

docker-compose から k8s に入門してみる

2020-02-28 / 【関西】Kubernetes超入門勉強会

登壇者

@vvanitter82

VVani / allegrogiken

関西圏

エンジニア8年目くらい

所属:

株式会社メンバーズエッジ



私とコンテナの関係

- ○ 開発環境でのコンテナ活用経験
 - 環境構築は `docker-compose up` で
- ○ プロダクションでのコンテナ活用経験
 - サーバのセットアップを簡単にしたいという理由で、本番でも `docker-compose` を使ったことがある
- ○ パブリッククラウドでのコンテナ活用経験
 - 上の流れから、Amazon ECS をちょっとだけ
- ✕ 開発環境・プロダクションでのk8s活用経験
 - 勉強中
 - このスライドを作ってる間に学んでる

今回のテーマ

docker-compose up で立ち上がる開発環境を土台にして、
ローカルの kubernetes にデプロイするまでの流れを追体験してもらう

話さないこと

- 本番運用に耐えうる kubernetes の使い方
 - 開発環境用にフォーカスしています
- パブリッククラウド上(AWS とか GCP とか) の話
 - ローカル環境にフォーカスしています

この辺は他の方がしゃべってくれるでしょう 😊

前提

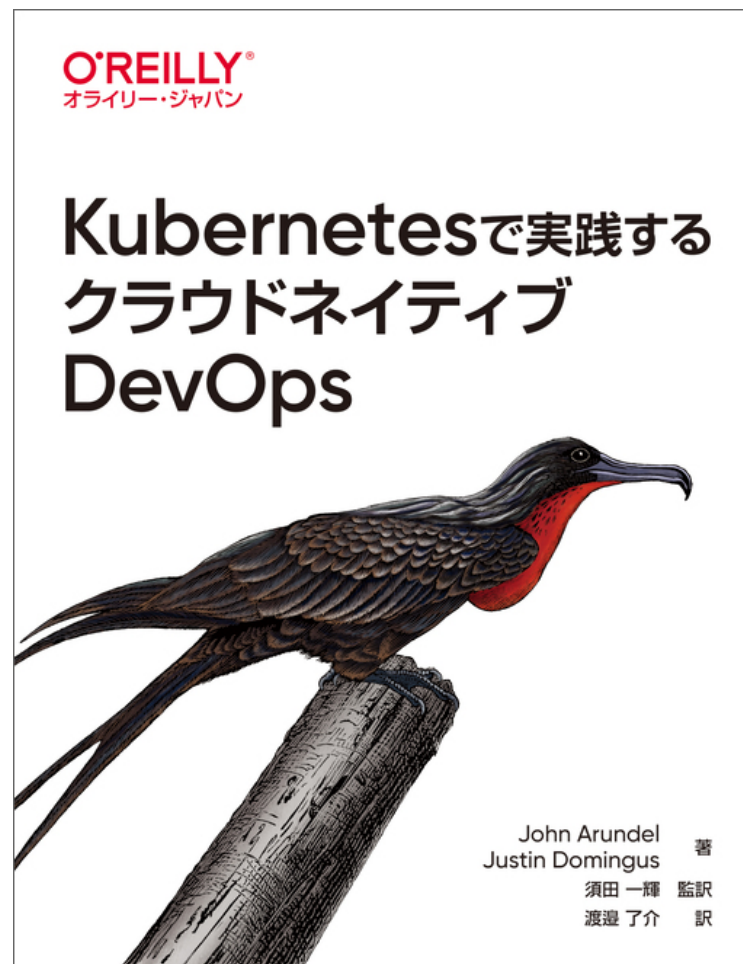
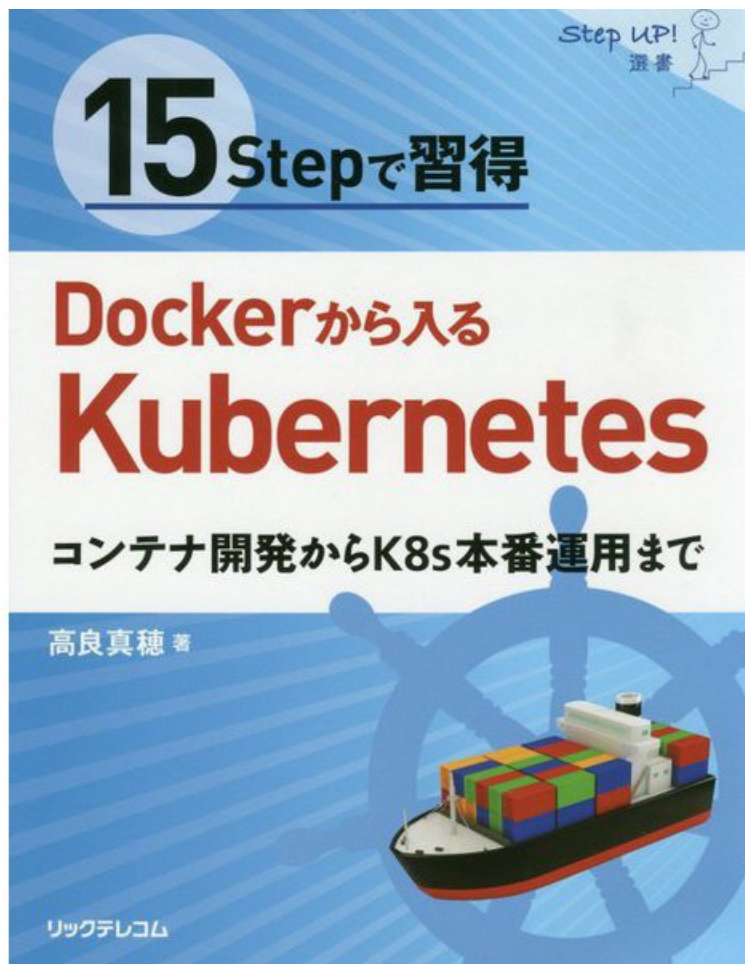
Docker Desktop インストール済み

- docker, docker-compose が使える
- kubernetes が使える
- kubectl が使える

試したときの環境

- Windows 10 Pro + WSL2(Ubuntu) + Docker Desktop
- Terminal 操作はすべて Ubuntu 側で実施
- Docker CE: v19.03.5 / Kubernetes: v1.15.5

読んでいる本



ここから本文

よくあるWEBアプリケーション

今回用のサンプルプロジェクト

- APIアプリケーション
 - 個人的な趣味を兼ねて `crystal` という Ruby 風の静的型付け言語で作成
- redis
- 「開発環境/CI 向け」に Dockerfile / docker-compose.yml を書いている

データベースはモロモロの都合で省略 😊

Dockerfile

```
# ビルドしてシングルバイナリを作成する用のコンテナ
FROM crystallang/crystal:0.33.0-alpine-build AS builder

RUN mkdir /app
WORKDIR /app

ADD ["/shard.yml", "/shard.lock", "/"]
RUN shards install

ADD ./ .
RUN crystal build src/main.cr --release --static

# ビルド済みバイナリを builder から持ってきて動かすコンテナ
FROM busybox

WORKDIR /root
COPY --from=builder /app/main .

CMD ["/main"]
```

docker-compose.yml

```
version: "3.7"
services:
  api:
    build:
      context: ./api
    env_file: .env
    ports:
      - 3000
    depends_on:
      - kvs
  kvs:
    image: redis:6.0-rc1
```

.env

docker-compose.yml でサービスに対して設定できる環境変数群

サービスに対して設定できるほか、 `docker-compose.yml` への展開もできる

```
REDIS_HOST=kvs  
REDIS_PORT=6379
```

立ち上げる

```
$ docker-compose up -d
Starting sample_project_kvs_1 ... done
Starting sample_project_api_1 ... done
```

```
$ docker-compose ps
```

Name	Command	State	Ports
sample_project_api_1	./main	Up	0.0.0.0:32779->3000/tcp
sample_project_kvs_1	docker-entrypoint.sh redis ...	Up	6379/tcp

APIが動いている様子

```
$ curl localhost:32779  
Hello World! #1↵
```

```
$ curl localhost:32779  
Hello World! #2↵
```

```
$ curl localhost:32779  
Hello World! #3↵
```

コールされる度に Redis の特定キーを加算するだけ

ここまでは docker-compose のはなし

docker-compose との付き合いが長いので、k8s を使うときでも docker-compose から入りたい
(わがまま)

Kompose

docker-compose.yml からk8sのyamlファイルを生成するCLIツール

Kubernetes + Compose = Kompose

<https://kompose.io> / <https://github.com/kubernetes/kompose>

インストール (on Ubuntu)

※ 2020/02/20 時点の `README.md` から引用

```
$ curl -L https://github.com/kubernetes/kompose/releases/download/v1.20.0/kompose-linux-amd64 -o kompose
$ chmod +x kompose
$ sudo mv ./kompose /usr/local/bin/kompose
```


使い方

```
$ kompose
```

Kompose is a tool **to** help users who are familiar with docker-compose move **to** Kubernetes.

Usage:

```
kompose [command]
```

Available Commands:

completion	Output shell completion code
convert	Convert a Docker Compose file
down	Delete instantiated services/deployments from kubernetes
help	Help about any command
up	Deploy your Dockerized application to a container orchestrator.
version	Print the version of Kompose

Flags:

--error-on-warning	Treat any warning as an error
-f, --file stringArray	Specify an alternative compose file
-h, --help	help for kompose
--provider string	Specify a provider. Kubernetes or OpenShift. (default "kubernetes")
--suppress-warnings	Suppress all warnings
-v, --verbose	verbose output

やってみる

```
$ kompose convert -f docker-compose.yml  
  
INFO Kubernetes file "api-service.yaml" created  
INFO Kubernetes file "api-deployment.yaml" created  
INFO Kubernetes file "api-env-configmap.yaml" created  
INFO Kubernetes file "kvs-deployment.yaml" created
```

作業ディレクトリになんか出た 🙄

レッツデプロイ

```
$ kubectl apply -f api-deployment.yaml  
deployment.extensions/api created
```

```
$ kubectl apply -f api-service.yaml  
service/api created
```

いけてそう 😊

デプロイした結果

```
$ kubectl get pods
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
default	api-5948c9b999-mvz7x	0/1	ImagePullBackOff	0	44m
...					

ダメそう 🙄🙄🙄

STATUS: ImagePullBackOff

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
default	api-5948c9b999-mvz7x	0/1	ImagePullBackOff	0	44m

GKE のトラブルシューティングいわく

[https://cloud.google.com/kubernetes-engine/docs/troubleshooting?
hl=ja#ImagePullBackOff](https://cloud.google.com/kubernetes-engine/docs/troubleshooting?hl=ja#ImagePullBackOff)

ImagePullBackOff と ErrImagePull は、コンテナが使用するイメージをイメージ レジストリからロードできないことを示します。

deployment のファイルを見してみる

```
...
spec:
  containers:
  - env:
    - name: REDIS_HOST
      valueFrom:
        configMapKeyRef:
          key: REDIS_HOST
          name: api-env
    - name: REDIS_PORT
      valueFrom:
        configMapKeyRef:
          key: REDIS_PORT
          name: api-env
  image: api
  name: api
```

コンテナイメージを参照できていない

`api` という名前では `docker-compose build` で作ったイメージを参照できない

```
$ docker image ls api
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
------------	-----	----------	---------	------

`docker-compose build` のデフォルトでは、コンテナイメージに `${project}_${service}` という名前をつける

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
sample_project_api	latest	5ee77759cbd3	1 hours ago	17.4MB

deployment を書き換えてみる

```
...
spec:
  containers:
  - env:
    - name: REDIS_HOST
      valueFrom:
        configMapKeyRef:
          key: REDIS_HOST
          name: api-env
    - name: REDIS_PORT
      valueFrom:
        configMapKeyRef:
          key: REDIS_PORT
          name: api-env
  - image: api
+ image: sample_project_api
  name: api
```


deployment を書き換えてみる

```
$ kubectl apply -f api-deployment.yaml  
deployment.extensions/api configured
```

```
$ kubectl get pods
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
default	api-5948c9b999-mvz7x	0/1	ErrImagePull	0	14s

ステータスは変わったけどダメそう

名前からイメージの解決はできたけど、pull しようとしている？ 🤔

k8s でローカルのコンテナを使う

<https://kubernetes.io/ja/docs/concepts/configuration/overview/#コンテナイメージ>

コンテナイメージ

`imagePullPolicy`とイメージのタグは、`kubelet`が特定のイメージをpullしようとしたときに作用します。

- `imagePullPolicy: IfNotPresent`: ローカルでイメージが見つからない場合にのみイメージをpullします。
- `imagePullPolicy: Always`: Podの起動時に常にイメージをpullします。
- `imagePullPolicy` のタグが省略されていて、利用してるイメージのタグが `:latest` の場合や省略されている場合、`Always` が適用されます。
- `imagePullPolicy` のタグが省略されていて、利用してるイメージのタグはあるが `:latest` でない場合、`IfNotPresent` が適用されます。
- `imagePullPolicy: Never`: 常にローカルでイメージを探そうとします。ない場合にもイメージはpullしません。

💡 ローカルのコンテナイメージが `tag=latest` なのが悪そう

`docker-compose.yml` でタグを付ける

`docker-compose build` でビルドする時にイメージ名・タグを指定できる

<http://docs.docker.jp/compose/compose-file.html#build>

`build` と `image` の両方を記述するとそのように動く

```
version: "3.7"
services:
  api:
    build:
      context: ./api
+   image: sample_project/api:0.0.1
    env_file: .env
    ports:
      - 3000
    depends_on:
      - kvs
  kvs:
    image: redis:6.0-rc1
```

タグをつけて docker-compose build

```
$ docker-compose build

kvs uses an image, skipping
Building api
...(中略)...

Successfully built 12a0b15e0dcf
Successfully tagged sample_project/api:0.0.1
```

```
$ docker image ls sample_project/api
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
sample_project/api	0.0.1	12a0b15e0dcf	43 hours ago	17.4MB

イメージ名:タグ付きでビルドができた

続いて kompose

api-deployment.yml

```
spec:
  containers:
  - env:
    - name: REDIS_HOST
      valueFrom:
        configMapKeyRef:
          key: REDIS_HOST
          name: api-env
    - name: REDIS_PORT
      valueFrom:
        configMapKeyRef:
          key: REDIS_PORT
          name: api-env
  image: sample_project/api:0.0.1
  name: api
```

コンテナイメージの値がいい感じになった(気がする)

改めてデプロイ

```
$ kubectl apply -f api-deployment.yaml  
deployment.extensions/api configured
```

```
$ kubectl get pods
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
default	api-7f5f4fdbf7-67nqm	0/1	CreateContainerConfigError	0	13s

ダメっぽい、しかしSTATUSのエラーは変わった

イメージは参照できている模様

CreateContainerConfigError なるほど

そういえば

api-deployment.yml に `configMapKeyRef` というものが見えている

```
spec:
  containers:
    - env:
      - name: REDIS_HOST
        valueFrom:
          configMapKeyRef:
            key: REDIS_HOST
            name: api-env
      - name: REDIS_PORT
        valueFrom:
          configMapKeyRef:
            key: REDIS_PORT
            name: api-env
    image: sample_project/api:0.0.1
    name: api
```

configmap

`kompose` で生成されたモノの中に同じような響きのものがあった

api-env-configmap.yaml

```
apiVersion: v1
data:
  REDIS_HOST: kvs
  REDIS_PORT: "6379"
kind: ConfigMap
metadata:
  creationTimestamp: null
  labels:
    io.kompose.service: api-env
  name: api-env
```

`docker-compose` の時は `.env` に書いていた内容となっている

configmap もデプロイしてみる

```
$ kubectl apply -f api-env-configmap.yaml  
configmap/api-env created
```

```
$ kubectl get pods
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
default	api-7f5f4fdbf7-67nqm	0/1	CrashLoopBackOff	1	5h8m

またSTATUSが変わったが、起動はできてない模様

ログを見る

```
$ kubectl logs api-7f5f4fdbf7-67nqm
```

```
Unhandled exception: Socket::Addrinfo::Error: Hostname lookup  
for kvs failed: No address found (Redis::CannotConnectError)  
  from /app/lib/redis/src/redis/connection.cr:10:5 in 'connect'  
  from /app/src/main.cr:7:1 in '__crystal_main'  
  from /usr/share/crystal/src/crystal/main.cr:106:5 in 'main'
```

Redis の接続エラーでアプリケーションが落ちている

そういえば Redis のデプロイを行っていない 😊

Redisをk8sにデプロイしたいけど

```
$ kompose convert -f docker-compose.yml  
  
INFO Kubernetes file "api-service.yaml" created  
INFO Kubernetes file "api-deployment.yaml" created  
INFO Kubernetes file "api-env-configmap.yaml" created  
INFO Kubernetes file "kvs-deployment.yaml" created
```

kompose が kvs-service.yaml を出力していない 🤔

Redisのserviceも出したい

apiサービスとの違いとして `port` 記述の有無があるが..
これを加えると出てくるのではないだろうか（雑な直感）

docker-compose.yml

```
version: "3.7"
services:
  api:
    build:
      context: ./api
    env_file: .env
    ports:
      - 3000
    depends_on:
      - kvs
  kvs:
    image: redis:6.0-rc1
+   ports:
+     - 6379
```

出た

```
$ kompose convert -f docker-compose.yml

INFO Kubernetes file "api-service.yaml" created
+ INFO Kubernetes file "kvs-service.yaml" created
INFO Kubernetes file "api-deployment.yaml" created
INFO Kubernetes file "api-env-configmap.yaml" created
INFO Kubernetes file "kvs-deployment.yaml" created
```

デプロイしよう

```
$ kubectl apply -f kvs-deployment.yaml
deployment.extensions/kvs created
$ kubectl apply -f kvs-service.yaml
service/kvs created
```

やってみたが・・・

```
$ kubectl get pods
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
default	api-7f5f4fdbf7-67nqm	0/1	CrashLoopBackOff	10	5h37m
default	kvs-d44fc5984-vwqst	1/1	Running	0	2m27s

apiのpodが起動しない状態は変わらず

起動 -> エラー -> 再起動 -> エラー ... のループにはまったので、
10回もやったしもう無理よね・・・っていう状態に見える 🤔

人為的に再起動させたら回復しそう

`kubect1` に「再起動する」みたいはものは無さそうだが・・
雑に検索すると「レプリカ数を 0 にしてから 1以上 にすると良い」とある

レプリカ数を 0 にする

```
$ kubectl scale deployment api --replicas=0  
deployment.extensions/api scaled
```

```
$ kubectl get pods
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
default	kvs-d44fc5984-vwqst	1/1	Running	0	3m37s
...					

レプリカ数を元に戻す

```
$ kubectl scale deployment api --replicas=1  
deployment.extensions/api scaled
```

```
$ kubectl get pods
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
default	api-7f5f4fdbf7-97k6k	1/1	Running	0	4s
default	kvs-d44fc5984-vwqst	1/1	Running	0	3m52s

apiサービスが Running になった！ 🎉

手元からアクセスしてみたい

サービスの情報を見える

```
$ kubectl get service api kvs
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
api	ClusterIP	10.104.4.109	<none>	3000/TCP	22h
kvs	ClusterIP	10.102.147.186	<none>	6379/TCP	13h

この状態でブラウザから `CLUSTER-IP` に対してアクセスしても応答が無い 🤔

```
$ curl 10.104.4.109:3000
```

```
curl: (7) Failed to connect to 10.104.4.109 port 3000: Connection timed out
```

ServiceSpec: type=ClusterIP とは

k8s公式チュートリアルの「Serviceを使ったアプリケーションの公開」

<https://kubernetes.io/ja/docs/tutorials/kubernetes-basics/expose/expose-intro/>

ClusterIP（既定値）

クラスター内の内部IPでServiceを公開します。
この型では、Serviceはクラスター内からのみ到達可能になります。

おそらく・・・

- サービスの設定で `type` を省略すると `ClusterIP` になる
- `ClusterIP` のサービス単体では外部への公開ができないっぽい
- `ClusterIP` のサービス同士での通信はできているっぽい (`api <=> kvs`)

ServiceSpec: type=NodePort

さっきのURLと同じページにある

NodePort

NATを使用して、クラスター内の選択された各ノードの同じポートにServiceを公開します。
<NodeIP>:<NodePort>を使用してクラスターの外部からServiceにアクセスできるようにします。
これはClusterIPのスーパーセットです。

これ使ったらできそうな予感がする（雑な感覚）

NodePort を使ってみる (1)

api-service.yaml の `spec` に `type: NodePort` だけ追記

```
spec:
+ type: NodePort
  ports:
    name: "3000"
    port: 3000
    targetPort: 3000
  selector:
    io.kompose.service: api
```

NodePort を使ってみる (2)

```
$ kubectl apply -f api-service.yaml  
service/api configured
```

```
$ kubectl get service api kvs
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
api	NodePort	10.104.4.109	<none>	3000:31690/TCP	22h
kvs	ClusterIP	10.102.147.186	<none>	6379/TCP	14h

ポートが 3000:31690 という表記に変わった

ノード側のポートは空いてるところを自動で割り当てたように見える

アクセスしてみる

```
$ curl localhost:31690  
Hello World! #1↵
```

```
$ curl localhost:31690  
Hello World! #2↵
```

できた！ 🎉

ここまでの学び

- kompose はしっかり使えた
- docker-compose.yml の書き方にちょっとコツがある
 - Dockerfile を書いている場合、コンテナイメージ名:タグ の記載が必要
 - ports を書いてない場合はサービスが作られない
- 外部からアクセスできる状態にするには kompose 後にひと手間必要
 - service の type をいじればOK

(なんとなく)カイゼン

1 kompose で生成されたファイルを弄らず `kubectl apply` したい

- できるだけ docker-compose.yml を軸としたいというわがまま
- 今回書き換えたのは Service の `type: NodePort` だけ

2 docker-compose 同様にグルーピングしたい

- 今のままだと他の何かを試したときに混ざる予感がある
- docker-compose では作業ディレクトリをもとにいい感じのグルーピングが働くので、それと同じ感じにできたらいいな

kompose の生成ファイルを弄らずデプロイしたい (1)

kompose のユーザーガイドを見ると、いい感じの記述がある

<https://github.com/kubernetes/kompose/blob/master/docs/user-guide.md>

Labels

kompose supports Kompose-specific labels within the docker-compose.yml file to explicitly define the generated resources' behavior upon conversion, like Service, PersistentVolumeClaim...

kompose の生成ファイルを弄らずデプロイしたい (2)

docker-compose.yml に `label1` を書き加えてみよう

```
services:
  api:
    image: sample_project/api:0.0.1
    build:
      context: ./api
    env_file: .env
    ports:
      - 3000
+   labels:
+     kompose.service.type: nodeport
    depends_on:
      - kvs
  kvs:
    image: redis:6.0-rc1
    ports:
      - 6379
```

kompose の生成ファイルを弄らずデプロイしたい (3)

```
$ kompose convert -f docker-compose.yml  
  
INFO Kubernetes file "api-service.yaml" created  
...
```

api-service.yaml

```
spec:  
  ports:  
    - name: "3000"  
      port: 3000  
      targetPort: 3000  
  selector:  
    io.kompose.service: api  
  type: NodePort
```

さきほど手で修正したやつと同じ内容になった 🍗

(なんとなく)カイゼン

✓ ~~komposeで生成されたファイルを弄らず~~ `kubectl apply` したい

- ~~できるだけ docker-compose.yml を軸としたい~~
- ~~今回書き換えたのは Service の~~ `type: NodePort` ~~だけ~~

2 `docker-compose` 同様にグルーピングしたい

- 今のままだと他の何かを試したときに混ざる予感がある
- `docker-compose` では作業ディレクトリをもとにいい感じのグルーピングが働くので、それと同じ感じにできたらいいな

docker-compose 同様にグルーピングしたい (1)

namespaceという概念を使うとうまくきそう、公式の docs に書いてある

<https://kubernetes.io/ja/docs/concepts/overview/working-with-objects/namespaces/>

- **1** `kubect1 apply` する際に `-n $NAMESPACE` で指定する
 - `-n` を省略した際の namespace が `default` になっている
 - 今回はこっちをやってみる
- **2** 省略時の namespace を変更する
 - docs では `Namespace設定の永続化` と書かれているところ

docker-compose 同様にグルーピングしたい (2)

やってみる

```
$ kubectl create namespace sample-project
namespace/sample-project created
$ kubectl apply -f api-env-configmap.yaml -n sample-project
configmap/api-env created
$ kubectl apply -f kvs-deployment.yaml -n sample-project
deployment.extensions/kvs created
$ kubectl apply -f api-deployment.yaml -n sample-project
deployment.extensions/api created
$ kubectl apply -f kvs-service.yaml -n sample-project
service/kvs created
$ kubectl apply -f api-service.yaml -n sample-project
service/api created
```

ちょっとめんどいけど、ここは愚直に・・

docker-compose 同様にグルーピングしたい (3)

```
$ kubectl get pods -n sample-project
```

NAME	READY	STATUS	RESTARTS	AGE
api-7f497f79cf-jzjw9	1/1	Running	0	67s
kvs-d44fc5984-22cnq	1/1	Running	0	8m48s

```
$ kubectl get service -n sample-project
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
api	NodePort	10.101.83.11	<none>	3000:31414/TCP	79s
kvs	ClusterIP	10.96.105.90	<none>	6379/TCP	3m43s

```
$ curl localhost:31414  
Hello World! #1↵
```

default とは別の名前空間で別のモノとしてデプロイできた 🍏

docker-compose 同様にグルーピングしたい (4)

default も sample-project も動いている様子

```
$ kubectl get pods --all-namespaces
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
default	api-7f5f4fdbf7-97k6k	1/1	Running	2	23h
default	kvs-d44fc5984-vwqst	1/1	Running	0	23h
sample-project	api-7f497f79cf-jzjw9	1/1	Running	0	3h8m
sample-project	kvs-d44fc5984-22cnq	1/1	Running	0	3h15m
...					

せっかくなので default 側に作ってしまったモノをお掃除しよう

```
$ kubectl delete service kvs api  
service "kvs" deleted  
service "api" deleted
```

```
$ kubectl delete deployment kvs api  
deployment.extensions "api" deleted  
deployment.extensions "kvs" deleted
```

docker-compose 同様にグルーピングしたい (5)

namespace 自体を消すと中にあるやつが一気に消えてくれるみたい

```
$ kubectl get pods -n sample-project
```

NAME	READY	STATUS	RESTARTS	AGE
api-7f497f79cf-jzjw9	1/1	Running	0	26h
kvs-d44fc5984-22cnq	1/1	Running	0	26h

```
$ kubectl delete namespace sample-project  
namespace "sample-project" deleted
```

```
$ kubectl get pods -n sample-project  
No resources found.
```

 namespace に属さないものもあるらしい

<https://kubernetes.io/ja/docs/concepts/overview/working-with-objects/namespaces/>

(なんとなく)カイゼンできた

✓ kompose で生成されたファイルを弄らず `kubectl apply` したい

- できた 🎉

✓ `namespace: default` を変更したい

- (ちょっとめんどいけど) できた 🎉
- 楽にしたい場合は省略時の `Namespace`設定の永続化 をすると良い
 - <https://github.com/ahmetb/kubectx> がさらに便利らしい

総仕上げ (1)

これまでの学びで、docker-compose.yml をもとに

`docker-compose up/down` 感覚で k8s にデプロイできるようになったはず

総仕上げ (2)

kompose の生成ファイルが `docker-compose.yml` と混ざるとつらいので、
k8sのyamlファイル達を置く専用のフォルダを切っておく

ディレクトリのイメージ

- /
 - api/
 - src/**
 - Dockerfile
 - kompose-files/
 - (komposeで生成されたyaml)
 - docker-compose.yml

📌 k8s に関するコマンドは `kompose-files` に移動して叩く

総仕上げ (3)

コンテナイメージのビルドは `docker-compose` でやる

```
$ docker-compose build
...
Successfully built fdb1a88f5921
Successfully tagged sample_project/api:0.0.1
```

`docker-compose.yml` と混ざらないように `kompose` を動かす

```
$ kompose convert -f ../docker-compose.yml

INFO Kubernetes file "api-service.yaml" created
INFO Kubernetes file "kvs-service.yaml" created
INFO Kubernetes file "api-deployment.yaml" created
INFO Kubernetes file "api-env-configmap.yaml" created
INFO Kubernetes file "kvs-deployment.yaml" created
```

総仕上げ (4)

namespace を作っておき、 `kompose` で出た全ファイルを `kubectl apply` する

```
$ kubectl create namespace sample-project
namespace/sample-project created

$ ls *.yaml | xargs -I@ kubectl apply -n sample-project -f @
deployment.extensions/api created
configmap/api-env created
service/api created
deployment.extensions/kvs created
service/kvs created
```

これでデプロイが完了したはず 🍷

総仕上げ (5)

動作確認

```
$ kubectl get service -n sample-project
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
api	NodePort	10.103.98.128	<none>	3000:32299/TCP	57s
kvs	ClusterIP	10.105.145.185	<none>	6379/TCP	56s

```
$ curl localhost:32299  
Hello World! #1↵
```

いけてそう、なので全部消す

```
$ kubectl delete namespace sample-project  
namespace "sample-project" deleted
```


いかがでしたか？

感想

体験として..

- docker-compose.yml からわりとシームレスに k8s に入門できた
 - `kompose` はそれを助けるのに十分なツールだった
 - yaml を書き始めるハードルはそこそこ下がる
- k8s を意識して docker-compose.yml を書くことができた
 - 「開発環境として」の運用だと、自前コンテナの名前・タグをあまり意識しないと思うが、意識すると (k8sに関わらず) 本番向けにコンテナを使う機運が高まるはず

感想

実用面を考えると..

- 普段使いの開発環境として k8s を継続的に使っていくメリットは薄そう
 - 特にフロントエンド・動的型付けなスクリプト言語の開発において、HMR(Hot Module Replacement) を使うなら `docker-compose` で ホスト:コンテナ間をボリュームマウントする方が楽に見える
- ネットワーク周りを意識する必要があるのは悪くない
 - 本番環境を作るときでも「構成」のイメージができているはず
 - その分ハードルは上がるけど..

注意点

- 変換済み yaml の `apiVersion` が現時点で古いかも
 - Deployment が `extensions/v1beta1` が出るが k8s v1.16 以降で非推奨
 - 今後のアップデートで更新される予定
- `docker-compose.yml` のサポートバージョンが `3.2` まで
 - `3.7` とかのファイルを拒絶するわけではないっぽい
- そのまま本番環境で使えるわけではない
 - 特に外部公開に `NodePort` を使っているあたり
 - 本番ではロードバランサーを使うのが推奨っぽい
 - 本番運用まで考えるならしっかり本を読みましよう

次は？

とりあえずローカルを超える

- 今回は k8s Pod へのアクセスに「NodePort」を雑に使った
 - たぶんこれは k8s らしさ低めの設定だと思う・・・
 - LoadBalancer / Ingress を使ってドメイン当てたりしてみたい
 - ブランチごとのテスト用環境を `ブランチ名 = k8s namespace` な関係でCIを使ってさくさく作れたら楽しそう
- 本を読もう
- GKE or EKS を触ろう

ご清聴ありがとうございました

ここからはおまけ

namespace: default を変更したい

kompose 側の情報も眺めてみる..

```
$ kompose
```

```
Kompose is a tool to help users who are familiar with docker-compose move to Kubernetes.
```

```
Usage:
```

```
  kompose [command]
```

```
Available Commands:
```

```
  completion  Output shell completion code
  convert      Convert a Docker Compose file
  down         Delete instantiated services/deployments from kubernetes
  help         Help about any command
  up           Deploy your Dockerized application to a container orchestrator.
  version      Print the version of Kompose
```

up, down というコマンドが見える

kompose up/down

kompose up の usage を見る

```
$ kompose up --help
```

Deploy your Dockerized application to a container orchestrator. (default "kubernetes")

Usage:

```
kompose up [flags]
```

OpenShift Flags:

--build-branch	Specify repository branch to use for buildconfig (default is current branch name)
--build-repo	Specify source repository for buildconfig (default is current branch's remote url)
--insecure-repository	Specify to use insecure docker repository while generating Openshift image stream object

Flags:

--build string	Set the type of build ("local" "build-config" (OpenShift only) "none") (default "local")
--controller string	Set the output controller ("deployment" "daemonSet" "replicationController")
-h, --help	help for up
--namespace string	Specify Namespace to deploy your application (default "default")
--push-image	If we should push the docker image we built (default true)

namespaceを指定しつつ、`docker-compose up` のノリで使えそうな雰囲気 🤔

kompose up/down

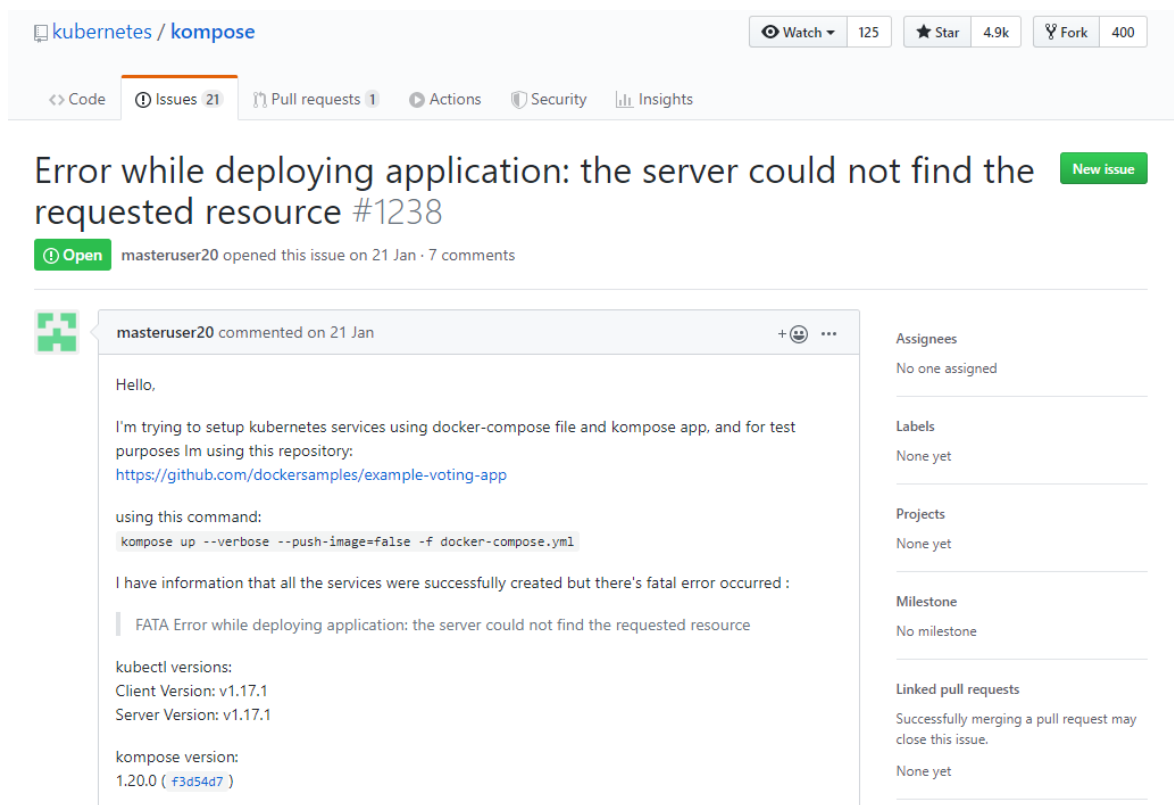
kompose up をやってみる

```
$ kompose up -f docker-compose.yml --namespace=sample-project --push-image=false  
  
INFO Build key detected. Attempting to build image 'sample_project/api:0.0.1'  
INFO Building image 'sample_project/api:0.0.1' from directory 'api'  
INFO Image 'sample_project/api:0.0.1' from directory 'api' built successfully  
INFO We are going to create Kubernetes Deployments, Services and PersistentVolumeClaims for  
your Dockerized application. If you need different kind of resources, use the 'kompose convert  
and 'kubectl create -f' commands instead.  
  
INFO Deploying application in "sample-project" namespace  
FATA Error while deploying application: the server could not find the requested resource (post services)
```

エラー出た 😞😞

kompose up/down

<https://github.com/kubernetes/kompose/issues/1238>



The screenshot shows the GitHub interface for the `kubernetes/kompose` repository. At the top, there are navigation tabs for `<> Code`, `Issues 21` (which is selected), `Pull requests 1`, `Actions`, `Security`, and `Insights`. To the right of these tabs are buttons for `Watch` (125), `Star` (4.9k), and `Fork` (400). Below the navigation bar, the issue title is "Error while deploying application: the server could not find the requested resource #1238", followed by a `New issue` button. Below the title, a green `Open` button is shown, along with the text "masteruser20 opened this issue on 21 Jan · 7 comments". The main content area shows a comment from `masteruser20` dated "21 Jan". The comment text is: "Hello, I'm trying to setup kubernetes services using docker-compose file and kompose app, and for test purposes Im using this repository: <https://github.com/dockersamples/example-voting-app> using this command: `kompose up --verbose --push-image=false -f docker-compose.yml` I have information that all the services were successfully created but there's fatal error occurred : FATA Error while deploying application: the server could not find the requested resource kubectl versions: Client Version: v1.17.1 Server Version: v1.17.1 kompose version: 1.20.0 (f3d54d7)". To the right of the comment, there are sections for `Assignees` (No one assigned), `Labels` (None yet), `Projects` (None yet), `Milestone` (No milestone), and `Linked pull requests` (Successfully merging a pull request may close this issue. None yet).

次のリリース(おそらく近日)でなおる予感
master をビルドしたやつだと動かしらい？（未検証）