【Docker】全容

目次

Docker

■はじめに

- ▶ ※ Dockerは、コンテナ型の仮想環境を作成、配布、実行するためのソフトである。
- ▶ ※ コンテナも「仮想マシンのようにOSを再現している」ような認識をしてしまいがちだが、OS機能はホストOSに依存しており、「アプリケーションの動作を再現しているだけ」という認識が重要。
- ▶ ※ 仮想マシンはOSを実行することが目的であるいっぽう、コンテナは Root Process と呼ばれるプロセスを実行することが目的。
- ▶ ☆ インストール
- ▶ バージョン確認
- ▶ ☆ コンテナのライフサイクル
- ▶ ※ コンテナは永続的にデータを保持せず、コンテナの破棄と同時にコンテナ内のデータは全て失われる。ただ、データを永続化させる方法が2つある。
- ▶ ☆ データ永続化の2つの方法
- ▶ ☆ `docker container` `docker image` コマンド体系

■Dockerfileについて

- ▶ ※ 公式ドキュメントはコチラかコチラ（どちらにも未翻訳の部分があるので両方で補完しあおう）。
- ▶ ☆ 効率的なDockerfileの作成手順
- ▶ コメント
- ▶ ※ `EXPOSE` 命令はコンテナが特定のポートを実行時にリッスンすることを Docker Engine に伝えるためだけのもので、これだけでホストからコンテナにアクセスできることを意味しない。そして、どのポートに接続するのか親切に明示するために書いておくのがよい。
- ▶ ☆ コンテナを勝手に終了させないようにしたい

【Docker】全容

目次

- Docker
- Docker Compose
- Docker Swarm

Docker

■はじめに

- ▶ ※ Dockerは、コンテナ型の仮想環境を作成、配布、実行するためのソフトである。
- ▶ ※ コンテナも「仮想マシンのようにOSを再現している」ような認識をしてしまいがちだが、OS機能はホストOSに依存しており、「アプリケーションの動作を再現しているだけ」という認識が重要。
- ▶ ※ 仮想マシンはOSを実行することが目的であるいっぽう、コンテナは Root Process と呼ばれるプロセスを実行することが目的。
- ▶ ☆ インストール
- ▶ バージョン確認 $ docker --version
- ▶ ☆ コンテナのライフサイクル
- ▶ ※ コンテナは永続的にデータを保持せず、コンテナの破棄と同時にコンテナ内のデータは全て失われる。ただ、データを永続化させる方法が2つある。
- ▶ ☆ データ永続化の2つの方法
- ▶ ☆ `docker container` `docker image` コマンド体系

■Dockerfileについて

- ▶ ※ 公式ドキュメントはコチラかコチラ（どちらにも未翻訳の部分があるので両方で補完しあおう）。
- ▶ ☆ 効率的なDockerfileの作成手順
- ▶ コメント #で始まる行はコメントになる
- ▶ ※ `EXPOSE` 命令はコンテナが特定のポートを実行時にリッスンすることを Docker Engine に伝えるためだけのもので、これだけでホストからコンテナにアクセスできることを意味しない。そして、どのポートに接続するのか親切に明示するために書いておくのがよい。
- ▶ ☆ コンテナを勝手に終了させないようにしたい

▶ ※ `COPY ./ ./` は先頭で行わないようにしよう 。初めにやるとキャッシュを活かせない。

■コンテナイメージの管理

▶ 今あるイメージの一覧

▶ イメージの詳細を確認

▶ イメージを取得

▶ Dockerfileから作成

▶ イメージを削除

▶ ☆ `<none>:<none>` のイメージを削除

■コンテナ（のインスタンス）の管理

▶ ※ 以下の *`container`* にはコンテナー覧より確認できる `CONTAINER ID` または `NAMES` の値を入れる。なお、 `CONTAINER ID` の場合、IDの一字一句を書ききる必要はなく、一意に定まるような前方部分のみ書けばOK。

情報の取得

▶ 作成済みコンテナー覧

▶ コンテナの詳細を確認

▶ IPアドレスを確認

▶ コンテナのログを表示

▶ 最後から*n*行のみ〃

▶ コンテナのOSを確認

新規作成して起動

▶ 新しくコンテナを起動

▶ バックグラウンドで〃

▶ ポート転送して〃

▶ 名前をつけて〃

▶ 〃して終了時に自動削除

▶ 〃してCLIで操作可能に

▶ volume でデータ永続化して〃

▶ ※ `COPY ./ ./` は先頭で行わないようにしよう 。初めにやるとキャッシュを活かせない。

■コンテナイメージの管理

▶ 今あるイメージの一覧　$ docker image ls　（か $ docker images ）

▶ イメージの詳細を確認　$ docker image inspect *imageID*

▶ イメージを取得　$ docker image pull *image:tag*

▶ Dockerfileから作成　$ docker image build ※1 *context*※2　※1ここにオプを
※2 Dockerfile内で記述されている `./` が指す場所のこと。

▶ イメージを削除　$ docker image rm *imageID*　（か $ docker rmi *imageID* ）

▶ ☆ `<none>:<none>` のイメージを削除

■コンテナ（のインスタンス）の管理

▶ ※ 以下の *`container`* にはコンテナー覧より確認できる `CONTAINER ID` または `NAMES` の値を入れる。なお、 `CONTAINER ID` の場合、IDの一字一句を書ききる必要はなく、一意に定まるような前方部分のみ書けばOK。

情報の取得

▶ 作成済みコンテナー覧　・起動中ののみ　$ docker container ls　（か $ docker ps ）
・停止中の含め　$ docker container ls -a

▶ コンテナの詳細を確認　$ docker container inspect *container*

▶ IPアドレスを確認　$ docker container inspect *container* | grep IPAddress

▶ コンテナのログを表示　$ docker container logs ※1 *container*　※1ここにオプを

▶ 最後から*n*行のみ〃　$ ‥ -n *n* ‥

▶ コンテナのOSを確認　コンテナに接続のうえで　$ cat /etc/*-release

新規作成して起動

▶ 新しくコンテナを起動　$ docker container run ※1 *image:tag*　※1ここにオプを

▶ バックグラウンドで〃　$ ‥ -d ‥　：デタッチモードで起動する

▶ ポート転送して〃　$ ‥ -p *machinePort:containerPort* ‥

▶ 名前をつけて〃　$ ‥ --name *name* ‥

▶ 〃して終了時に自動削除　$ ‥ --rm ‥

▶ 〃してCLIで操作可能に　$ ‥ -it ‥

▶ volume でデータ永続化して〃　$ ‥ -v *namedVol:mountPoint* ‥　※ *`namedVol`* は未存でも可
$ ‥ --mount type=volume, src=*namedVol*, dst=*mP* ‥　※〃

- ▶ bind mount でデータ永続化して〃
- ▶ あるﾈｯﾄﾜｰｸに接続して〃
- ▶ 〃して同時にコマ実行

接続

- ▶ コンテナに接続
- ▶ 接続状態から抜ける

停止、起動、再起動、削除

- ▶ 起動中コンテナを停止
- ▶ 停止中コンテナを起動
- ▶ 起動中コンテナを再起動
- ▶ コンテナを削除

■ボリュームの管理

- ▶ ※ ボリューム（データ・ボリューム）とは、1つまたは複数のコンテナ内で、特別に設計されたディレクトリ。データ保持のために設計されており、コンテナのライフサイクルとは独立している。
- ▶ ※ ただ、コンテナ作成時に未存のボリュームが指定された場合、自動的に作成される。
- ▶ ボリューム一覧
- ▶ ボリュームの詳細を確認
- ▶ ボリュームを新規作成
- ▶ ※ デフォルトでは、**管理ディレクトリ**にボリュームが作られる。
- ▶ ボリュームを削除

■ネットワークの管理

- ▶ ネットワーク一覧
- ▶ ※ 最初から `bridge` `host` `none` という3つのネットワークが存在している。
- ▶ ※ コンテナを起動するとデフォルトでは `bridge` に接続される。
- ▶ ※ `host` は、Dockerを起動しているパソコン環境に対して直接接続することを可能にするネットワーク。
- ▶ ※ `none` はネットワークとして完全に遮断された環境を実現する。
- ▶ ネットワークの詳細確認

- ▶ bind mount でデータ永続化して〃 $ ･･ -v *pathOnMachine:mountPoint* ･･
  $ ･･ --mount type=**bind**, src=*pathOnMachine*, dst=*mP* ･･
- ▶ あるﾈｯﾄﾜｰｸに接続して〃 $ ･･ --net *network* ･･ ※無ければ bridge に接続される
- ▶ 〃して同時にコマ実行 $ ････ *command arg1 arg2 ...*

接続

- ▶ コンテナに接続 $ docker container exec -it *container* sh
- ▶ 接続状態から抜ける {Ctrl}+{P} → {Ctrl}+{Q} ※ {Ctrl}+{C} や `$ exit` はダメ

停止、起動、再起動、削除

- ▶ 起動中コンテナを停止 $ docker container stop *container*
- ▶ 停止中コンテナを起動 $ docker continer start *container*
- ▶ 起動中コンテナを再起動 $ docker container restart *container*
- ▶ コンテナを削除 $ docker container rm *container*

■ボリュームの管理

- ▶ ※ ボリューム（データ・ボリューム）とは、1つまたは複数のコンテナ内で、特別に設計されたディレクトリ。データ保持のために設計されており、コンテナのライフサイクルとは独立している。
- ▶ ※ ただ、コンテナ作成時に未存のボリュームが指定された場合、自動的に作成される。
- ▶ ボリューム一覧 $ docker volume ls
- ▶ ボリュームの詳細を確認 $ docker volume inspect *volume1 volume2 ...*
- ▶ ボリュームを新規作成 $ docker volume create *volumeName*
- ▶ ※ デフォルトでは、**管理ディレクトリ**にボリュームが作られる。
- ▶ ボリュームを削除 $ docker volume rm *volume1 volume2 ...*

■ネットワークの管理

- ▶ ネットワーク一覧 $ docker network ls
- ▶ ※ 最初から `bridge` `host` `none` という3つのネットワークが存在している。
- ▶ ※ コンテナを起動するとデフォルトでは `bridge` に接続される。
- ▶ ※ `host` は、Dockerを起動しているパソコン環境に対して直接接続することを可能にするネットワーク。
- ▶ ※ `none` はネットワークとして完全に遮断された環境を実現する。
- ▶ ネットワークの詳細確認 $ docker network inspect *network1 network2 ...*

- ネットワークを新規作成
- ※ 同じネットワークにあるコンテナ同士は通信ができる。しかも、IPアドレスの代わりにコンテナ名で通信相手を指定できる。

Docker Compose

■はじめに

- ※ Docker Compose は、複数のDockerコンテナを効率的に操作するためのツール。
- ※ Docker Composeを利用するコマンドとして、`docker compose` と `docker-compose` の２つがあるが、前者を使うべきである。
- ☆ インストール

■複数のDockerコンテナを構築する手順

- １．プロジェクトディレクトリへ移動
- ※ デフォルトではこのディレクトリの名前がそのままプロジェクト名になる。
- ２．プロディにcomposeファを作成
- ３．コンテナ群を構築して起動
- ※ コンテナ群を起動しているあいだはcomposeファを編集するべきでない。
- Ｆ．コンテナ群を一括停止して削除

■composeファイルについて

- ※ 公式ドキュメントはコチラ、またはコチラ (英語) 。
- ※ composeファイルとしてカスタムな名前のものを使用する、あるいは複数のcomposeファイルを使用する場合、`docker compose -f fileName1 -f fileName2 … ‥` とする。ただし、`docker-compose.yml` (一般的) 、`docker-compose.yaml` 、`compose.yaml` (公式が推奨) 、`compose.yml` のいずれかの名前のファイルのみがカレントディレクトリにある場合は、`-f` オプションなしで `docker compose` コマンドが使える。
- ☆ composeファイルの大まかな構成
- ※ **composeファイルや .env ファイルで `$` を使う場合は注意が必要。**
- ☆ 複数のcomposeファイルの統合結果をYAML形式で表示
- ☆ 本番環境向けのcomposeファイルについて

■composeプロジェクト（コンテナ群）の管理

- ※ `docker compose` コマンドに関する公式ドキュメントはコチラ。

- ネットワークを新規作成　$ docker network create *networkName*
- ※ 同じネットワークにあるコンテナ同士は通信ができる。しかも、IPアドレスの代わりにコンテナ名で通信相手を指定できる。

Docker Compose

■はじめに

- ※ Docker Compose は、複数のDockerコンテナを効率的に操作するためのツール。
- ※ Docker Composeを利用するコマンドとして、`docker compose` と `docker-compose` の２つがあるが、前者を使うべきである。
- ☆ インストール

■複数のDockerコンテナを構築する手順

- １．プロジェクトディレクトリへ移動　$ mkdir *dirName*　　$ cd *dir*
- ※ デフォルトではこのディレクトリの名前がそのままプロジェクト名になる。
- ２．プロディにcomposeファを作成　　$ vi compose.yaml※[1]　　※[1]他の名前も可能
- ３．コンテナ群を構築して起動　　$ docker compose up -d　※デタッチモード
- ※ コンテナ群を起動しているあいだはcomposeファを編集するべきでない。
- Ｆ．コンテナ群を一括停止して削除　　$ docker compose down　(当然プロディにて)

■composeファイルについて

- ※ 公式ドキュメントはコチラ、またはコチラ (英語) 。
- ※ composeファイルとしてカスタムな名前のものを使用する、あるいは複数のcomposeファイルを使用する場合、`docker compose -f fileName1 -f fileName2 … ‥` とする。ただし、`docker-compose.yml` (一般的) 、`docker-compose.yaml` 、`compose.yaml` (公式が推奨) 、`compose.yml` のいずれかの名前のファイルのみがカレントディレクトリにある場合は、`-f` オプションなしで `docker compose` コマンドが使える。
- ☆ composeファイルの大まかな構成
- ※ **composeファイルや .env ファイルで `$` を使う場合は注意が必要。**
- ☆ 複数のcomposeファイルの統合結果をYAML形式で表示
- ☆ 本番環境向けのcomposeファイルについて

■composeプロジェクト（コンテナ群）の管理

- ※ `docker compose` コマンドに関する公式ドキュメントはコチラ。

- ▶ 実行中のプロジェ一覧
- ▶ 停止中のコンテナの情報も含めて〃
- ▶ ※ 以下のコマンドにおいて、composeファイルとしてカスタムな名前のものしている場合は必ず `-f` オプションを使うこと。

プロジェクトの起動（コンテナ群の起動）

- ▶ コンテナ群を構築して起動
- ▶ バックグラウンドで〃
- ▶ イメージを再ビルドしたうえで〃

プロジェクトの終了（コンテナ群の停止および削除）

- ▶ コンテナ群を一括停止して削除
- ▶ volumes要素に書いたボリュも削除
- ▶ イメージ (カスタム, タグなし) も削除

■サービスの管理

- ▶ サービスのコンテナに接続

- ▶ 実行中のプロジェ一覧　$ docker compose ls
- ▶ 停止中のコンテナの情報も含めて〃　$ docker compose ls -a
- ▶ ※ 以下のコマンドにおいて、composeファイルとしてカスタムな名前のものしている場合は必ず `-f` オプションを使うこと。

プロジェクトの起動（コンテナ群の起動）

- ▶ コンテナ群を構築して起動　$ docker compose up ※1　※1ここにオプを
- ▶ バックグラウンドで〃　$ ‥ -d　：デタッチモードで起動する
- ▶ イメージを再ビルドしたうえで〃　$ ‥ --build

プロジェクトの終了（コンテナ群の停止および削除）

- ▶ コンテナ群を一括停止して削除　$ docker compose down ※1　※1ここにオプを
- ▶ volumes要素に書いたボリュも削除　$ ‥ -v
- ▶ イメージ (カスタム, タグなし) も削除　$ ‥ --rmi local

■サービスの管理

- ▶ サービスのコンテナに接続　$ docker compose exec *service* sh　※ `-it` 不要

Docker Swarm

■はじめに

- ▶ ※ Docker Swarm はDockerネイティブなコンテナオーケストレーションツール。
- ▶ ※ インストールについて、Dockerをインストールすればdocker Swarmも使えるようになる。確認として `$ docker swarm help` すればよい。

■概念

- ▶ ☆ ノード
- ▶ ☆ マネージャノード
- ▶ ☆ ワーカノード

- ▶ ☆ 概念の全体像
- ▶ ☆ サービス：コンテナの集まり
- ▶ ☆ スタック：サービスの集まり

■その他の用語

Docker Swarm

■はじめに

- ▶ ※ Docker Swarm はDockerネイティブなコンテナオーケストレーションツール。
- ▶ ※ インストールについて、Dockerをインストールすればdocker Swarmも使えるようになる。確認として `$ docker swarm help` すればよい。

■概念

- ▶ ☆ ノード
- ▶ ☆ マネージャノード
- ▶ ☆ ワーカノード

- ▶ ☆ 概念の全体像
- ▶ ☆ サービス：コンテナの集まり
- ▶ ☆ スタック：サービスの集まり

■その他の用語

- ▶ Dockerホスト

■クラスタリングの手順

- ▶ ☆ デプロイの流れ
- ▶ ☆ １？．docker swarm init

■クラスタの管理

マネージャノード（になるつもりのマシン）にて

- ▶ マネージャノードとして自身を登録

ワーカノード（になるつもりのマシン）にて

- ▶ ☆ ワーカノードとして自身を登録

■ノードの管理

- ▶ クラスタ上のノード一覧

■サービスの管理

- ▶ ※ **独自のオーバーレイネットワークを用意しておこう**：（マネージャノードにて）`$ docker network create -d overlay networkName` 。クラスタ内のサービスとして登録されるコンテナは、このオーバーレイネットワークを介して互いに通信ができるようになる。
- ▶ ※ **サービスの管理はマネージャノードからしかできない**。以下はマネージャノードで行う。

情報の取得

- ▶ サービスの一覧
- ▶ サービスの詳細を確認
- ▶ 〃内で稼働中のタスク一覧

新規作成して起動

- ▶ 新しくサービスを起動
- ▶ コンテナ名を指定して〃
- ▶ ポート転送して〃
- ▶ あるﾈｯﾄﾜｰｸに接続して〃
- ▶ レプリカ数を指定して〃

その他

- ▶ Dockerホスト　Dockerがインストールされたマシン

■クラスタリングの手順

- ▶ ☆ デプロイの流れ
- ▶ ☆ １？．docker swarm init

■クラスタの管理

マネージャノード（になるつもりのマシン）にて

- ▶ マネージャノードとして自身を登録　$ docker swarm init --advertise-addr *IPAddress*※1
  ※自身のIPアドレス。複数もっている場合は1つ選ぶ。

ワーカノード（になるつもりのマシン）にて

- ▶ ☆ ワーカノードとして自身を登録

■ノードの管理

- ▶ クラスタ上のノード一覧　$ docker node ls

■サービスの管理

- ▶ ※ **独自のオーバーレイネットワークを用意しておこう**：（マネージャノードにて）`$ docker network create -d overlay networkName` 。クラスタ内のサービスとして登録されるコンテナは、このオーバーレイネットワークを介して互いに通信ができるようになる。
- ▶ ※ **サービスの管理はマネージャノードからしかできない**。以下はマネージャノードで行う。

情報の取得

- ▶ サービスの一覧　$ docker service ls
- ▶ サービスの詳細を確認　$ docker service inspect *service1 service2 ...*
- ▶ 〃内で稼働中のタスク一覧　$ docker service ps ※1 *service1 service2 ...*　※1オプ

新規作成して起動

- ▶ 新しくサービスを起動　$ docker service create ※1 *image:tag*　※1オプ
- ▶ コンテナ名を指定して〃　$ ‥ --name *name* ‥
- ▶ ポート転送して〃　$ ‥ -p *machinePort:containerPort* ‥
- ▶ あるﾈｯﾄﾜｰｸに接続して〃　$ ‥ --network *network* ‥
- ▶ レプリカ数を指定して〃　$ ‥ --replicas レプリカ数 ‥

その他

▶ 設定を変更して再起動

▶ サービスを削除

■スタックの管理

▶ ※ コマンドは `docker stack ～` を用いる。

■シークレットの管理

▶ ※ コマンドは `docker secret ～` を用いる。

▶ 設定を変更して再起動　$ docker service update ※[1] *image:tag*　※[1]ここにオプを

▶ サービスを削除　$ docker service rm *service1 service2 ...*

■スタックの管理

▶ ※ コマンドは `docker stack ～` を用いる。

■シークレットの管理

▶ ※ コマンドは `docker secret ～` を用いる。