

Internet Week 2018



Kubernetesハンズオン



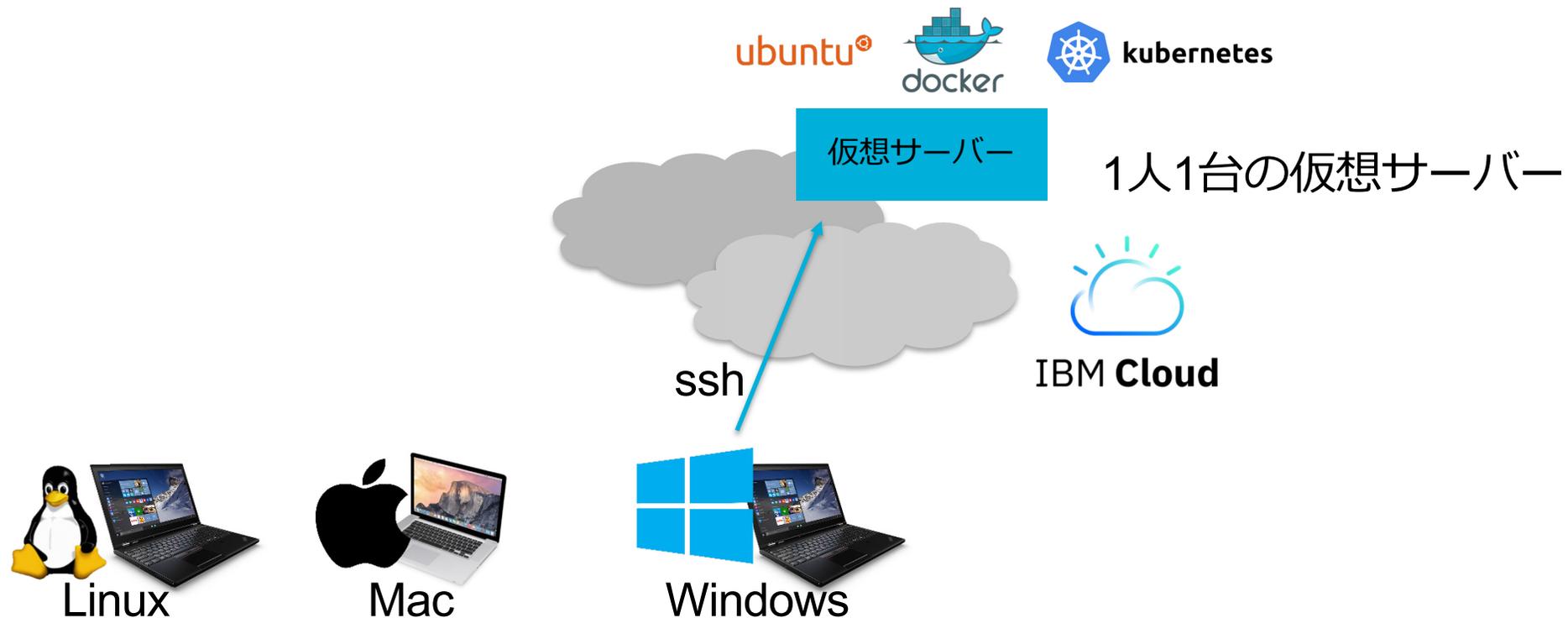
2018年11月28日

日本アイ・ビー・エム株式会社
クラウド事業本部 高良 真穂

本日のハンズオンの環境

- このハンズオンセッションでは、以下を利用します。

- 想定クライアント： Windows PC、Mac、Linux
- 必要なクライアントソフト： sshクライアント (Windowsの場合はTeraTerm,Putty等)



ログイン後の確認

次のコマンドを実行して、master is running が表示されることを確認してください。

```
$ kubectl cluster-info
Kubernetes master is running at https://192.168.99.100:8443
KubeDNS is running at https://192.168.99.100:8443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

ノードのリスト表示をお願いします。

```
$ kubectl get node
NAME      STATUS    ROLES    AGE    VERSION
minikube  Ready    master   14m    v1.10.0
```

レッスン #1

Kubernetes基本動作の理解

コンテナを実行してメッセージが表示されることを確認してください。

コンテナ hello-world を実行して、動作を確認します。

```
kubectl run hello-world --image=hello-world -it --restart=Never
```

以下のように表示されていれば、コンテナがk8s上で正常に実行されています。

```
Hello from Docker!  
This message shows that your installation appears to be working correctly.  
  
To generate this message, Docker took the following steps:  
1. The Docker client contacted the Docker daemon.  
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.  
   (amd64)  
3. The Docker daemon created a new container from that image which runs the  
   executable that produces the output you are currently reading.  
4. The Docker daemon streamed that output to the Docker client, which sent it  
   to your terminal.
```

<以下省略>

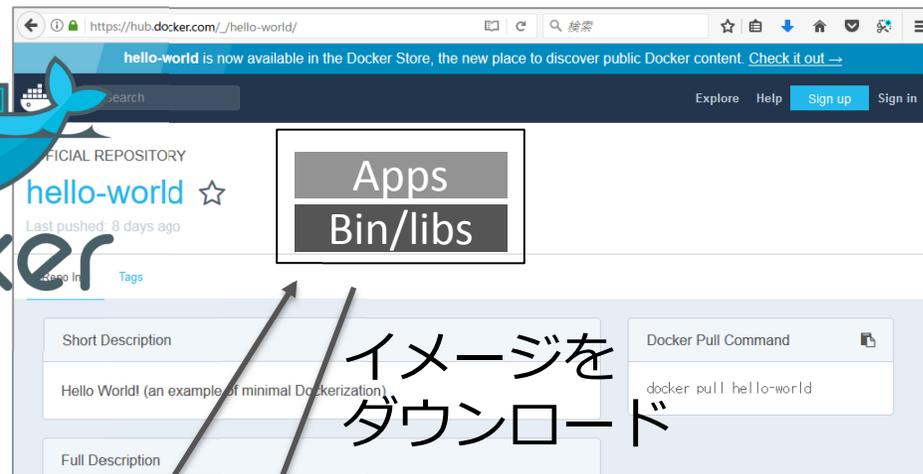
Hello-Worldメッセージ表示までの過程

– 画面表示は、①～④の動作によって、得られた表示です。

DockerHub
コンテナレジストリ



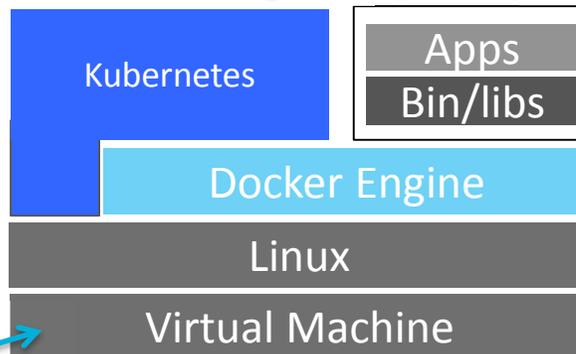
https://hub.docker.com/_/hello-world/



イメージを
ダウンロード

① コマンド実行

```
kubectl run hello-world --image=hello-world -it --restart=Never
```



④ コンテナ実行 & メッセージ

```
Hello from Docker!  
This message shows that your installation appears to be working correctly.  
  
To generate this message, Docker took the following steps:  
1. The Docker client contacted the Docker daemon.  
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.  
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.  
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.  
  
To try something more ambitious, you can run an Ubuntu container with:  
$ docker run -it ubuntu bash  
  
Share images, automate workflows, and more with a free Docker ID:  
https://cloud.docker.com/  
  
For more examples and ideas, visit:  
https://docs.docker.com/engine/userguide/
```

コンテナの
プロセスから
の表示



Hello-Worldのポッドをクリーンナップ

ポッドのリスト表示 (終了済も含む)

```
kubectl get pod
```

ポッドの詳細表示

```
kubectl describe pod
```

ポッドの削除

```
kubectl delete pod hello-world
```

レッスン#1のまとめ

1. コンテナを実行するにはkubectrlコマンドを利用する
2. Kubernetesは、Docker Hub リポジトリから、イメージをダウンロードする
3. ポッドからコンテナを実行する



レッスン#2

基本要素を習得する

1. ポッド
2. コントローラー
3. サービス

ポッドとはコンテナの基本起動単位

シングルコンテナポッド



マルチコンテナポッド



ポッドのマニフェスト

マニフェストの基本とポッドAPI

Kubernetes API
の基本要素

1. apiVersion
2. kind
3. metadata
4. spec

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.12
```

APIの種別を識別

オブジェクトの名前

オブジェクト仕様

オブジェクトの生成と削除の共通のコマンド

マニフェストに指定したオブジェクトの生成と削除ができます。

オブジェクトの生成、変更

```
kubectl apply -f pod.yml
```

ファイル名
ディレクトリ
URL を設定できる

オブジェクトの削除

```
kubectl delete -f pod.yml
```

マニフェスト

pod.ymlファイル

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.12
```

オブジェクトを操作するためのコマンド

オブジェクトのリスト取得

```
kubectl get オブジェクト種類 [名前]
```

オブジェクトの詳細表示

```
kubectl describe オブジェクト種類 [名前]
```

オブジェクト種別

ポッド pod

デプロイメント deployment

サービス service

オブジェクトの削除

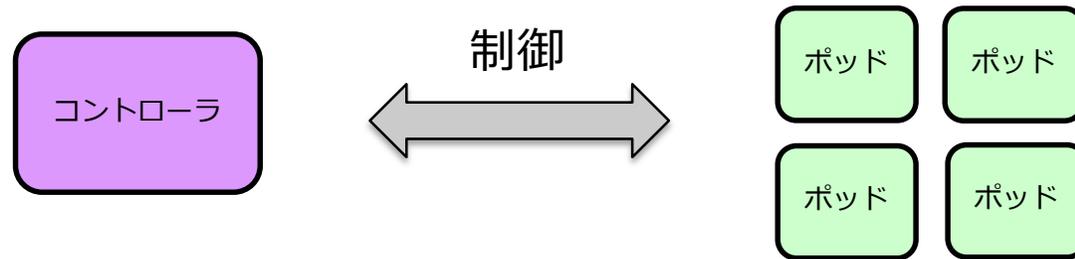
```
kubectl delete オブジェクト種類 [名前]
```

Kubectl コマンドの使い方を習得するために、次を実行してみましょう。

- ポッドを生成 `kubectl apply -f pod.yml`
- リスト表示 `kubectl get pod`
- ポッドの詳細表示 `kubectl describe pod NAME`
- ポッドの削除 `kubectl delete pod NAME`

コントローラーとは、ポッドを制御するオブジェクト

ワークロードの特性に応じて適切なコントローラーを選択します。



コントローラーの種類

- **Deployments**
- ReplicaSet
- ReplicationController
- StatefulSets
- Job
- CronJob
- DaemonSet
- Garbage Collection

コントロール内容

- 起動／停止／削除
- ポッド数
- 自己回復
- 永続ストレージ
- デプロイ先

ワークロードの種類

- ウェブ・アプリケーション
- バックエンド・サービス
- 処理時間の長いバッチ処理
- 定期実行のジョブ
- クラスタの基盤機能

デプロイメント（コントローラー）とは？

適用範囲の広いコントローラーです。



デプロイメントコントローラーの役割

- ・ポッド数を維持する様に制御
- ・ロールアウト／ロールバック
- ・スケール（ポッド数の増減）
- ・イメージの更新
- ・レプリカセットの指定

レプリカコントローラーの役割

- ・ポッドの規定数を維持

ポッドの役割

- ・コンテナの実行
- ・コンテナ異常時の再スタート
- ・IPアドレス取得と共有

デプロイメントのマニフェストの基本

deployment.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-deploy
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
        - name: nginx
          image: nginx:1.12
```

デプロイメント (オブジェクト) の名前

レプリカ数

デプロイメントとポッドの対応づけ

pod.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
    - name: nginx
      image: nginx:1.12
```

デプロイメント

デプロイメント
の仕様

ポッドテンプレート

ポッドを生成時
の雛形となる

デプロイメントからポッド起動して確認しましょう。

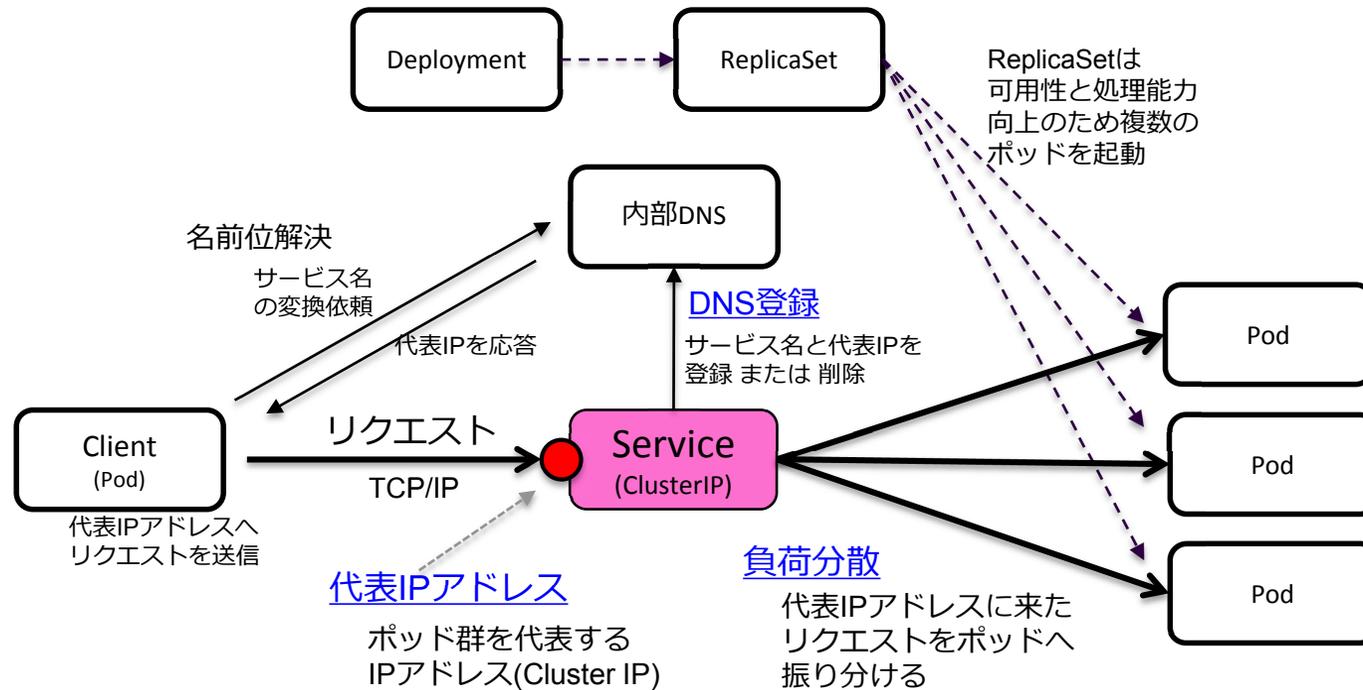
kubectl コマンドの使い方を習得するために、次を実行してみましょう。

デプロイメントの生成	<code>kubectl apply -f deploy.yml</code>
リスト表示	<code>kubectl get deploy</code>
詳細表示	<code>kubectl describe deploy NAME</code>
デプロイメントの削除	<code>kubectl delete deploy NAME</code>

デプロイメントコントローラーから作成されるポッドを確認してみてください。

ポッドのリスト	<code>kubectl get pod</code>
ポッドの詳細表示	<code>kubectl describe pod</code>

サービスとは、ポッドにアクセスするためのIPアドレス、DNS登録、負荷分散を担当します。



リクエストの転送先ポッドは、ラベルで指定する。

セレクターのラベルと一致するポッドに、リクエストを転送します。

内部DNSへ
登録します。

service.yml

```
apiVersion: v1
kind: Service
metadata:
  name: web-service
spec:
  selector:
    app: web
  ports:
  - protocol: TCP
    port: 80
```

deployment.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-deploy
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
      - name: nginx
        image: nginx:1.12
```

ラベル一致で
負荷分散の
転送先とする

サービスを生成して、確認してみましよう

サービスを生成して確認してみましよう。

サービスの生成	<code>kubectl apply -f service.yml</code>
リスト表示	<code>kubectl get service</code>
詳細表示	<code>kubectl describe service</code>
負荷分散対象ポッドのリスト	<code>kubectl get po -l app=web</code>

ポッドに対話シェルを起動して、サービス経由でポッドのNginxへアクセスしてみましよう。

対話型ポッド起動

```
kubectl run test -it --restart=Never --image=busybox sh
```

内部DNSに登録された名前アクセス

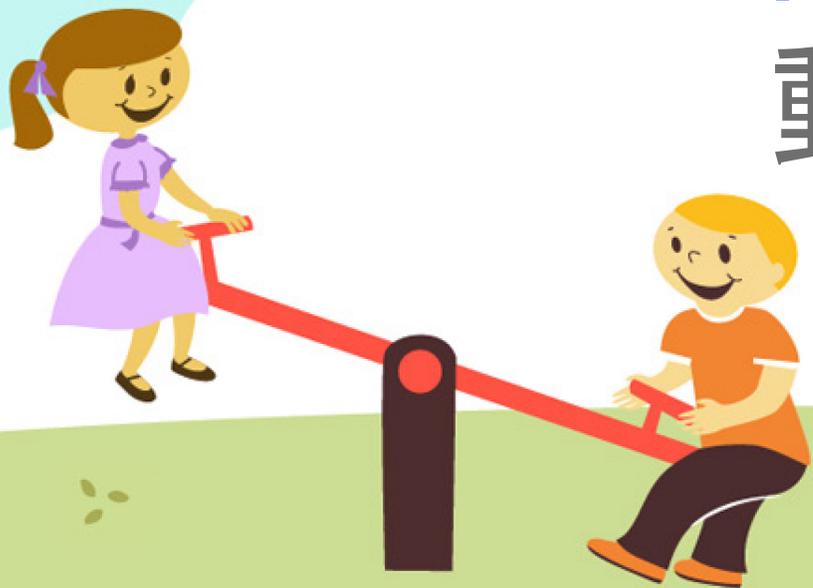
```
wget -O - http://web-service
```

レッスン#2のまとめ

1. ポッドは、コンテナを起動する基本単位である
2. コントローラーは、ポッドの制御を担当する
3. サービスは、ポッドへのアクセスをサポートする
 - 代表IPアドレス
 - 内部DNS登録
 - ロードバランス



ROKECT.CHATを
Kubernetesで
動かしてみましよう！



レッスン #3

永続ボリュームの準備

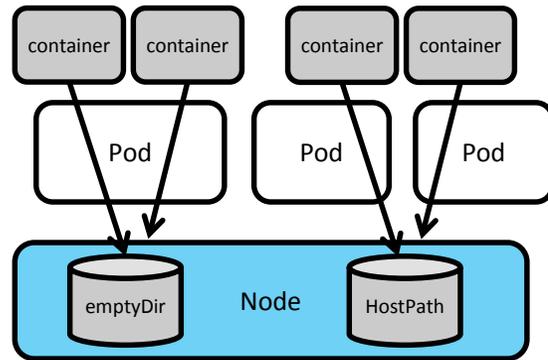
ROKECT.CHATのデータを
安全なストレージに保存する

大切なデータは、k8sクラスタ外に保存する

永続ボリュームの実装には、多くの種類があります。

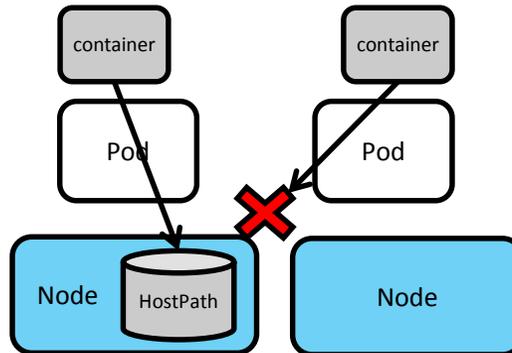
K8sクラスタの外部のSDSやストレージ装置を利用することが一般的です。

シングルノード・クラスタ



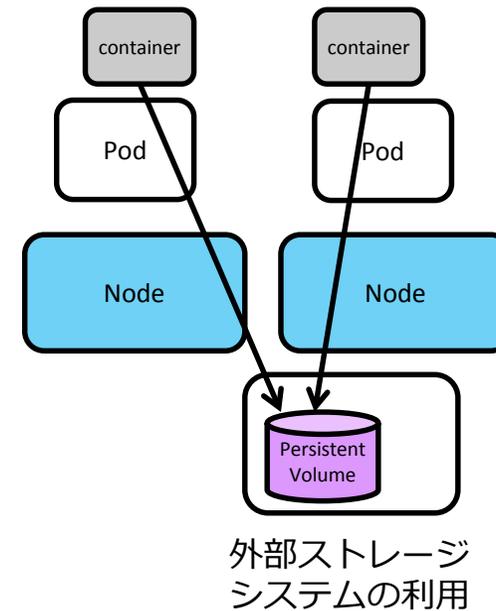
- emptyDirは同一ポッド内ファイル共有
- HostPathは同一ノード内ファイル共有

マルチノード・クラスタ



- ノード間でファイル共有は出来ない
- ノード停止でデータが喪失

マルチノード・クラスタ
+ ストレージシステム

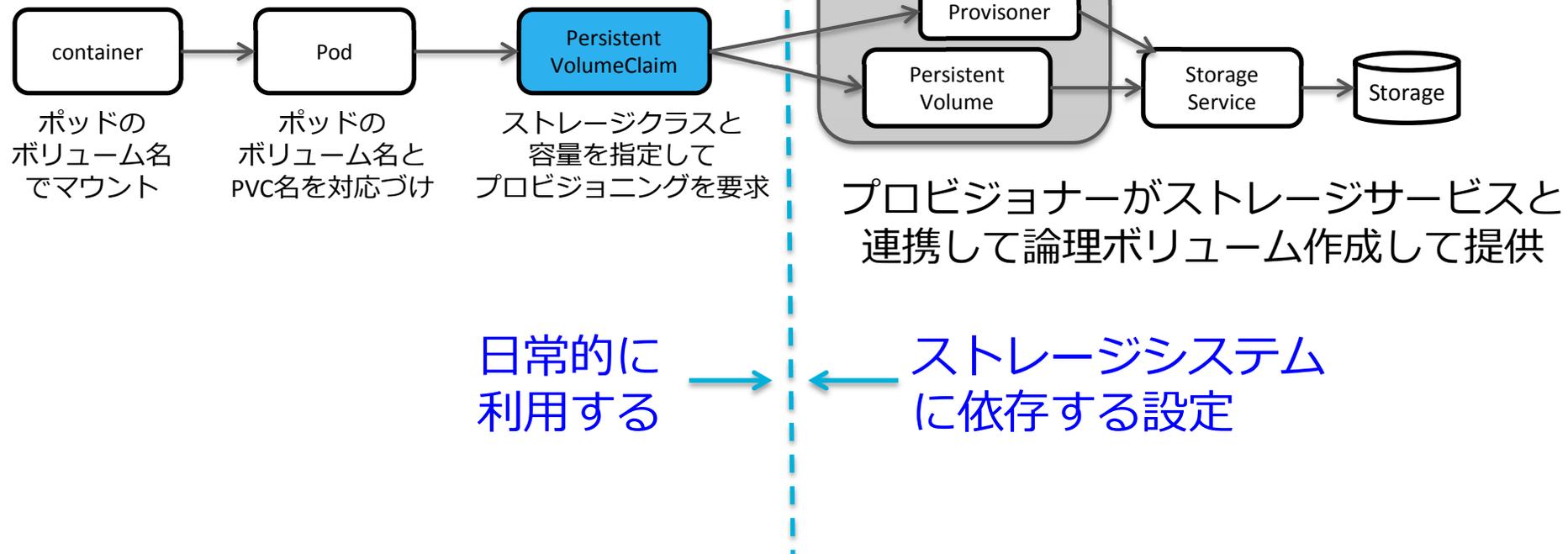


NFS
GlusterFS
EBS etc

ストレージシステムの実装を隠蔽する抽象化

永続ボリュームを利用する場合は、Persistent Volume Claim で論理ボリュームを確保して、ポッドからマウントする。

自動プロビジョニングするケース



MongoDBに割り当てるマニフェストを作成する

永続ボリュームを要求するマニフェスト

Kubernetes APIの
バージョンと種類

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: data1
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: standard
resources:
  requests:
    storage: 2Gi
```

オブジェクト名

ReadWriteOnce 1ノードからのみRW
ReadWriteMany 複数ノードからRW
ReadOnlyMany 複数ノードからReadOnly

EBSなどブロックストレージは、ReadWriteOnceになります。
また、NFSやGlusterFSでは、ReadWriteManyにできます。

ストレージ種類の指定

プロビジョニングする容量指定

永続ボリュームを作成して確認してみましよう

対話型ポッドからサービスを作成して、動作を確認しましょう。

永続ボリュームの作成	<code>kubectl apply -f pvc.yml</code>
リスト表示	<code>kubectl get pvc</code>
永続ボリュームの表示	<code>kubectl get pv</code>
ストレージクラスの表示	<code>kubectl get storageclass</code>
ストレージクラスの詳細	<code>kubectl describe storageclass</code>

Minikube の永続ボリュームの Provisioner を確認してみてください。

レッスン#3のまとめ

1. 永続ボリュームはk8sクラスタの外部に確保する
2. ストレージクラスは基盤固有のストレージ設定を隠蔽する
3. 永続ボリューム要求(PVC)でプロビジョニング実行する



レッスン #4

MongoDBの起動

ROKECT.CHATをKubernetesで動かしてみましよう。

MongoDBを実行するマニフェストを作成

MongoDBのマニフェスト

ポッドを
デプロイメントコントローラ
および、サービスと紐付ける
大切なラベルです。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongodb
spec:
  replicas: 1
  selector:
    matchLabels:
      app: db
  template:
    metadata:
      labels:
        app: db
    spec:
      containers:
        - name: mongodb
          image: mongo:3.0
          args:
            - --smallfiles
          volumeMounts:
            - name: vol
              mountPath: /data/db
          volumes:
            - name: vol
              persistentVolumeClaim:
                claimName: data1
```

コンテナの
ファイルシステムに
PVCをマウントする

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

How to use this image

Start a mongo server instance

```
$ docker run --name some-mongo -d mongo:tag
```

... where `some-mongo` is the name you want to assign to your container and `tag` is the tag specifying the MongoDB version you want. See the list above for relevant tags.

Connect to MongoDB from another Docker container

The MongoDB server in the image listens on the standard MongoDB port, `27017`, so connecting via container linking or Docker networks will be the same as connecting to a remote `mongod`. The following example starts another MongoDB container instance and runs the `mongo` command line client against the original MongoDB container from the example above, allowing you to execute MongoDB statements against your database instance:

```
$ docker run -it --link some-mongo:mongo --rm mongo mongo --host mongo test
```

... where `some-mongo` is the name of your original `mongo` container.

マウントポイントや引数などのAPI詳細は、
DockerHubを参照願います。

PVCをポッドに接続

次のコマンドを実行して、MongoDBを起動して、状態を確認しましょう。

MongoDBの起動	<code>kubectl apply -f mongodb.yml</code>
ポッドの起動状態の確認	<code>kubectl get pod</code>
ポッドのIPアドレスを表示	<code>kubectl get pod -o wide</code>
デプロイメントの確認	<code>kubectl get deploy</code>
起動失敗時の原因調査1	<code>kubectl logs POD-NAME</code>
起動失敗時の原因調査2	<code>Kubectl describe pod POD-NAME</code>
起動失敗時の原因調査3	<code>kubectl get events</code>

MongoDBを実行するマニフェストを作成

サービスのマニフェスト

サービスが、**負荷分散**の
リクエスト転送先の
ポッドを発見するため
のラベル

```
apiVersion: v1
kind: Service
metadata:
  name: db
spec:
  type: ClusterIP
  selector:
    app: db
  ports:
    - protocol: TCP
      port: 27017
```

内部DNSへ登録するサービス名
およびオブジェクト名です。

ポッドのクラスタを
代表するIPアドレスを
獲得する指定です。

内部DNSへ登録するサービス名
およびオブジェクト名です。

MongoDBのサービス(オブジェクト)を作成しましょう

マニフェストを適用してサービスのオブジェクトを作成

```
minion@iw01:~$ kubectl apply -f svc-mongodb.yml
service/db created
```

サービスの状態確認

```
minion@iw01:~$ kubectl get svc
NAME          TYPE          CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
db            ClusterIP     10.109.65.18    <none>           27017/TCP        1h
```

サービス名

内部ネットワークのIPアドレスで、
負荷分散の代表IPアドレスです。

MongoDBの
ポート番号

レッスン#4のまとめ

1. ラベルによって、コントローラ、サービス、ポッドを繋ぐ
2. DockerHubのガイドを参照してマニフェストを作成する
3. ポッドが一つでもサービスは必須である



レッスン #5

ROCKET.CHATの
ポッド(コンテナ)を起動

Internetからアクセスするためのサービスを作成

NodePortは、ノードのIPアドレス上に公開用ポートを開きます。

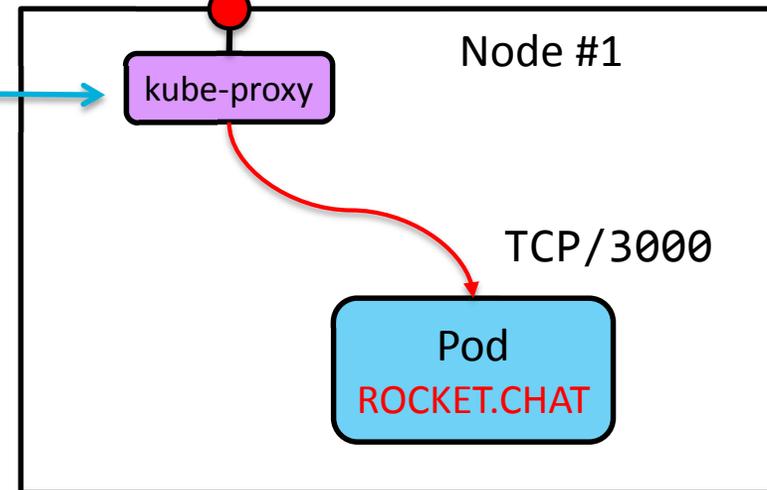
サービスのマニフェスト

```
apiVersion: v1
kind: Service
metadata:
  name: rocket
spec:
  selector:
    app: rocket
  ports:
    - protocol: TCP
      port: 3000
      nodePort: 31000
  type: NodePort
```

サービス

Node#1 IP address
+ Port No (NodePort)
TCP/31000

クラスタ外からの
リクエスト



ROCKET.CHATを実行するマニフェストを作成

DockerHubのROCKET.CHATのガイドに従って、YAMLを準備します。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rocket
spec:
  replicas: 1
  selector:
    matchLabels:
      app: rocket
  template:
    metadata:
      labels:
        app: rocket
    spec:
      containers:
        - name: rocket
          image: rocket.chat
          env:
            - name: ROOT_URL
              value: http://仮想サーバーのIP:31000/
```

<https://hub.docker.com/r/library/rocket.chat/>



How to use this image

First, start an instance of mongo:

```
$ docker run --name db -d mongo:3.0 --smallfiles
```

Then start Rocket.Chat linked to this mongo instance:

```
$ docker run --name rocketchat --link db -d rocket.chat
```

This will start a Rocket.Chat instance listening on the default Meteor port of 3000 on the container.

If you'd like to be able to access the instance directly at standard port on the host machine:

```
$ docker run --name rocketchat -p 80:3000 --env ROOT_URL=http://localhost --link db -d r
```

Then, access it via `http://localhost` in a browser. Replace `localhost` in `ROOT_URL` with your own domain name if you are hosting at your own domain.

If you're using a third party Mongo provider, or working with Kubernetes, you need to override the `MONGO_URL` environment variable:

```
$ docker run --name rocketchat -p 80:3000 --env ROOT_URL=http://localhost --env MONGO_UR
```

ROCKET.CHATの起動

ROCKET.CHATをデプロイして、起動を確認します。

ROCKET.CHATのポッド起動	<code>kubectl apply -f deploy-rocket.yml</code>
起動状態の確認	<code>kubectl get -f deploy-rocket.yml</code>
ROCKET.CHATのサービス作成	<code>kubectl apply -f svc-rocket.yml</code>
サービスの作成状態確認	<code>kubectl get -f svc-rocket.yml</code>

ROCKET.CHATのアクセステスト

ブラウザから次のURLをアクセスして、ROCKET.CHATの初期化ウィザードを表示して、初期設定を完了させます。

`http://仮想サーバーのIPアドレス:31000/`

自己回復のテスト

- ブラウザでROBOT.CHAT表示した状態で、下記を実施しましょう。
 - ROCKET.CHATのポッドを削除
 - MongoDBのポッドを削除
 - チャットに書き込んで、データが失われないことも確認してみましょう。

ポッドのリスト	<code>kubectl get pod</code>
ROCKET.CHATのポッドを指定して削除	<code>kubectl delete pod rocket-XXXXXXXXXX-YYYYY</code>
MongoDBのポッドを指定して削除	<code>kubectl delete pod mongodb-XXXXXXXXXX-YYYYY</code>

レッスン#5のまとめ

1. NodePortは、ノードのIPアドレスでアプリケーションを公開する
2. ポッドが削除されても、デプロイメントコントローラが瞬時に代替ポッドを起動する
3. DockerとKubernetesを利用することで、コンテナ化されたアプリケーションが運用できる

