kubernetes勉強会

(基本編)

kubernetes(k8s)とは?

• Kubernetes は**コンテナ化されたアプリケーション**のデプロイ、スケーリングなどの管理を自動化するためのプラットフォーム(コンテナオーケストレーションエンジン)です。

コンテナとは?

• OS上に「独立したサーバーと同様の振る舞いをする区画」のこと。 カーネルなどの機能を使って、OSのプロセスとして起動する。

@boxbg-orange text-white rounded demo-box-pad

Dockerコンテナ

Dockerコンテナ http://image.itmedia.co.jp/ait/articles/1701/30/wi-docker01002.png

コンテナ ≠ 仮想ホスト (VM)

※コンテナは仮想ホスト(VM)のプロセス単位に近い

+++

参考

DockreFileの書き方を学ぶ

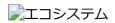
 改めてDockerfileのベストプラクティスを振り返ろう https://www.slideshare.net/ssuser1f3c12/introduce-that-best-practices-for-writing-dockerfiles

k8sで何ができるか?

- 複数のKubernetes Node の管理 |
- コンテナのスケジューリング |
- ローリングアップデート |
- スケーリング/オートスケーリング |
- コンテナの死活監視 |
- 障害時のセルフヒーリング |
- サービスディスカバリ |
- ロードバランシング |
- データの管理 |

- ワークロードの管理 |
- □グの管理 |
- Infrastructure as Code |
- その他エコシステムとの連携や拡張 (下ページ) |

+++



アーキテクチャ



kubernetes リソース

- Workloads @size16px
- Discovery & LB @size16px
- Config & Storage @size16px
- Cluster @size16px
- Metadata @size16px

Workloads リソース

- Pod
- ReplicationController (廃止)
- ReplicaSet
- Deployment
- DaemonSet
- StatefulSet
- Job
- CronJob

Workloadsの階層構造

- Pod ---> ReplicaSet ---> Deployment
- Pod ---> DemonSet
- Pod ---> StatefulSet
- Pod ---> Job ---> CronJob

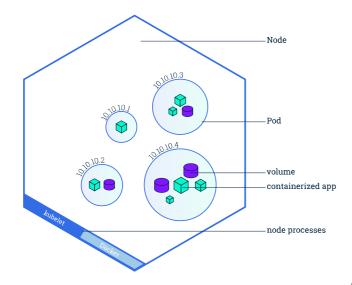
@box[bg-orange text-white rounded demo-box-pad](Pod が1サーバに相当。コンテナが1プロセスに相当。)

Pods-overview



 https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/#podsoverview

Node-overview



https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/#node-overview

Discovery & LB リソース

- Service |
 - ∘ ClusterIP ★
 - 。 ExternalIP (ClusterIP の一種)
 - NodePort ★
 - o LoadBalancer
 - Headless (None)
 - o ExternalName
 - o None-Selector
- Ingress ★ |

Service の役割

- L4 LoadBalancing
- クラスタ内DNSによる名前解決
- ラベルを利用したPodのサービスディスカバリ

Ingress の役割

- L7 LoadBalancing
- HTTPS終端
- パスベースルーティング

ClusterIP



kind: Service \mathcal{O} type: ClusterIP

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

# ClusterIP Serviceを作成
$ kubectl apply -f clusterip_sample.yml
```

@2 @6

NodePort



kind: Service ∅ type: NodePort

```
apiVersion: v1
kind: Service
metadata:
 name: sample-nodeport
spec:
 type: NodePort
 ports:
    - name: "http-port"
      protocol: "TCP"
      port: 8080
      targetPort: 80
      nodePort: 30080
  selector:
    app: sample-app
# NodePort Serviceの作成
kubectl apply -f nodeport_sample.yml
```

@2 @6

Ingress



Config & Storage リソース

- Config
 - Secret 機密情報などを管理する
 - o ConfigMap 単純なKey-Value値や設定ファイルなどは、ConfigMapで管理する
- Storage
 - PersistentVolumeClaim PersistentVolumeリソースの中から「xxxGBの領域ちょうだい!」 と要求するためのリソース。
- volume k8sノードのstrage相当。 (抽象化されておらず、直接ノードのディレクトリを指定する)
 - o EmptyDir
 - HostPath
 - nfs などのvolumeプラグインがある。

+++

volume は POD定義で直接指定する。

apiVersion: v1 kind: Pod metadata:

name: sample-hostpath

```
spec:
  containers:
    - image: nginx:1.12
    name: nginx-container
    volumeMounts:
    - mountPath: /srv
        name: hostpath-sample
    volumes:
    - name: hostpath-sample
    hostPath:
        path: /data
        type: DirectoryOrCreate

$ kubectl apply -f hostpath-sample.yml
```

@2 @12-16 @9-11

+++

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: sample-pv
  labels:
    type: nfs
    environment: stg
spec:
  capacity:
    storage: 10G
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  storageClassName: slow
  mountOptions:
    - hard
  nfs:
    server: xxx.xxx.xxx.xxx
    path: /nfs/sample
$ kubectl create -f pv_sample.yml
```

@2

persistentVolumeClaim
 PersistentVolumeリソースの中から「xxxGBの領域ちょうだい!」と要求するためのリソース。



Cluster リソース

- Node
- Namespace
- PersistentVolume
- ResourceQuota アクセス制御
- ServiceAccount
- Role
- ClusterRole
- RoleBinding
- ClusterRoleBinding
- NetworkPolicy

Node

• yamlは作らないけど、表示は頻繁に行うリソース。

kubectl get node

Namespace

- 仮想的な Kubernetes クラスタの分離機能。
- 初期状態で
 - o default
 - o kube-system
 - 。 kube-public の 3 種類のNamespace がある。

PersistentVolume

• 前の方でやった。

ResourceQuota

- 各Namespace ごとに、すなわち仮想Kubernetes クラスタごとに利用可能なリソースを制限することが可能。
 - 「作成可能なリソース数の制限」
 - 「リソース使用量の制限」

「作成可能なリソース数の制限」

apiVersion: v1 kind: ResourceQuota

metadata:

name: sample-resourcequota

namespace: default

spec: hard:

作成可能なリソースの数 count/configmaps: 10

@[8-9]

「リソース使用量の制限」

apiVersion: v1 kind: ResourceQuota

metadata:

name: sample-resourcequota-count-new

namespace: default

spec: hard:

作成可能なリソースの数 count/deployments.apps: 10 count/replicasets.apps: 10

count/deployments.extensions: 10

@[8-10]

ServiceAccount

- Pod で実行されるプロセスのために割り当てるアカウント。
- ServiceAccount はNamespace に紐づくリソースです。
 - ※ 指定しない場合はdefault ServiceAccount

Role

• どういった操作を許可するか、Namespace単位で定める。 ※ RBAC (Role Based Access Control)で権限管理する。(後述)

ClusterRole

• どういった操作を許可するか、Cluster 単位で定める。 ※ RBAC (Role Based Access Control) で 権限管理する。(後述)

RoleBinding

• roleRef で紐づけるRole を、subjects に紐づけるServiceAccount を指定。

RoleBindingの定義

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: sample-rolebinding

namespace: default

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: Role

name: sample-role

subjects:

- kind: ServiceAccount
name: sample-serviceaccount

namespace: default

ClusterRoleBinding

• roleRef で紐づけるClusterRole を、subjects に紐づけるServiceAccount を指定。

NetworkPolicy

• NetworkPolicy は、Kubernetes クラスタ内でPod 同士が通信する際のトラフィックルールを規定 するもの。(flanelでは使えない) ※ NetworkPolicy を利用しない場合、クラスタ内の全てのPod 同士で通信を行うことが可能

Metadata リソース

- LimitRange
- HorizontalPodAutoscaler
- PodDisruptionBudget
- CustomResourceDefinition

小ネタ

• kubectlコマンドのパラメタ補完(必須!) https://kubernetes.io/docs/tasks/tools/install-kubectl/#enabling-shell-autocompletion

リンク集

• 今こそ始めよう! Kubernetes入門 記事一覧 https://thinkit.co.jp/series/7342

終わり