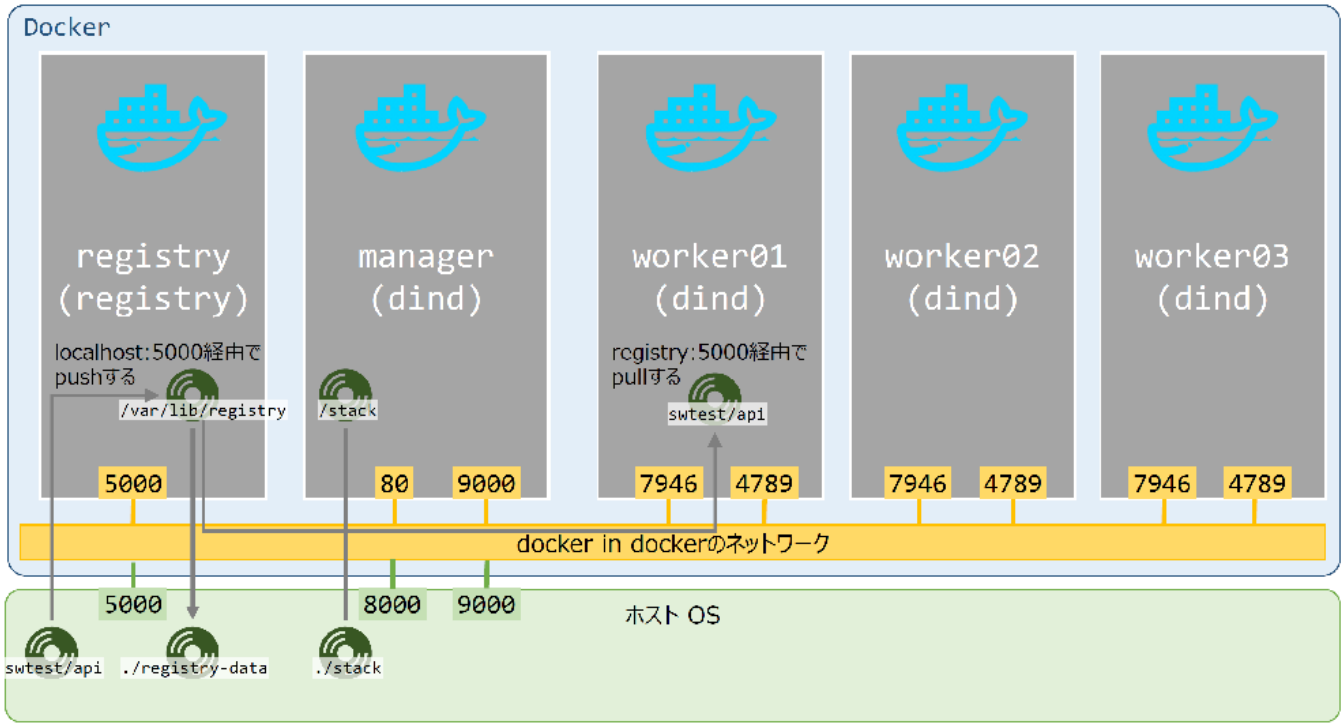
Docker勉強会(3回目) SWARMによるクラスタ体験

今回は、Docke-swarm を使用して複数コンテナをスケールアウトしクラスタ運用する体験をします。

体験環境について

通常は複数のサーバーを束ねてクラスタ運用しますが、今回の体験ではDockerコンテナを1台のサーバーに見立てて、仮想的に複数サーバーを建て、クラスタ運用を行います。

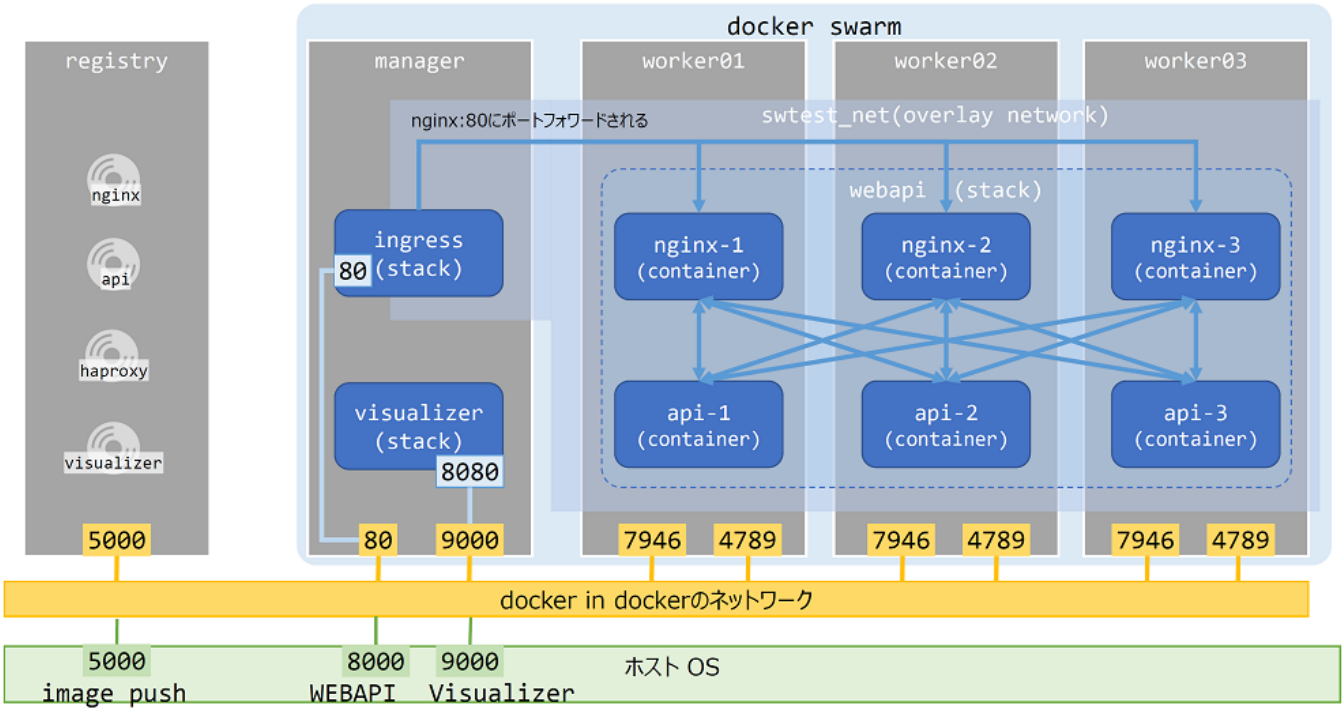
仮想サーバーのコンテナは、「Dockerをホストとして機能するDockerコンテナ」を **「Docker in docker」** という仕組みで構築します。



- `registry:` ローカルでコンテナイメージを保存／供給するサーバー
- `manager:` docker swarm の管理サーバー
- `worker01~03:` アプリを載せたコンテナを実行するサーバー

本日のゴール

最終的にはプロキシサーバーとAPIで構成されたアプリを3セット立ち上げて、それらの状態をWEBページで確認でき、また、外部(ホストOS)からコンテナ内部のAPIサービスを利用できる環境を作ります。



- registryサーバーには今回使用するコンテナイメージ4つを格納する
- managerでswarmを起動し、そのクラスタにworker01～03を組み込む
- swarm内でコンテナ同士が通信できるよう、overlay networkを作る
- nginx<->apiのアプリを3セットまとめて(stack化して)swarmにデプロイする
- swarmの状態をビジュアルに表示できるWEBアプリ Visualizerをデプロイする
- 外部からのアクセスを内部のアプリに振り分けるためのingressを構築する

コンテナイメージのビルド

ローカルのレジストリサーバーに保管するイメージをあらかじめビルドしておく

1. apiイメージの作成

```
cd 作業ディレクトリ\api

docker image build -t swtest/api:latest .
```

2. nginxイメージの作成

Dockerfile

```
FROM nginx:1.16

ENV http_proxy 'http://gwproxy.daikin.co.jp:3128'           # proxy設定
ENV https_proxy 'http://gwproxy.daikin.co.jp:3128'
ENV no_proxy '127.0.0.1,localhost'
```

```
RUN apt-get update
RUN apt-get install -y wget
RUN wget --no-check-certificate
https://github.com/progrium/entrykit/releases/download/v0.4.0/entrykit_0.4.0_linux
_x86_64.tgz
RUN tar -xvzf entrykit_0.4.0_linux_x86_64.tgz
RUN rm entrykit_0.4.0_linux_x86_64.tgz
RUN mv entrykit /usr/local/bin/
RUN entrykit --symlink

RUN rm /etc/nginx/conf.d/*
COPY etc/nginx/nginx.conf.tmpl /etc/nginx/
COPY etc/nginx/conf.d/ /etc/nginx/conf.d/

ENTRYPOINT [ \
  "render", \
      "/etc/nginx/nginx.conf", \            # nginx.conf.tmplを元に環境変数を置き換えて
nginx.confを作成する
      "--", \
  "render", \
      "/etc/nginx/conf.d/upstream.conf", \
      "--", \
  "render", \
      "/etc/nginx/conf.d/public.conf", \
      "--" \
]

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

entrykit　を利用して、実行時の環境変数をnginxの設定ファイルへ展開するようにしている

```
cd 作業ディレクトリ\api

docker image build -t swtest/nginx:latest .
```

3. visualizerイメージを取得

```
docker pull dockersamples/visualizer
```

4. haproxyイメージ取得

```
docker pull dockercloud/haproxy
```

swarmクラスタ環境の構築

1. docker-compose 内容確認

作業ディレクトリ\docker-compose.yml

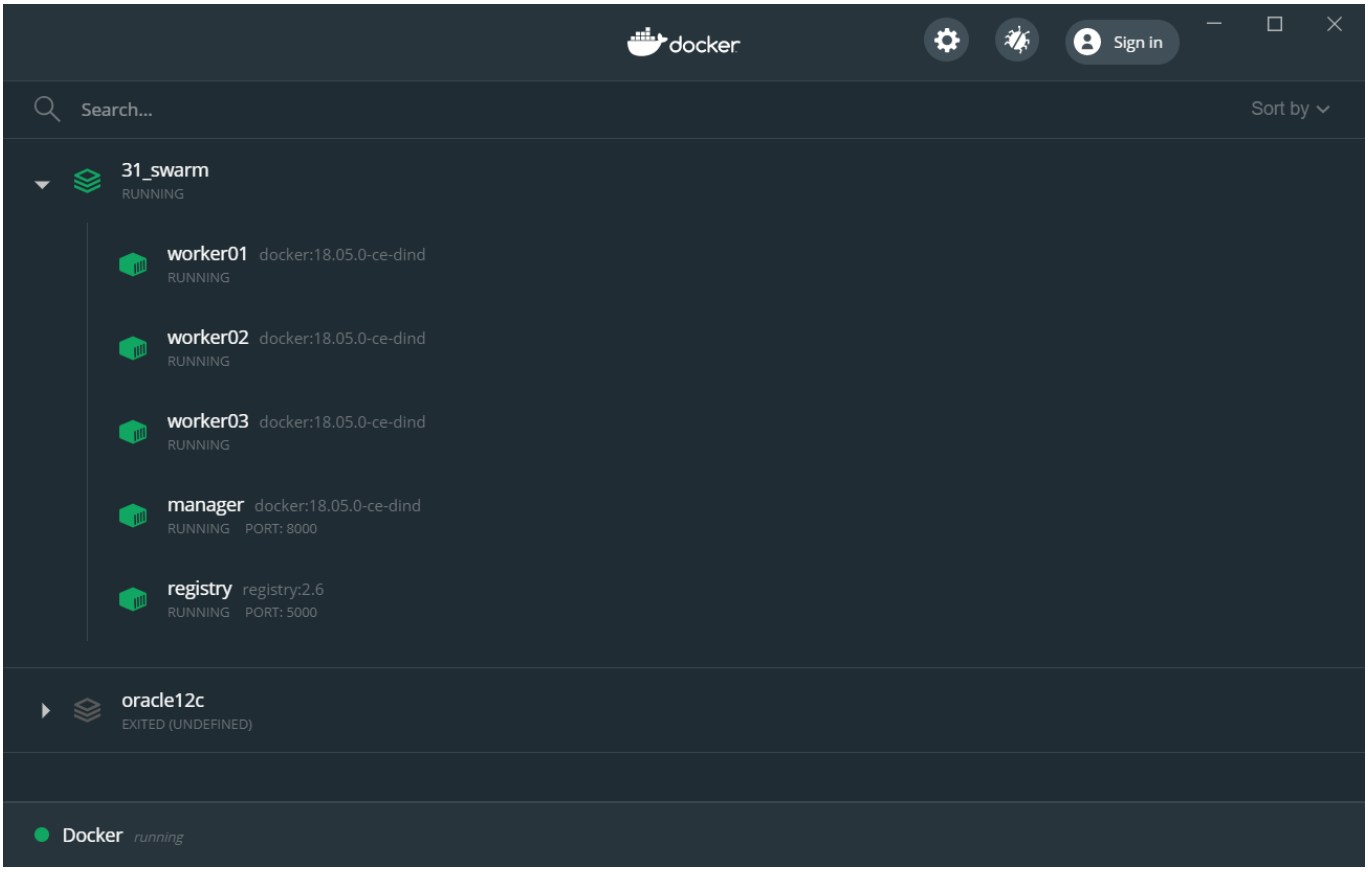
```
version: "3"
services:
    registry:
        container_name: registry
        image: registry:2.6                                 # registryというベースイメージを
使う
        ports:
            - 5000:5000                                     # ホスト側5000番  docker内部
5000番を割り当てる
        volumes:
            - "./registry-data:/var/lib/registry"           # ホスト側 ./registry-data
registry側  /var/lib/registr を共有する
                                                            # ローカルregistryのデータをホス
トOS側に永続化するため
    manager:
        container_name: manager
        image: docker:18.05.0-ce-dind                       # dockerイメージを使う(docker
in docker)
        privileged: true
        ports:
            - 8000:80                                       # ホスト側8000番  docker内部
80番を割り当てる -> webapi アクセス用
            - 9000:9000                                     # ホスト側9000番  docker内部
9000番を割り当てる -> Visualizer アクセス用
        depends_on:
            - registry
        expose:
            - 3375
        command: "--insecure-registry registry:5000"        # registry:5000へはHTTP通信
でイメージをpullできるように設定
        volumes:
            - "./stack:/stack"                              # ホスト側 ./stack  registry
側  /stak を共有する
                                                            # stackに各stackをデプロイする
ためのymlを格納する(ホスト側で編集)
    worker01:
        container_name: worker01
        image: docker:18.05.0-ce-dind                       # workerにもdockerイメージを使
う(docker in docker)
        privileged: true
        tty: true
        depends_on:
            - manager
            - registry
        expose:
            - 7946                                          # swarm通信用のポートを開けてお
く
            - 7946/udp
```

```
            - 4789/udp
        command: "--insecure-registry registry:5000"

    以下　worker02、03は名前だけ違うので省略
```

2. docker-composeでコンテナを立ち上げる

```
cd 作業ディレクトリ

docker-compose up -d
```

下図の状態で5つのコンテナが起動されます。



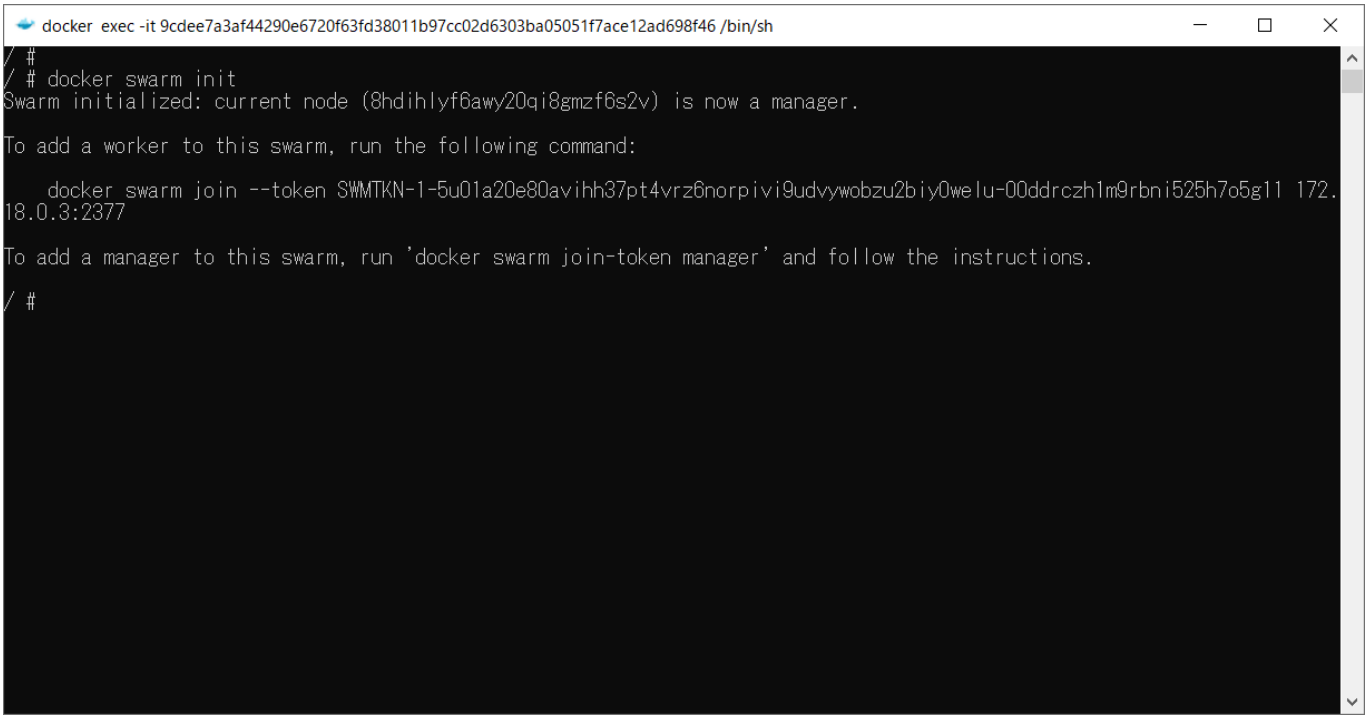
3. swarmクラスタを起動

managerコンテナのCLIを起動して、swarmクラスタを初期化します。

1) CLI起動

managerのCLIを起動します。

2) manager CLIでswarm初期化コマンドを入力




```
docker swarm init

Swarm initialized: current node (8hdihlyf6awy20qi8gmzf6s2v) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-
5u01a20e80avihh37pt4vrz6norpivi9udvywobzu2biy0welu-00ddrczh1m9rbni525h7o5g11
172.18.0.3:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow
the instructions.
```

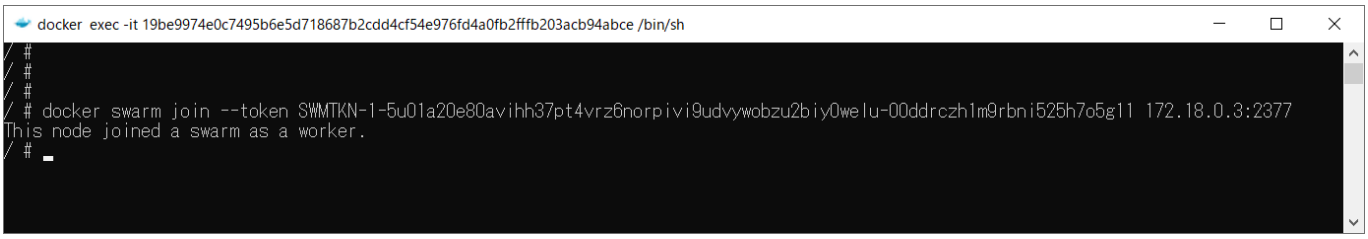
上記のような出力となります。

`「 docker swarm join --token SWMTKN-1-...5g11 172.18.0.3:2377 」`

が、workerをswarmクラスタに追加するためのコマンドになります。

各workerのCLIを立ち上げて、`docker swarm join ...` コマンドを投入します。

3) 各workerのCLIでswarm参加のコマンドを投入




```
docker swarm join --token SWMTKN-1-...5g11 172.18.0.3:2377

This node joined a swarm as a worker.
```

全workerに同じコマンドを投入します。

4) manager CLIで各nodeの状態を確認

```
docker node ls

ID                            HOSTNAME            STATUS              AVAILABILITY
MANAGER STATUS      ENGINE VERSION
8hdihlyf6awy20qi8gmzf6s2v *   9cdee7a3af44        Ready               Active
Leader              18.05.0-ce
8o6onmjvagenxlmy9csgm5gz5     19be9974e0c7        Ready               Active
18.05.0-ce
rtij47tsox0fimy4f2bmagjvq     6960c0f9f754        Ready               Active
18.05.0-ce
n0fmlmzdh3135z3ddortlx52k     d55c16d6f52e        Ready               Active
18.05.0-ce
```

4台のノードが稼働しており、1台がLeaderとなっていることが分かります。

コンテナイメージをローカルのレジストリサーバーに格納する

---

1. コンテナイメージにタグ名を付ける

ホストOSのコマンドラインから、コンテナイメージに、各ノードから参照する際のタグ名を付けます。

@ホストOS

```
docker image tag swtest/api:latest localhost:5000/swtest/api:latest
docker image tag swtest/nginx:latest localhost:5000/swtest/nginx:latest
docker image tag dockersamples/visualizer:latest localhost:5000/visualizer:latest
docker image tag dockercloud/haproxy:latest localhost:5000/haproxy:latest
```

2. ローカルレジストリにimageをpushする

ホストOSのコマンドラインから、コンテナイメージをレジストリサーバーに格納します。

@ホストOS

```
docker image push localhost:5000/swtest/api:latest
docker image push localhost:5000/swtest/nginx:latest
docker image push localhost:5000/visualizer:latest
docker image push localhost:5000/haproxy:latest
```

3. ローカルレジストリからコンテナイメージをpull出来るかテスト

worker01から、regsitry:5000経由でコンテナイメージを取得できるかテストします。

worker01のCLIを起動して、下記コマンドを入力します。

@worker01

```
docker image pull registry:5000/swtest/api:latest

latest: Pulling from swtest/api
e7c96db7181b: Pull complete
f910a506b6cb: Pull complete
c2274a1a0e27: Pull complete
f5dd5aa1c311: Pull complete
Digest: sha256:3d6261dbf843ab4d4dec65ad5a6546660f7b889fa7d5436c25523c5c5f2da5a8
Status: Downloaded newer image for registry:5000/swtest/api:latest
```

Service概念の確認

swarmでは一つのコンテナを複数個まとめて起動し管理することが出来ます。これにより同じコンテナのスケールアウトが可能になります。

1. サービス作成@manager

managerのCLIで下記コマンドを投入し、apiサービスを作成します。

@manager

```
docker service create --replicas 1 --publish 8000:8080 --name api
registry:5000/swtest/api:latest

g8lnlh1ylqt0fenxnv1jaqhe8
overall progress: 1 out of 1 tasks
```

```
1/1: running   [==================================================>]
verify: Service converged
```

2. サービス確認

@manager

```
docker service ls

ID                  NAME                MODE                REPLICAS
IMAGE                                   PORTS
g8lnlh1ylqt0        api                 replicated          1/1
registry:5000/swtest/api:latest   *:8000->8080/tcp
```

サービスとして「api」が 1 つ起動していることが分かります。

3. サービスのスケールアウト

先ほどのサービスをスケールアウトし 6 つまで起動します。

@manager

```
docker service scale api=6

api scaled to 6
overall progress: 6 out of 6 tasks
1/6: running   [==================================================>]
2/6: running   [==================================================>]
3/6: running   [==================================================>]
4/6: running   [==================================================>]
5/6: running   [==================================================>]
6/6: running   [==================================================>]
verify: Service converged
```

実行状況を確認します。

@manager

```
docker service ps api

ID                  NAME                IMAGE                                   NODE
DESIRED STATE       CURRENT STATE                ERROR                PORTS
46f9mpevo3f7        api.1               registry:5000/swtest/api:latest
9cdee7a3af44        Running             Running 6 minutes ago
ms1si10zr0j4        api.2               registry:5000/swtest/api:latest
19be9974e0c7        Running             Running about a minute ago
n3ex8052eamg        api.3               registry:5000/swtest/api:latest
```

```
6960c0f9f754        Running             Running about a minute ago
83601xd8g72n        api.4               registry:5000/swtest/api:latest
d55c16d6f52e        Running             Running about a minute ago
pwor251haceu        api.5               registry:5000/swtest/api:latest
d55c16d6f52e        Running             Running about a minute ago
xj12h7mm8j5t        api.6               registry:5000/swtest/api:latest
9cdee7a3af44        Running             Running about a minute ago
```

ここで、注目するのは、NODEの項目です。

６つのプロセスが４つのノード(manager,worker01~03)にバランスを見ながら自動で振り分けられています。

4. サービスの終了と削除

起動したサービスを終了しコンテナを削除します。

@manager

```
docker service rm api

api

docker service ls

ID                  NAME                MODE                REPLICAS
IMAGE               PORTS
```

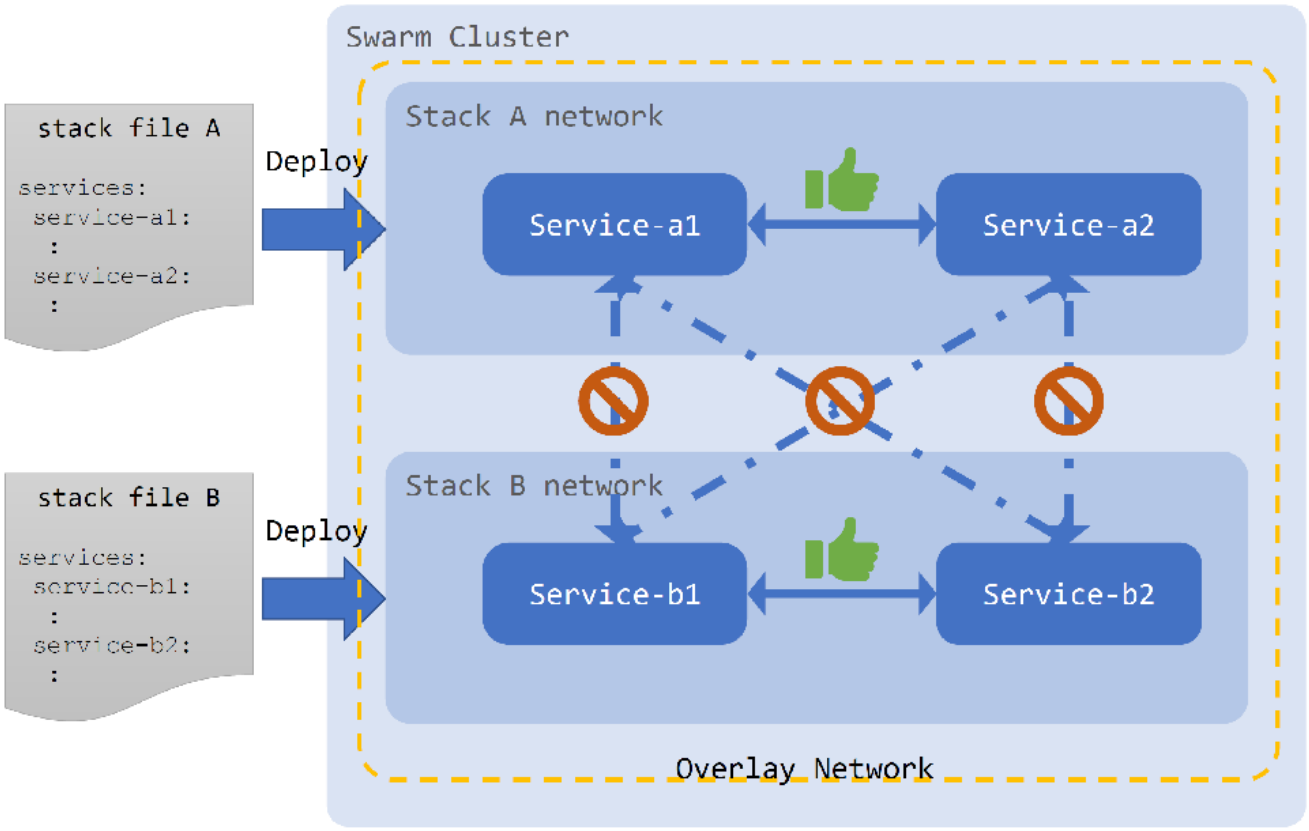
サービス削除を確認できました。

---

stackを作りアプリ(nginx<->api)をswarm内で起動する

Stackとは複数のServiceをグルーピングした単位で、アプリケーション全体の構成を定義します。

```
container-1 -> service-1 ┐
                         ├-> stack
container-2 -> service-2 ┘
```



上図のように、通常Stackをswarmクラスタにデプロイしただけでは、それぞれのStack用のネットワークを構成し、異なるのStackのServiceとは通信できません。

複数のStack間で通信を行いたい場合は、それぞれのStackを包含する `Overlay Network` をswamクラスタ内に作成し、各StackのServiceを所属させます。

1. オーバーレイネットワークの作成

manager CLIにてoverlay networkを作成します。

`@manager`

```
docker network create --driver=overlay --attachable swtest_net
```

2. Stackの確認

まず、stack構成ファイルを確認します。

`作業ディレクトリ\stack\webapi.yml`

```
version: "3"
services:
    nginx:
        image: registry:5000/swtest/nginx:latest        # ローカルレジストリのイメージから作成
        deploy:
```

```
            replicas: 3                                     # 起動数=3
            placement:
                constraints: [node.role != manager]     # manager以外のnodeで起動する
        environment:
            SERVICE_PORTS: 80                           # 内部的には80ポートを公開
            BACKEND_HOST: swtest_api:8080               # リバースプロキシ先は
swtest_apiの8080ポート
        depends_on:
            - api
        networks:
            - swtest_net                                # swtest_net というオーバーレイ
ネットワークに所属

    api:
        image: registry:5000/swtest/api:latest          # ローカルレジストリのイメージから作
成
        deploy:
            replicas: 3                                 # 起動数=3
            placement:
                constraints: [node.role != manager]     # manager以外のnodeで起動する
        networks:
            - swtest_net                                # swtest_net というオーバーレイ
ネットワークに所属

networks:
    swtest_net:                                         # swtest_net というオーバーレイ
ネットワークを利用する
        external: true
```

3. stackをswarmクラスタにデプロイする

1) stackをデプロイする

manager CLIにて`/stack/webapi.yml`を`swtest`としてデプロイします。

`@manager`

```
docker stack deploy -c /stack/webapi.yml swtest

Creating service swtest_api
Creating service swtest_nginx
```

2) デプロイされた`stack`を確認

`@manager`

```
docker stack services swtest

ID                  NAME                MODE                REPLICAS
```

```
IMAGE                                  PORTS
do79y357rahf        swtest_nginx        replicated          3/3
registry:5000/swtest/nginx:latest
udgryplvejvb        swtest_api          replicated          3/3
registry:5000/swtest/api:latest
```

nginxとapiが３つづつ起動していることが確認出来ます。

3) デプロイされたコンテナを確認する

@manager

```
docker stack ps -f "DESIRED-STATE=Running" swtest

ID                  NAME                IMAGE                                  NODE
DESIRED STATE       CURRENT STATE           ERROR               PORTS
6523s6sxk0z4        swtest_nginx.1      registry:5000/swtest/nginx:latest
6960c0f9f754        Running             Running 9 minutes ago
s26ot14832tz        swtest_api.1        registry:5000/swtest/api:latest
6960c0f9f754        Running             Running 9 minutes ago
9awuu3u5ztdk        swtest_nginx.2      registry:5000/swtest/nginx:latest
d55c16d6f52e        Running             Running 9 minutes ago
wiz7gcwp4kl5        swtest_api.2        registry:5000/swtest/api:latest
19be9974e0c7        Running             Running 9 minutes ago
kt6erv2af1rq        swtest_api.3        registry:5000/swtest/api:latest
d55c16d6f52e        Running             Running 9 minutes ago
syp556p98zrr        swtest_nginx.3      registry:5000/swtest/nginx:latest
19be9974e0c7        Running             Running 9 minutes ago
```

周辺のアプリもデプロイし最終的な環境を構築する

1. Visualizerのデプロイ

swarmクラスタ起動状況を見える化するVisualizerを導入します。

1) 構成ファイルを確認

作業ディレクトリ\stack\visualizer.yml

```
version: "3"
services:
    app:
        image: registry:5000/visualizer                 # ローカルレジストリのイメージを使用
        ports:
            - "9000:8080"                               # 外部ポート9000番を内部ポート
8080にバインド
        volumes:
            - /var/run/docker.sock:/var/run/docker.sock
```

```
        deploy:
            mode: global
            placement:
                constraints: [node.role == manager]      # managerノードに配備する
```

localhost:9000 -> manager:9000:manager:8080 -> visualizer application という経路となります。

2) デプロイ

manager CLIにてvisualizer.ymlをデプロイします。

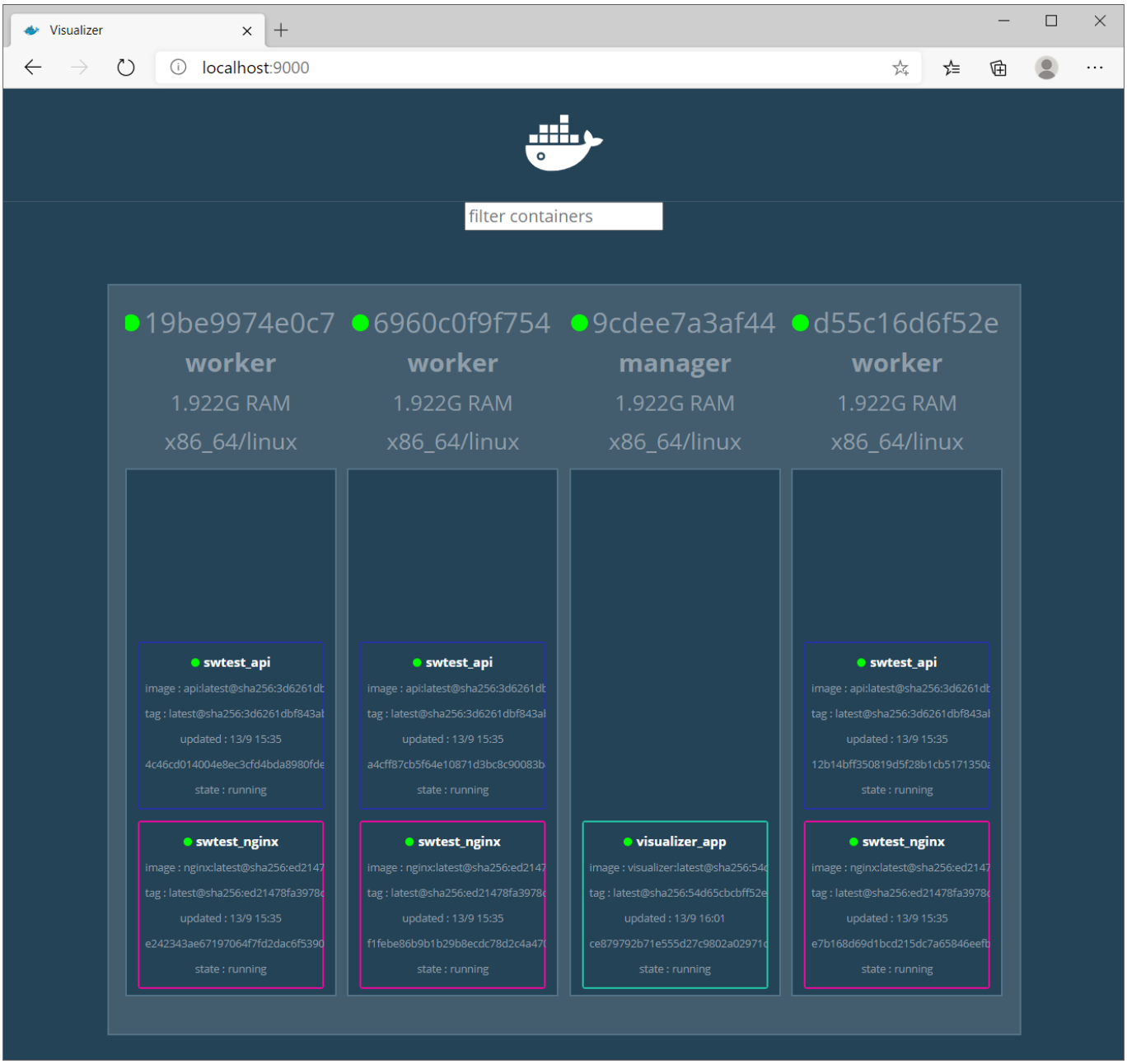
@manager

```
docker stack deploy -c /stack/visualizer.yml visualizer

Creating network visualizer_default
Creating service visualizer_app
```

3) 動作確認

WEBブラウザで下記のURLにアクセスします。

```
http://localhost:9000/
```



各ノードでのサービス起動状況が見える化されています。

2. Ingressのデプロイ

swarmクラスタ内のサービスをクラスタ外から利用するためには、オーバーレイネットワーク内にアクセスする必要があります。

これを実現するため、managerサーバー内に外部からのリクエストを内部のサービスに転送するプロキシを建てます。

プロキシhaproxyを導入します。

1) 構成ファイルを確認

作業ディレクトリ\stack\ingress.yml

```
version: "3"
services:
    haproxy:
        image: registry:5000/haproxy                    # ローカルレジストリのイメージを使用
        networks:
            - swtest_net                                # オーバーレイネットワークを利用
        volumes:
            - /var/run/docker.sock:/var/run/docker.sock
        deploy:
            mode: global
            placement:
                constraints: [node.role == manager]     # managerノードに配備する
        ports:
            - 80:80                                     # 外部ポート80番を内部ポート80
番にバインド
            - 1936:1936 # for stats page (basic auth, stats:stats)

networks:
    swtest_net:
        external: true
```

localhost:8000 -> manager:8000->80 -> haproxy:80->80 -> swtest_nginx:80->swtest_api:8080 -> api:8080

という経路となります。

2) デプロイ

manager CLIにてingress.ymlをデプロイします。

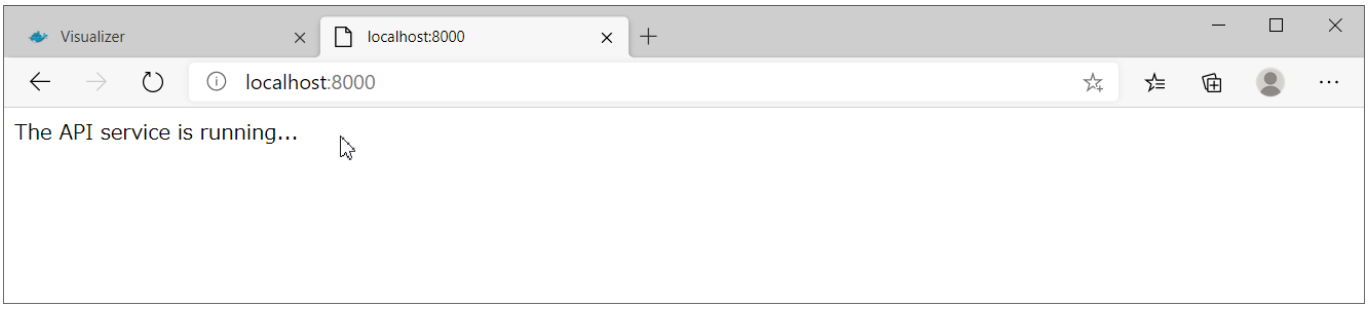
@manager

```
docker stack deploy -c /stack/ingress.yml ingress

Creating service ingress_haproxy
```

3) 動作確認

WEBブラウザで下記のURLにアクセスします。

```
http://localhost:8000/
```



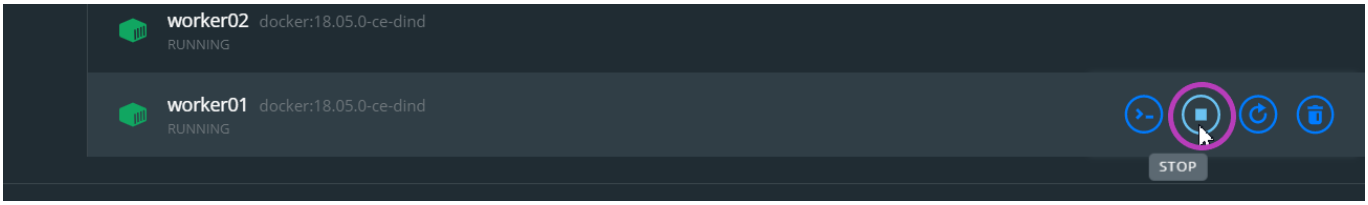
> The API service is running...

表示されたでしょうか？

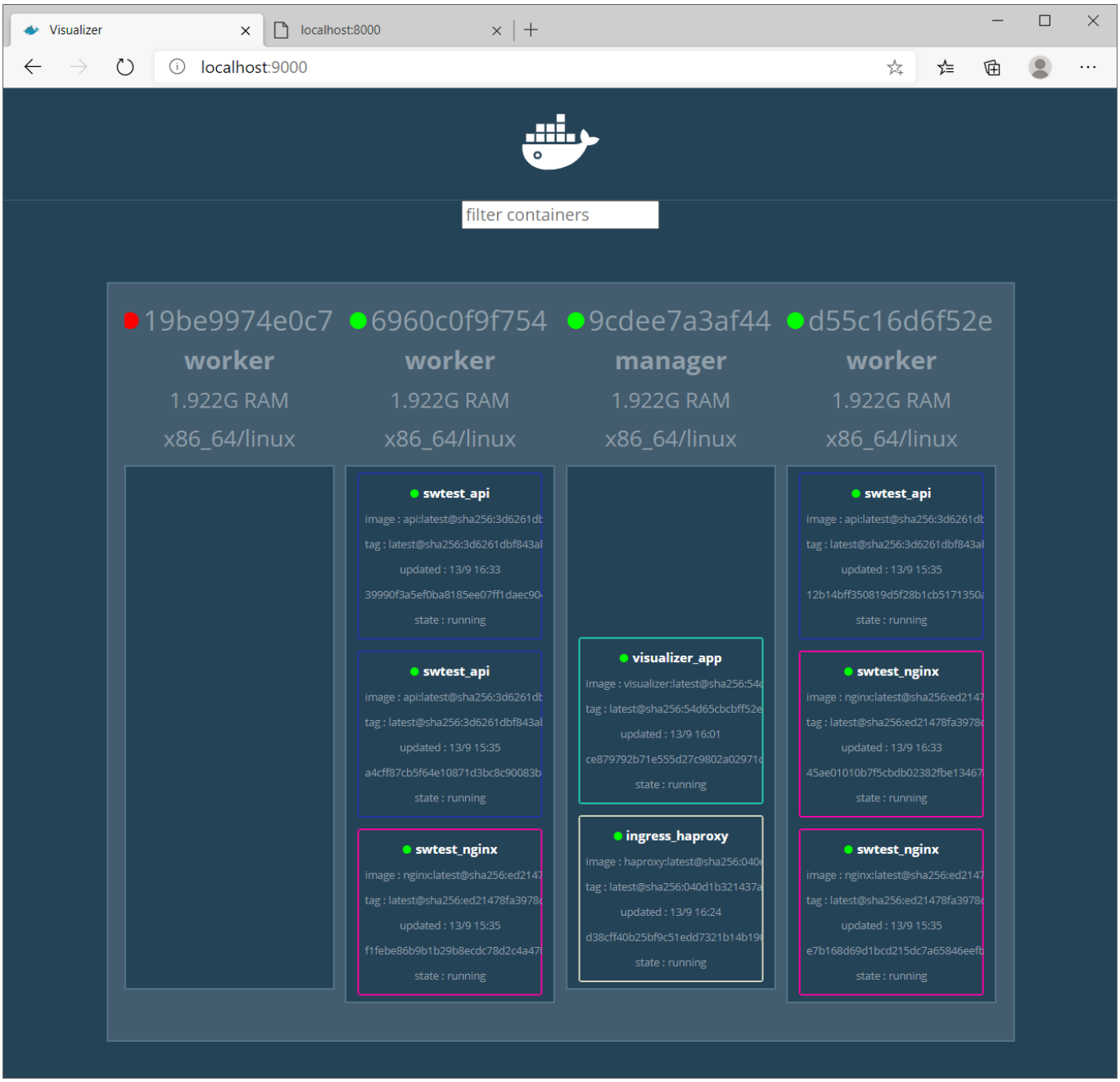
クラスタの動作確認

せっかくですので、ノードをひとつ止めてみましょう。

Dockerダッシュボードから、worker01を停止してみます。



しばらくすると、自動でサービスが起動されて、３つづつになっていることがvisualizerで確認できます。



最後に全環境を破棄します

ホストOSのコマンドラインで、作業ディレクトリに移動して、`docker-compose down` を実行します。

コンテナイメージファイルは残りますが、swarmクラスタ設定などはなくなります。

一時的に終了する場合は、作業ディレクトリで`docker-compose stop` を実行します。

もちろんGUIで停止しても構いません。

再度立ち上げたい場合は、作業ディレクトリで`docker-compose start` を実行します。