuke/wp_practice1/ に構築済みの環境があります

2.9コンテナオーケストレーションとは

確認テスト

01.デプロイ、スケーリング、運用を自動化および管理する技術をなんというか。
オーケストレーション

02.システムを構成するコンテナが正常に稼働しているかどうかを監視することをなんというか。
死活監視

2.10 Kubernetesとは

確認テスト④

Kubernetesでできることとして正しいものをすべて選択してください。

1.ストレージの管理ログの記録や監視
2.機密情報や構成情報の保管
3.ローリングアップデートとロールバック
4.自動ビルド
5.エンドポイント提供と負荷分散
6.ミドルウェア的機能
7.マシン自体の管理
8.死活監視と停止したコンテナの再起動
9.消費リソースを考慮したコンテナの配備

2.10 Kubernetesとは

確認テスト

01.コンテナオーケストレーションツールのデファクトスタンダードであるツールをなんというか。
A. Docker
B. Ansible
C. Kubernetes
D. Terraform

02.Kubernetes ができることとして適切ではないものはどれか。
A. 死活監視と停止したコンテナの再起動
B. エンドポイント提供と負荷分散
C. 自動ビルド
D. ストレージの管理

2.10 Kubernetesとは

確認テスト

03.Kubernetesによって管理されるクラスタをなんというか。
A.ワーカーノード
B.コンテナ
C.Kubernetes クラスタ
D.ポッド

04.Kubernetesのアーキテクチャに関する文章として適切なものはどれか。
A.Kubernetes にはマスターとワーカー、マネージャーの3 種類のノードがある
B.ユーザーは、kubectl コマンドなどを使ってマスターに命令を送り、マスターはその命令に従って、ワーカーを操作する
C.マスターとワーカーではそれぞれ1つのコンポーネントが動作しており、そのコンポーネントが連携し合うことで、オーケストレーションは実行されている
D.ユーザーはコマンド使ってKubernetesを操作し、GUIなどは利用できない

2.11 ポッドとデプロイメントコントローラ

確認テスト

01.Kubernetes がコンテナを管理するための最小単位をなんというか。
A. インスタンス
B. マシン
C. ポッド
D. サービス

02.ポッドの必要性に関する文章として適切ではないものはどれか。
A.ポッド内で複数のコンテナを起動する場合はコンテナの管理をKubernetes に任せられる
B.コンテナの管理をKubernetes へ任せることで、運用の負荷やリソースの消費量を減らすことができる
C.コンテナごとに必要最小限のパッケージをインストールしてなるべく余分なものを減らすなど、ソフトウェアの依存関係を柔軟に管理できる
D.ポッド内で複数のコンテナを起動する場合でもプロセスをユーザー自身で管理しなければならないが、消費リソースの面でメリットがある

2.11 ポッドとデプロイメントコントローラ

確認テスト

03.Kubernetesでポッドの複製を管理し、指定した数だけ維持する機能をなんというか。
A. スケーリング
B. ロードバランシング
C. レプリカセット
D. オートスケーリング

04.レプリカセットをさらにグループにして管理する単位をなんというか。
A. ポッド
B. サービス
C. デプロイメント
D. レプリケーションコントローラ

2.11 ポッドとデプロイメントコントローラ

確認テスト

05.現在の状態を期待する状態にするためのメカニズムのことをなんというか。
A. インフラストラクチャ
B. デプロイメント
C. コントローラ
D. サービス

06.デプロイメントをあるべき状態に保つためのコントローラのことをなんというか。
A. ポッドコントローラ
B. レプリケーションコントローラ
C. ステートフルセットコントローラ
D. デプロイメントコントローラ

13

2.12 Kubernetesを使う上での基本的な考え方

確認テスト⑤

用語と説明文の正しくない組み合わせをすべて選択してください。

1.スケーリング：コンテナ技術においては、リクエストの規模に応じてコンテナを増減させること

2.冗長化：予備を用意しておくこと

3.単一障害点：トラブルが発生した際の最も重要な障害となる点のこと

4.サービス(Service)：コンテナへアクセスするための窓口となるエンドポイントを提供するしくみ

5.負荷分散：複数のコンテナへリクエストを振り分けて負荷を分散すること

6.死活監視：システムを構成するコンテナが正常に稼働しているかどうかを監視すること

7.可用性：ユーザーがシステムやサービスを利用したいときに利用できる性質

19

2.12 Kubernetesを使う上での基本的な考え方

確認テスト

01.コンテナへアクセスするための窓口となるエンドポイントを提供するしくみをなんというか。
A. インターフェース
B. ゲートウェイ
C. サービス
D. アクセスポイント

02.4種類あるサービスは4 種類に分類されるが、それぞれの名称と特徴について適切ではない組み合わせはどれか。
A.ClusterIP：クラスタ内部と外部双方からアクセス可能な仮想IPアドレスが発行されます。
B.NodePort：クラスタ内の一部ノードの指定ポートへのアクセスがコンテナへ転送されます。クラスタ外部からはアクセスできません。
C.LoadBalancer：クラウドサービスのロードバランサーへのアクセスがコンテナへ転送されます。クラスタ外部からもアクセス可能です。
D.ExternalName：一部ノードへのアクセスが指定のホストへ転送されます。

20

2.12 Kubernetesを使う上での基本的な考え方

確認テスト

03.Kubernetes のネットワーク実装のひとつで、クラスタを構成する複数のマシン同士をつなぐオーバーレイネットワークを構築する仕組みをなんというか。

A. Flannel

B. Calico

C. CNI

D. kube-proxy

04.flannel がIPv4 ネットワークを構築するために、Kubernetesクラスタを構成するすべてのマシン上で起動するエージェントプログラムをなんというか。

A. Flannel-agent

B. kube-proxy

C. kubelet

D. flanneld

2.12 Kubernetesを使う上での基本的な考え方

確認テスト

05.ユーザーがシステムやサービスを利用したいときに利用できる性質をなんというか。

A. フレキシビリティ

B. アベイラビリティ

C. アクセシビリティ

D. インターフェース

06.可用性を高める手段の例として適切なものはどれか。

A. デバッギング

B. スケーリング

C. テスト

D. デプロイメント

2.12 Kubernetesを使う上での基本的な考え方

確認テスト

07.Kubernetes では、kubeletというエージェントプログラムでポッドの状態を定期的にチェックして何を行っているか。

A. ポッドの削除

B. ポッドの再起動

C. ポッドの監視

D. ポッドの作成

08.コンテナ技術においてリクエストの規模に応じてコンテナを増減させることをなんというか。

A. スケーリング

B. ロードバランシング

C. クラスタリング

D. レプリケーション

2.13 Kubernetesの基本的な使い方

確認テスト

01.Kubernetes では何と呼ばれるテキストファイルを使用して、クラスタを操作するか。
A. プランファイル
B. マニフェストファイル
C. コンフィグファイル
D. リソースファイル

02.オブジェクトを選択するために指定されるラベルに関する条件をなんというか。
A. ラベルセレクター
B. ラベルフィルター
C. ラベルコンディション
D. ラベルクエリ

2.13 Kubernetesの基本的な使い方

確認テスト

03.オブジェクトに割り当てられたキーとバリューのペアのことをなんというか。
A. キーバリューペア
B. マップ
C. ラベル
D. アトリビュート

04.ポッドを作成するには、kubectl apply コマンドを実行するが、なんというマニフェストを使用するか。
A. podmanifest.yaml
B. deploymentmanifest.yaml
C. replicasetmanifest.yaml
D. pod.yaml

2.13 Kubernetesの基本的な使い方

確認テスト

05.レプリカセットを作成するには、kubectl apply コマンドを実行するが、なんというマニフェストを使用するか。
A. podManifest.yaml
B. deploymentManifest.yaml
C. replicaset.yaml
D. serviceManifest.yaml

06.デプロイメントを作成するには、kubectl apply コマンドを実行するが、なんというマニフェストを使用するか。
A. podmanifest.yaml
B. deploymentt.yaml
C. replicaSetmanifest.yaml
D. servicemanifest.yaml

2.13 Kubernetesの基本的な使い方

確認テスト

07.サービスを作成するには、kubectl apply コマンドを実行するが、なんというマニフェストを使用するか。
A. podmanifest.yaml
B. deploymentmanifest.yaml
C. replicasetmanifest.yaml
D. service.yaml

08.プログラムなどを利用可能な状態にする手順を自動化することをなんというか。
A. インテグレーション
B. インフラストラクチャー
C. デプロイメント
D. 自動ビルド

2.13 Kubernetesの基本的な使い方

確認テスト

09.プログラムなどをリリースする手順を自動化することをなんというか。
A. インテグレーション
B. デプロイメント
C. 自動デプロイ
D. オートメーション

10.複数のクラウドを組み合わせて構成された環境のことをなんというか。
A. マルチクラウド
B. ハイブリッドクラウド
C. オンプレミスクラウド
D. パブリッククラウド

24

2.15 AWSとは

確認テスト⑥

以下のうち、正しいものをすべて選択してください。

1. AWS ではデータセンターを本社にのみ有しており、世界で最も堅牢と言われるセキュリティのもと運用している。
2. Amazon ECS、Amazon EKSはどちらもAWSのコンテナオーケストレーションサービスである。
3. AWS はIaaSおよびPaaS市場で圧倒的なシェアを持つAmazon社のクラウドサービスで、世界最大のクラウドプロバイダーとして多彩なクラウドサービスと高い可用性を提供している。
4. AWS はサーバーレスなプログラム実行環境サービスとしてAWS Lambdaを提供している。

13

2.16 AWSの主要なサービス

確認テスト

01.AWSには多様なサービスが存在し、同じ目的のサービスが複数存在しているが、それぞれ異なるニーズや要件に対応している。サービス選択時に考慮すべき要素として適切でないものはどれか。

A.パフォーマンス要件：必要な処理能力やスケーラビリティを満たせるか

B.コスト：予算に合わせたサービス選択が可能か

C.セキュリティとコンプライアンス：業界の規制やセキュリティ要件に適合しているか

D.サービス知名度：どれだけ多くのユーザーが利用しており、共通認識のもと作業できるか

8

2.17 コンテナを利用したAWSアーキテクチャ

確認テスト⑦

Well-Architectedフレームワークの6つの柱について説明した以下の文章について、
それぞれ正しいか正しくないかを答えてください。

1つ目の柱：「運用上の優秀性」では、誰でも簡単に運用できるサービスを提供することが目的

2つ目の柱：「セキュリティ」では、データとシステムのセキュリティを保護することが目的

3つ目の柱：「信頼性」では、システムが意図したとおりに機能し続ける能力を確保することが目的

4つ目の柱：「パフォーマンス効率」ではリソースを最適に利用してパフォーマンスを最大化することが目的

5つ目の柱：「コスト最適化」では、全体を俯瞰しながら少しでも多くコストを削減することが目的

6つ目の柱：「持続可能性」では、どのような状況下であってもサービス提供を持続させることが目的

9

2.17 コンテナを利用したAWSアーキテクチャ

確認テスト

01.Well-Architectedフレームワークの6つの柱について適切ではないものはどれか。
A.セキュリティ
B.柔軟性
C.持続可能性
D.運用上の優秀性

02.Well-Architected フレームワークの1つ目の柱である「運用上の優秀性」について、適切ではないことはどれか。
A.運用をコードとして実行する
B.大規模かつ可逆的な変更を慎重に行う
C.運用手順を頻繁に改善する
D.障害を予想する

10

2.17 コンテナを利用したAWSアーキテクチャ

確認テスト

03.Well-Architected フレームワークの2つ目の柱である「セキュリティ」について、適切ではないことはどれか。
A.強力なアイデンティティ基盤を実装する
B.一部の重要なレイヤーでセキュリティを適用する
C.セキュリティのベストプラクティスを自動化する
D.伝送中および保管中のデータを保護する

04.Well-Architected フレームワークの3つ目の柱である「信頼性」について、適切ではないことはどれか。
A.障害からは必ず手動で復旧する
B.復旧手順をテストする
C.水平方向にスケールして集合的なワークロードの可用性を向上する
D.キャパシティーを勘に頼らない

11

2.17 コンテナを利用したAWSアーキテクチャ

確認テスト

05.Well-Architected フレームワークの4つ目の柱である「パフォーマンス効率」について、適切ではないことはどれか。
A.システムに対する精通の程度を考慮する
B.わずか数分でグローバル展開する
C.サーバーレスアーキテクチャを使用する
D.慎重に少ない回数で実験する

06.Well-Architected フレームワークの5つ目の柱である「コスト最適化」について、適切ではないことはどれか。
A.クラウド財務管理を実装する
B.消費モデルを導入する
C.全体的な効率は測定せず、部分的な効率化を積み重ねて全体を効率化する
D.差別化につながらない高負荷の作業に費用をかけるのをやめる

12

2.17 コンテナを利用したAWSアーキテクチャ

確認テスト

07.Well-Architected フレームワークの6つ目の柱である「持続可能性」について、適切ではないことはどれか。

A.影響を理解する

B.持続可能性の目標を設定する

C.使用率は最小限にする

D.より効率的なハードウェアやソフトウェアの新製品を予測して採用する

13

2.18 AWSの使い方①

確認テスト

01.AWSリソースのプロビジョニングと管理を自動化するためのサービスはなにか。
A. AWS CloudFormation
B. AWS Lambda
C. AWS Elastic Beanstalk
D. AWS IAM

02.CloudFormationの主な特徴について適切でないものはどれか。
A.テンプレートベースの構成：リソースの構成情報をJSONまたはYAML形式のテンプレートファイルに記述
B.自動化と一貫性：テンプレートを使用してリソースをデプロイすることで、迅速なデプロイメントが実現
C.簡単なアップデートとロールバック：テンプレートを更新するだけでアップデートやロールバックが可能
D.リソースの依存関係の管理：リソース間の依存関係を処理し、指定した順序で手動で作成および削除する

2.18 AWSの使い方①

確認テスト

03.AWSが提供しているクラウドベースの統合開発環境（IDE）をなんというか。
A. AWS Cloud9
B. AWS CodeDeploy
C. AWS CodeCommit
D. AWS CodePipeline

04.AWS Cloud9の主な特徴について適切でないものはどれか。
A.クラウドベース：インストールが不要で、インターネットに接続された任意のブラウザからアクセス可能
B.組み込みのAWSツールサポート：AWSリソースへの簡単なアクセスと管理が可能
C.間接的なコード実行とデバッグ：コードを直接ではなく間接的に実行することで、ステップごとにデバッグすることが可能
D.サポート言語とツール：JavaScript、Python、PHPなど複数のプログラミング言語に対応

2.19 AWSの使い方②

確認テスト⑧

AWSの使い方について説明した以下の文章について、正しいものを全て選んでください。

1.疎通確認はネットワーク上でデバイス間やシステム間の通信が正しく行われているか確認するプロセス。

① AWS の東京リージョンに新規VPCを作成（マルチAZ構成が前提）

②AZ（ Availability Zone ） ごとに「パブリックサブネット」と「プライベートサブネット」を作成

③インターネットゲートウェイを作成

④ルートテーブルを設定

⑤セキュリティグループを設定

⑥CloudFormationを利用してリソースを作成

2.19 AWSの使い方②

確認テスト

01.仮想化の説明として正しいものはどれか？
A. 実体のないものをあたかも実在しているかのごとく表現する技術のこと
B. インターネット上でのバーチャルな世界を構築する技術のこと
C. データの暗号化を行う技術のこと
D. 実際のハードウェアに影響を与える仮想のエージェントを作成する技術

02.仮想化技術の主な目的は何か？
A. ハードウェアリソースの効率的な利用
B. ソフトウェアの開発速度の向上
C. インターネットの速度向上
D. データのセキュリティ向上

2.20 AWSの使い方③

演習②

演習課題①ではDockerで立ち上げたDBを使用しましたが、
演習課題②ではAWS RDS上にDBインスタンスを立ち上げて、ローカルから接続してください。

3.1 演習課題①で起動したコンテナのうち、DBのコンテナを終了させてください

3.2 その状態でその他のコンテナを起動させて、WordPressがエラーになることを確認してください

3.3 AWS RDS上にDBを立ち上げてください

* 適切にセキュリティグループ等を設定してください

* インターネット経由で接続できるように設定してください

3.4 3.3で立ち上げたDBに、WordPressからアクセスするように設定してください

3.5 設定後、WordPressが表示されることを確認してください

7

令和５年度「専門職業人材の最新技能アップデートのための専修学校リカレント教育推進事業」
情報技術者の技能アップデートのためのリカレント教育推進事業

コンテナ技術基礎教材資料

令和６年３月

一般社団法人全国専門学校情報教育協会
〒164-0003　東京都中野区東中野 1-57-8　辻沢ビル３F
電話：03-5332-5081　FAX.03-5332-5083