Bab 7

Kubernetes Fundamental - Deployment Local to Production Part 2

SEKOLAH DEVOPS CILSY

Detail Materi

Indikator :
Mengenal Namespaces, cara deploy NodeJS pada Kubernetes, Pengenalan Ingress

Modul Sekolah DevOps Cilsy

Penulis : Estu Fardani
Editor: Taufik Maulana, Iqbal Ilman Firdaus & Muhammad Fakhri A
Revisi Batch 9

Penerbit : **PT. Cilsy Fiolution Indonesia**
Web Site : https://cilsyfiolution.com , https://devops.cilsy.id

Daftar Isi

9.
Kubernetes Fundamental & Deployment Local to Production

Learning Outcomes

Setelah selesai mempelajari bab ini, peserta mampu :

1. Memahami Konsep dan Arsitektur Kubernetes
2. Mempersiapkan Service AWS untuk Integrasi Kubernetes
3. Memahami Konsep Ingress

Outline Materi

1. Pengenalan Namespace
2. Deploy NodeJs dan Ingress di Kubernetes
3. Setup Ingress

9.1. Pengenalan Namespace

Ketika bekerja dalam tim, maka ada baiknya untuk memisahkan resource sesuai kebutuhan setiap tim. Inilah dimana namespace masuk kedalam solusi tersebut. Namespace adalah sebuah kluster virtual dari kluster fisik dimana kita dapat mengalokasikan resource seperti RAM, CPU dan disk sesuai kebutuhan.

Ada 3 Namespace default yang sudah dibuat ketika sebuah kluster dibuat, yakni :

- default

  Namespace default untuk objek yang dibuat tanpa mencantumkan *namespace* pada spesifikasinya

- kube-system

  namespace yang digunakan untuk objek yang dibuat oleh sistem Kubernetes

- kube-public

  namespace ini dibuat secara otomatis dan dapat diakses oleh semua pengguna (termasuk yang tidak diautentikasi).Namespace ini disediakan untuk penggunaan klaster, jika beberapa resouce harus terlihat dan dapat dibaca secara publik di seluruh klaster

Fungsi Namespace

Berikut adalah beberapa fungsi penting dari Namespace di Kubernetes

- Namespaces membantu komunikasi *pod-to-pod* menggunakan Namespace yang sama.
- Namespace adalah *cluster virtual* yang dapat duduk di atas *cluster* fisik yang sama.

- Mereka memberikan pemisahan logis antara tim dan lingkungan

9.1.1. Melihat Namespace

Kita dapat melihat namespace yang adad menggunakan perintah **kubectl get ns** atau **kubectl get namespaces**.

```
controlplane $ kubectl get ns
NAME              STATUS   AGE
default           Active   56m
kube-public       Active   56m
kube-system       Active   56m
```

9.1.2. Membuat Namespace

Langkah awal untuk membuat namespace adalah untuk menentukan kebutuhan dari namespace tersebut. Misalnya namespace tersebut digunakan untuk tim Developer untuk aplikasi yang sedang dikembangkannya. Maka kita dapat membuat namespace dengan menggunakan perintah ***kubectl create ns [nama namespacenya].***

```
controlplane $ kubectl create ns devops
namespace/devops created
```

Kita juga dapat membuat namespace dari file yaml dengan perintah **kubectl apply -f [file namespacenya]**. Pertama tama, kita buat terlebih dahulu file bernama **ns-devops.yaml** dengan isi sebagai berikut

```
apiVersion: v1
kind: Namespace
metadata:
  name: devops
```

script dapat dilihat pada
https://gist.github.com/sdcilsy/c296293ec35b740df31b0b8e354826d2

Lalu kita buat namespace tersebut menggunakan perintah **kubectl apply -f [file namespacenya]**

```
controlplane $ kubectl apply -f ns-devops.yaml
namespace/devops created
```

kita bisa melihat hasil dari perintah tersebut dengan perintah **kubectl get ns.**

```
controlplane $ kubectl get ns
NAME              STATUS   AGE
default           Active   160m
devops            Active   43s
kube-node-lease   Active   160m
kube-public       Active   160m
kube-system       Active   160m
```

9.1.3. Menentukan Resource pada Namespace

Seperti yang sudah dijelaskan pada awal materi, bahwa namespace dapat digunakan untuk mengalokasikan resource kluster fisik ke kluster virtual. Ini dapat dilakukan dengan cara membuat Resource Quota yang terapkan pada namespace.

Pertama tama, kita buat dulu file konfigurasi Resource Quota dengan nama **rq-devops.yaml** yang akan dibind ke namespace yang sudah dibuat sebelumnya. Lalu kita masukan resource yang diinginkan.

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: mem-cpu
  namespace: devops
spec:
  hard:
    requests.cpu: "0.5"
```

```
    requests.memory: 512Mi
    limits.cpu: "1"
    limits.memory: 1Gi
```

script dapat dilihat di
https://gist.github.com/sdcilsy/c4d774c99925c49a9612ef82a9d33853

Pada file konfigurasi tersebut, ada namespace yang harus diisi terlebih dahulu. Karena kita menggunakan namespace yang sudah dibuat sebelumnya, maka kita tinggal memasukannya. Lalu untuk resource yang dialokasikan untuk namespace tersebut bisa kita atur sesuai kebutuhan.

Lalu kita eksekusi perintah **kubectl apply -f [file resource quotanya]**

```
controlplane $ kubectl apply -f rq-devops.yaml
resourcequota/mem-cpu created
```

Hasil dari resource quota ini bisa kita lihat dengan menggunakan perintah **kubectl describe quota -n devops**

```
controlplane $ kubectl describe quota -n devops
Name:            mem-cpu
Namespace:       devops
Resource         Used  Hard
--------         ----  ----
limits.cpu       0     1
limits.memory    0     1Gi
requests.cpu     0     500m
requests.memory  0     512Mi
```

9.1.4. Binding Pod ke Namespace

Setelah kita berhasil membuat namespace dan mengalokasikan resource ke namespace tersebut, langkah selanjutnya adalah membuat pod dan melakukan bind ke namespace yang sudah dibuat. Caranya adalah kita membuat file

konfigurasi pod dengan nama **pod-nginx.yaml** dan menuliskan namespace yang sudah dibuat, lalu berapa banyak resource yang dibutuhkan.

```
apiVersion: v1
kind: Pod
metadata:
  namespace: devops
  name: nginx-app
  labels:
     app: nginx-app
spec:
  containers:
  - name: nginx-app
    image: nginx:alpine
    ports:
    - containerPort: 80
    resources:
      limits:
        memory: "400Mi"
        cpu: "0.3"
      requests:
        memory: "300Mi"
        cpu: "0.2"
```

script dapat dilihat di
https://gist.github.com/sdcilsy/7e9898d9664ef758463e7b699b0a0fa8

kita sudah membuat pod yang yang ditautkan ke namespace devops dengan resource yang dipakai adalah ram sebesar 300Mb, dan cpu sebesar 0,2.

Dan jika kita melihat alokasi resource untuk namespace **devops** tersebut menggunakan perintah **kubectl describe quota -n devops**, maka akan terlihat perbedaan setelah membuat pod.

```
controlplane $ kubectl describe quota -n devops
Name:           mem-cpu
```

```
Namespace:       devops
Resource         Used   Hard
--------         ----   ----
limits.cpu       300m   1
limits.memory    400Mi  1Gi
requests.cpu     200m   500m
requests.memory  300Mi  512Mi
```

Jika kita merequest resource diatas limit yang sudah ditentukan, maka akan terjadi error. Kita buat lagi file konfigurasi pod dengan nama **pod-nginx-err.yaml** yang hampir mirip dengan yang sebelumnya dibuat, namun pada bagian resourcenya yang berbeda.

```
apiVersion: v1
kind: Pod
metadata:
  namespace: devops
  name: nginx-app-err
  labels:
     app: nginx-app-err
spec:
  containers:
  - name: nginx-app-err
    image: nginx:alpine
    ports:
    - containerPort: 80
    resources:
      limits:
        memory: "700Mi"
        cpu: "0.6"
      requests:
        memory: "600Mi"
        cpu: "0.6"
```

script dapat dilihat di

https://gist.github.com/sdcilsy/ca5ec49950038c7bb5511a774dfaa08e

Maka ketika kita masukan perintah **kubectl apply -f pod-nginx-err.yaml** akan menghasilkan output seperti berikut.

```
controlplane $ kubectl apply -f pod-nginx-err.yaml

Error from server (Forbidden): error when creating "pod-nginx-err.yaml": pods
"nginx-app-err" is forbidden: exceeded quota: mem-cpu, requested:
requests.cpu=600m,requests.memory=600Mi, used: requests.cpu=0,requests.memory=0,
limited: requests.cpu=500m,requests.memory=512Mi
```

Error tersebut terjadi karena request resource pada pod melebihi kapasitas yang tersedia.

9.2. Deploy Nodejs dan Ingress di K8S

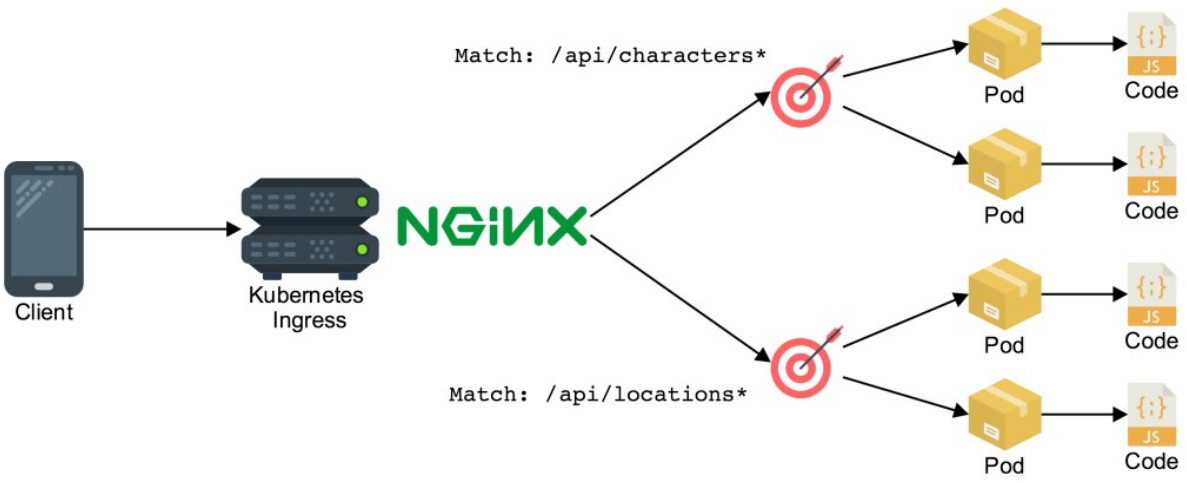
9.2.1. Apa itu Ingress ?

Di Kubernetes, Ingress adalah objek yang memungkinkan akses ke layanan Kubernetes dari luar cluster Kubernetes. Konfigurasi ini diatur dengan membuat kumpulan peraturan (*rules*) yang menentukan koneksi masuk mana yang boleh menjangkau layanan apa.

Konfigurasi ini memungkinkan untuk menggabungkan aturan perutean menjadi satu pintu. Misalnya, untuk mengatur alamat example.com/api/v1/ ke layanan api-v1, dan mengatur alamat example.com/api/v2/ ke layanan api-v2. Dengan Ingress, kita dapat mengatur ini tanpa membuat banyak Load Balancer atau mengekspos setiap layanan di *Node*. Karena kenaikan jumlah Load Balancer artinya menaikan biaya bulanan.

NGINX Ingress hanya merupakan salah satu implementasi Ingress Controller menggunakan NGINX. Selain NGINX, bisa juga menggunakan Traefik dan lainnya.

Berikut merupakan *desain* arsitekur yang akan kita buat :



9.2.2. Persiapan *Tools*

Pertama kita akan melakukan *clone repo* terlebih dahulu dengan menggunakan perintah berikut :

```
#git clone https://github.com/tuanpembual/nodejs-aws-workshop.git
#cd nodejs-aws-workshop/cd 7\ -\ Kubernetes\ \(kops\)
```

Setelah melakukan *clone repo* kita siapkan *image* API *characters* bisa menggunakan perintah berikut

```
#cd code/services/locations
#docker build -t tuanpembual/characters:latest
#docker push tuanpembual/characters:latest
```

Berikutnya deploy ke Kubernetes dengan memasukan perintah berikut :

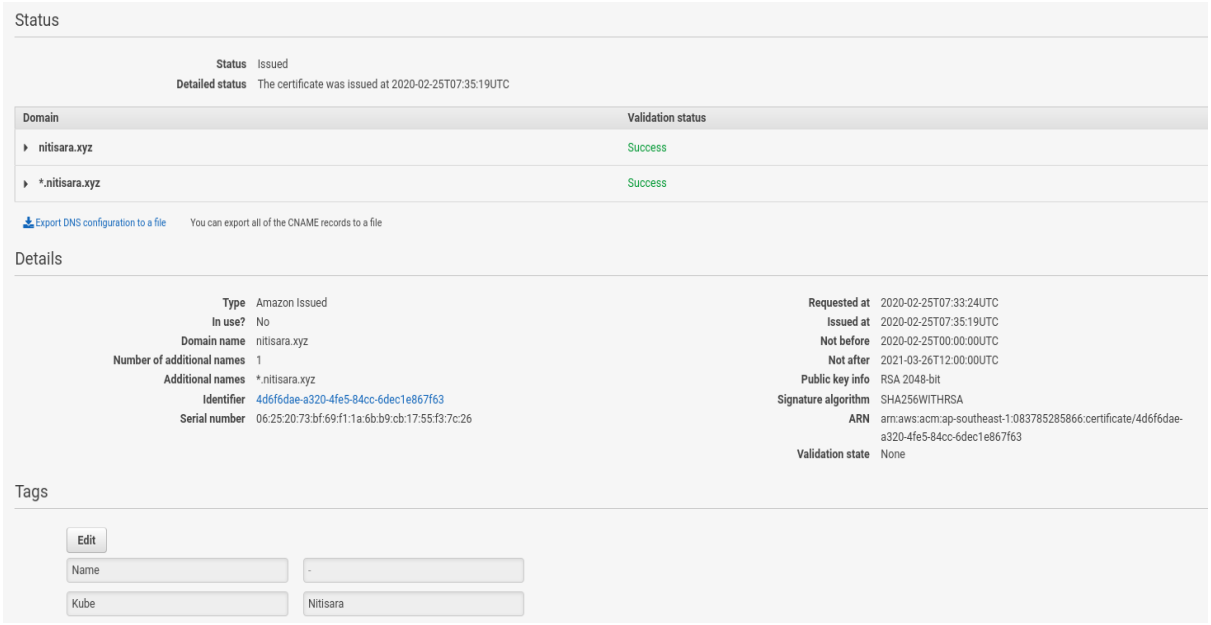
```
#cd recipes
#kubectl apply -f locations.yml
#kubectl apply -f characters.yml
```

9.3. *Setup* Ingress

Untuk membuat kedua api itu dapat diakses publik, dibutuhkan layanan proxy untuk menjembatani.

9.3.1. *Setup Domain* dan SSL

1. Buat SSL untuk ELB sesuai *domain* yang diinginkan pada *menu* ACM pada AWS *Console*. *Request a certificate* > *Public* > isi *domain*= nitisara.xyz dan *.nitisara.xyz (*setup wildcard subdomain*). Kemudian ikuti langkah untuk validasi. Validasi saya memilih dengan DNS. Cukup masukan CNAME sesuai yang dikeluarkan oleh dashboard
2. Verifikasi jika *domain* sudah aktif. Catat bagian ini. **arn:aws:acm:ap-southeast-1:083785285866:certificate/4d6f6dae-a320-4fe5-84cc-6dec1e867f63**



9.3.2. *Deploy* Ingress

Pada langkah ini kita akan menggunakan ELB *layer* 7 dengan menggunakan perintah :

```
#cd 7\ -\ Ingress\ NGINX\ \(ELB\)/
#cd code/ingress
#kubectl apply -f mandatory.yaml
#kubectl apply -f service-l7.yaml
#kubectl apply -f patch-configmap-l7.yaml
```

9.3.3. Pengaturan *Domain* di Ingress *Rules*

Selanjutnya kita akan melakukan konfigurasi pada file **rule-ingress-nginx.yaml**, lalu kita ubah pada bagian domain bisa dilihat pada konfigurasi berikut.

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: s-ingress
  annotations:
      ## Untuk menaikan batas upload file
```

```
      nginx.org/client-max-body-size: "10m"
      ## Memaksa jalan di https
      nginx.ingress.kubernetes.io/force-ssl-redirect: "true"
spec:
  rules:
  ## Service Location n Characters
  - host: ingress.nitisara.xyz
    http:
      paths:
      - path: /api/characters
        backend:
          serviceName: characters-service
          servicePort: 8081
      - path: /api/locations
        backend:
          serviceName: locations-service
          servicePort: 8081
```

script dapat dilihat di

https://gist.github.com/sdcilsy/20a21c11811081bc19167281ce3b00e8

Save dan tutup text editor nano dengan menekan CTRL+X lalu tekan Y dan enter.

Jalankan file konfigurasi menggunakan perintah sebagai berikut

```
#kubectl apply -f rule-ingress-nginx.yaml
```

9.3.4. *Domain* di Route53

Layanan DNS yang disediakan oleh AWS bernama AWS Route 53 adalah salah satu layanan paling terkenal, andal dan hemat biaya untuk mengelola dan memelihara domain.

1. Buka menu ELB. Cari domain ELB, kemudian salin
2. Buka menu Route53. Kemudian di bagian domain, tambahkan host baru sebagai CNAME: ingress.nitisara.xyz = domainelb

3. Buka browser, test domain baru. ingress.nitisara.xyz/api/locations

Referensi

https://tuanpembual.wordpress.com/2019/03/08/implementasi-ingress-elb-dan-ssl-bagian-6-binar-academy/

https://kubernetes.github.io/ingress-nginx/deploy/

https://github.com/nathanpeck/nodejs-aws-workshop/tree/master/6%20-%20Kubernetes%20(kops)

https://kubernetes.io/id/docs/concepts/overview/working-with-objects/namespaces/

https://kubernetes.io/id/docs/concepts/policy/resource-quotas/

https://nirmata.com/2018/11/30/kubernetes-namespaces/