POLITECHNIKA
LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI
I INFORMATYKI

Programowanie Full-Stack w Chmurze Obliczeniowej

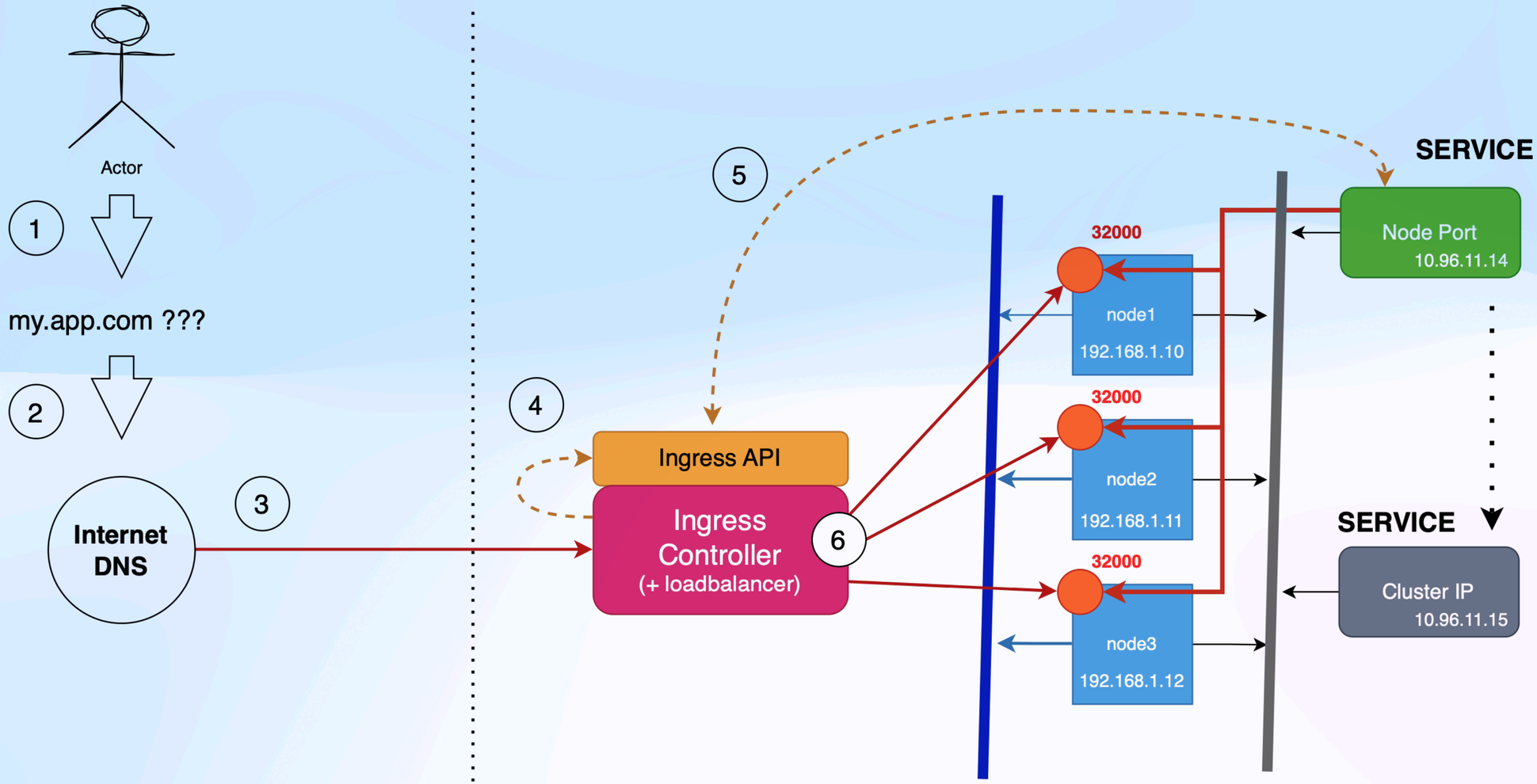
Instrukcja nr 9

Dobre praktyki powiązane z rozwiązaniem zadania 1 Wykorzystanie dostępu do zasobów w oparciu o Ingress. Podział implementacji Ingress i zasady ich konfiguracji.

Dr inż. Sławomir Przyłucki
s.przylucki@pollub.pl

Katedra Informatyki

Ingress - przypomnienie zasady działania w klastrze K8s



Zasady funkcjonowania Ingress - cz. I

Każda reguła definiowana dla obiektu Ingres zawiera następujące elementy:

- **Nazwa hosta - opcjonalnie** (nazwa DNS). Jeśli nie określono żadnego hosta, reguła dotyczy każdego przychodzącego ruchu HTTP.
- **Lista ścieżek** (np. /testpath). Każda ścieżka ma/wskazuje na określony backend. Ścieżki mogą być prezentowane jako wyrażenie regularne POSIX
- **Backend.** Może to być usługa (Service) lub inny zasób (Resource) w K8S.

Backend powiązany z usługą oraz backend powiązany z zasobem - WYKLUCZAJĄ się wzajemnie

Dobre praktyka nakazuje również konfigurować tzw. domyślny backend. Jego zadaniem jest obsługa ruchu przychodzącego, który nie pasuje do zdefiniowanych reguł

Referencyjna dokumentacja Ingress jest dostępna w dokumentacji Kubernetesa, pod adresem:

https://kubernetes.io/docs/concepts/services-networking/ingress/

Zasady funkcjonowania Ingress - cz. II

Ścieżki definiowane w ramach reguły Ingress są analizowane zgodnie z ustawieniami **Ingress pathType**. W praktyce wykorzystywane są dwa typy, które definiują sposób interpretowania przychodzących żądań:

- **Exact pathType** - definiuje ścisłą Interpretację ścieżki jako podstawę określania zgodność ścieżki w żądaniu ze ścieżką w regule, np. jeśli zdefiniowana jest ścieżka `/foo` a w żądaniu podano ścieżkę `/foo/` to według tego typu brak jest zgodności.
- **Prefix pathType** - definiuje Interpretację opartą o początek ścieżki, np. jeśli zdefiniowana jest ścieżka `/` to każde żądanie będzie uznane za zgodne. Analogicznie, jeśli reguła zawiera ścieżkę `/foo` to w takim wypadku zarówno `/foo` jak i `/foo/` będą uznane za zgodne

Proszę KONIECZNIE zapoznać się z przykładami ilustrującymi powyższe reguły oraz zasadami definiowania tzw. **hostname wildcards**. Pozwoli to na uniknięcie wielu błędów, których wykrycie może czasem zabierać sporo czasu.

https://kubernetes.io/docs/concepts/services-networking/ingress/#examples

Typy Ingres - cz. I - single backend

Najprostszym typem usługi Ingress jest tzw. Ingress wspierany przez pojedynczy Backend

```
kubectl create ingress single /
--rule="/files=fileservice:80"
```

- **W praktyce możliwe jest zrealizowanie omawianego typu usługi Ingress poprzez utworzenie domyślnego Backendu (ang. DefaultBackend).**

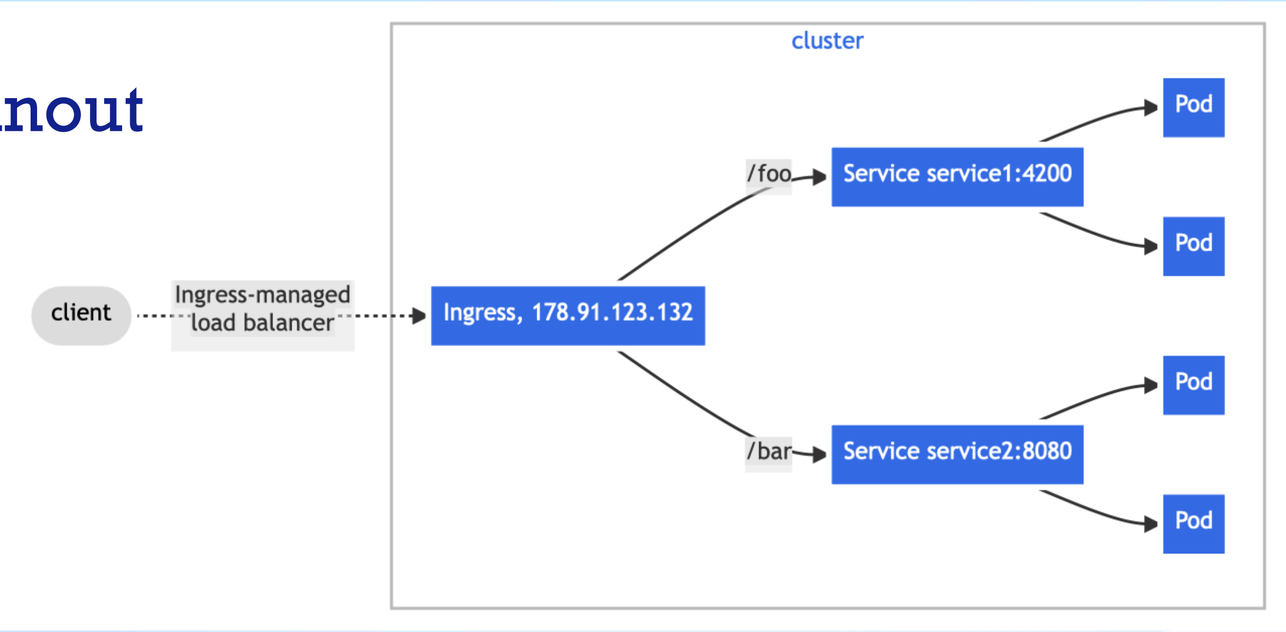
- **Jeśli konfiguracja usługi Ingress jest bardziej złożona to również pojedynczy Backend w postaci *defaultBackend* powinien/może być zastosowany do obsługi żądań niezgodnych ze zdefiniowanymi regułami. Definiowany jest w jako dedykowany obiekt w sekcji *spec***

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  creationTimestamp: null
  name: single
spec:
  rules:
  - http:
      paths:
      - backend:
          service:
            name: fileservice
            port:
              number: 80
        path: /files
        pathType: Exact
status:
```

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: single
spec:
  defaultBackend:
    service:
      name: fileservice
      port:
        number: 80
```

Typy Ingres - cz. II - simple fanout

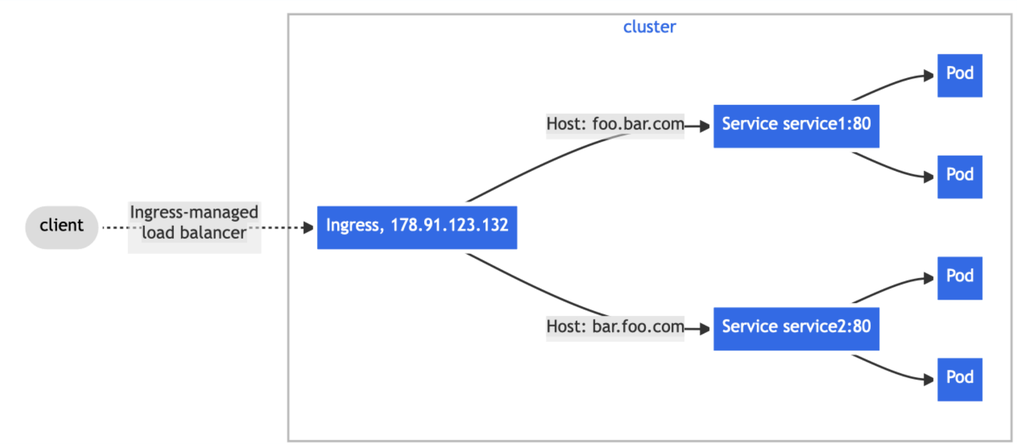
```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  creationTimestamp: null
  name: fanout
spec:
  rules:
  - http:
      paths:
      - backend:
          service:
            name: fileservice
            port:
              number: 80
        path: /files
        pathType: Exact
      - backend:
          service:
            name: dbservice
            port:
              number: 80
        path: /db
        pathType: Exact
status:
  loadBalancer: {}
```



W schemacie **simple fanout** definiowane są dwie lub więcej reguł definiujących ścieżki i przypisane im backendy

```
kubectl create ingress fanout
--rule="/files=fileservice:80"
--rule="/db=dbservice:80"
```

Typy Ingres - cz. III - name-based virtual hosting



W schemacie **name-based virtual hosting** definiowane są dwie lub więcej zasad, które zawierają w nagłówku żądania określoną nazwę hosta

```
kubectl create ingress multihost
—-rule=”lab91.com/files*=fileservice:80”
—-rule=”lab92.com/data*=dataservice:80”
```

Proszę zwrócić uwagę, że wykorzystanie wildcards w definicji zasady spowodowało zmianę `pathType` z Exact na Prefix

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  creationTimestamp: null
  name: multihost
spec:
  rules:
  - host: lab91.com
    http:
      paths:
      - backend:
          service:
            name: fileservice
            port:
              number: 80
        path: /files
        pathType: Prefix
  - host: lab92.com
    http:
      paths:
      - backend:
          service:
            name: dataservice
            port:
              number: 80
        path: /data
        pathType: Prefix
status:
  loadBalancer: {}
```

Wykorzystanie klas usługi Ingress

Usługa Ingress może być realizowane przez różne kontrolery, często z różną konfiguracją. Każda definicja Ingress powinien określać tzw. klasę czyli odwołanie do zasobu IngressClass, który zawiera dodatkową konfigurację, w tym nazwę kontrolera, który powinien implementować tą konkretną klasę.

```
~                                                    minikube on minikube (default)  uses desktop-linux
❯ kubectl api-resources | grep networking
ingressclasses                                 networking.k8s.io/v1                   false        IngressClass
ingresses                          ing         networking.k8s.io/v1                   true         Ingress
networkpolicies                    netpol      networking.k8s.io/v1                   true         NetworkPolicy
```

Zasięg definicji IngressClass (cały klaster czy tylko dana przestrzeń nazw) jest konfigurowalny. Domyślnie zasięg obejmuje cały klaster.

https://kubernetes.io/docs/concepts/services-networking/ingress/#ingressclass-scope

Istnieje też możliwość zdefiniowania domyślnej IngressClass. —-> W przypadku braku definicji chęci wykorzystania konkretnego kontrolera Ingress, wykorzystywane będzie ten, zdefiniowany jako domyślny IngressClass

```yaml
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  labels:
    app.kubernetes.io/component: controller
  name: nginx-example
  annotations:
    ingressclass.kubernetes.io/is-default-class: "true"
spec:
  controller: k8s.io/ingress-nginx
```

Specyfika wykorzystania usługi Ingress - cz. I

1 Utwórzmy Deployment o nazwie web na bazie obrazu nginx z trzema replikami oraz zdefiniujmy dla niego obiekt Service typu NodePort

```
~
❯ kubectl get deploy,svc
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/web   3/3     3            3           15m

NAME                 TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
service/kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP        7h35m
service/web          NodePort    10.103.188.179   <none>        80:30458/TCP   43s
```

2 Należy utworzyć obiekt Ingress pozwalający na dostęp do aplikacji *web* za pomocą żądania zawierającego nazwę hosta *lab9.info*

```
~
❯ kubectl create ingress web-ingress --rule="lab9.info/=web:80"
ingress.networking.k8s.io/web-ingress created
```

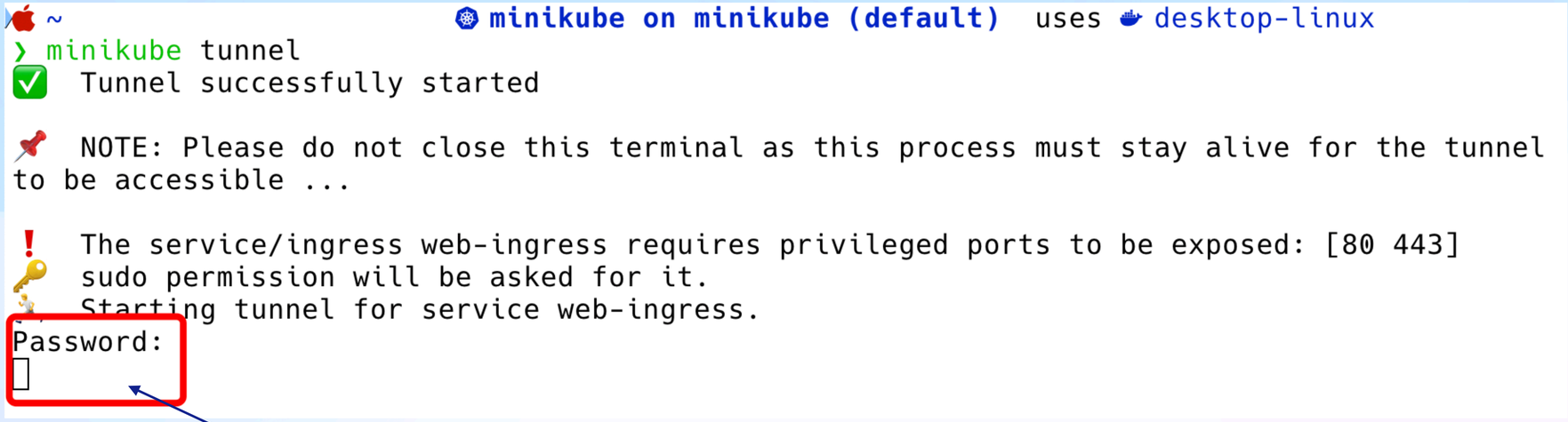
```
~
❯ kubectl get ing
NAME          CLASS   HOSTS        ADDRESS   PORTS   AGE
web-ingress   nginx   lab9.info              80      10s
```

```
~
❯ kubectl describe ing web-ingress
Name:             web-ingress
Labels:           <none>
Namespace:        default
Address:          192.168.49.2
Ingress Class:    nginx
Default backend:  <default>
Rules:
  Host         Path  Backends
  ----         ----  --------
  lab9.info
               /   web:80 (10.244.120.85:80,10.244.120.91:80,10.244.120.90:80)
Annotations:  <none>
Events:
  Type    Reason  Age                 From                      Message
  ----    ------  ----                ----                      -------
  Normal  Sync    17s (x3 over 77s)   nginx-ingress-controller  Scheduled for sync
```

Specyfika wykorzystania usługi Ingress - cz. II

3

Aby uzyskać dostęp zewnętrzny do aplikacji web za pomocą usługi Ingress należy wcześniej utworzyć tunel do serwisu Nodeport. Jego utworzenie należy wykonać w **oddzielnym terminalu**.



Należy podać hasło root-a, pozostawić terminal otwarty a następnie przejść do poprzedniego terminala, w którym będzie można kontynuować realizację zadań

Specyfika wykorzystania usługi Ingress - cz. III - metoda A oraz B

4A

```
~
❯ curl -H "Host: lab9.info" http://localhost
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>
```

W wyniku polecenia `curl -H "Host: lab9.info" http://localhost`, ustawiamy nagłówek (host header) w żądaniu na *lab9.info*. Nagłówek i ścieżka żądania są zgodne z regułą zdefiniowaną w Ingress *web-ingress*. Dzięki tunelowi, żądanie trafia do kontrolera NGINX a następnie do Service *web*

4B

```
~
❯ curl --resolve "lab9.info:80:127.0.0.1" -i http://lab9.info
HTTP/1.1 200 OK
Date: Sun, [redacted] GMT
Content-Type: text/html
Content-Length: 615
Connection: keep-alive
Last-Modified: Tue, [redacted] GMT
ETag: "6745ef54-267"
Accept-Ranges: bytes

<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
```

Flaga `--resolve` informuje polecenie `curl` by rozwinąć hostname *lab9.info* na adres IP 127.0.0.1 oraz port 80. Natomiast opcja `-i` poleca poleceniu `curl` by wykorzystał nagłówek HTTP jako nagłówek umieszczany w danych wyjściowych polecenia.

Działanie dostępu do aplikacji w oparciu o Ingress - cz. I

1. Utwórzmy kolejny Deployment o nazwie *web2* na bazie obrazu *spcr.spg51.dev/lab/hello-app:1.0 z trzema* replikami. Aplikacja „nasłuchuje” na porcie 8080

2. Dla *web2* należy utworzyć Service typu NodePort z parametrem `--port=8080`

```
~/Labs
❯ kubectl get deploy,svc -l "app in (web,web2)"
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/web     3/3     3            3           4d18h
deployment.apps/web2    3/3     3            3           30m

NAME            TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
service/web     NodePort    10.103.188.179   <none>        80:30458/TCP     4d18h
service/web2    NodePort    10.98.206.86     <none>        8080:31516/TCP   32s
```

3’ Należy dodać nową aplikację (web2) do reguł w istniejącym obiekcie Ingress.
Możliwe są 3 „szybkie” rozwiązania:

1. **Usunąć obecny obiekt Ingress i utworzyć nowy, skonfigurowany dla obu aplikacji** `kubectl create …..`
2. **Obecną konfigurację obiektu Ingress można wyedytować w oparciu o polecenie** `kubectl edit ….`
3. **Poddać edycji istniejący manifest dla Ingress, zmienić konfigurację dla 2 aplikacji i uruchomić ponownie** `kubectl apply ….`

Działanie dostępu do aplikacji w oparciu o Ingress - cz. II

3”

Ponieważ nie utworzyliśmy wcześniej manifestu (wykorzystywano podejście imperatywne) to należy teraz wygenerować manifest dla działającego obiektu Ingress

```
~                                          minikube on minikube (default)  uses desktop-linux
❯ kubectl get ing
NAME          CLASS   HOSTS        ADDRESS         PORTS   AGE
web-ingress   nginx   lab9.info    192.168.49.2    80      3h55m
~                                          minikube on minikube (default)  uses desktop-linux
❯ kubectl get ing web-ingress -o yaml > multiweb.yaml
```

4’

W utworzonym pliku *multiweb.yaml* usuwamy niepotrzebne elementy (zaznaczone czerwoną ramką)

```
3  metadata:
4    creationTimestamp: "2023-11-30T15:17:37Z"
5    generation: 1
6    name: web-ingress
7    namespace: default
8    resourceVersion: "163052"
9    uid: 00f45f0d-e077-4d16-86fb-e4ec44f74d8b
```

```
23 status:
24   loadBalancer:
25     ingress:
26     - ip: 192.168.49.2
```

Jednocześnie proszę zamienić „`pathType: Exact`” na „`pathType: Prefix`”

4”

Po „oczyszczeniu” pliku należy dodać deklarację nowej reguły dla aplikacji *web2*

Działanie dostępu do aplikacji w oparciu o Ingress - cz. III

4” cd

Do pliku *multiweb.yaml* należy dodać nową regułę (ramka niebieska)

5 Ponieważ wcześniej nie użyto metody deklaratywnej to przed uruchomieniem zmodyfikowanego Ingress, poprzedni obiekt należy usunąć

```
~/Labs
❯ kubectl delete ing web-ingress
ingress.networking.k8s.io "web-ingress" deleted
~/Labs
❯ kubectl apply -f multiweb.yaml
ingress.networking.k8s.io/web-ingress created
~/Labs
❯ kubectl describe ing web-ingress
Name:             web-ingress
Labels:           <none>
Namespace:        default
Address:
Ingress Class:    nginx
Default backend:  <default>
Rules:
  Host        Path  Backends
  ----        ----  --------
  lab9.info
              /     web:80 (10.244.120.100:80,10.244.120.101:80,10.244.120.102:80)
              /v2   web2:8080 (10.244.120.105:80,10.244.120.104:80,10.244.120.103:80)
Annotations:  <none>
Events:
  Type    Reason  Age   From                      Message
  ----    ------  ----  ----                      -------
  Normal  Sync    36s   nginx-ingress-controller  Scheduled for sync
```

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: web-ingress
  namespace: default
spec:
  ingressClassName: nginx
  rules:
  - host: lab9.info
    http:
      paths:
      - backend:
          service:
            name: web
            port:
              number: 80
        path: /
        pathType: Prefix
      - backend:
          service:
            name: web2
            port:
              number: 8080
        path: /v2
        pathType: Prefix
```

Działanie dostępu do aplikacji w oparciu o Ingress - cz. IV

5’ Test działania

```
~/Labs
❯ curl -H "Host: lab9.info" http://localhost
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
```

```
~/Labs
❯ curl -H "Host: lab9.info" http://localhost/v2
Hello, world!
Version: 1.0.0
```

5” Test działania z wykorzystanie „symulacji” systemu DNS poprzez wpis do pliku `/etc/hosts` (uwaga, potrzebne są uprawnienia root-a)

```
~/Labs
❯ cat /etc/hosts | grep lab9.info
127.0.0.1 lab9.info
```

```
~/Labs
❯ curl lab9.info
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
```

```
~/Labs
❯ curl lab9.info/v2
Hello, world!
Version: 1.0.0
Hostname: web2-576df8999-hggbm
```

Zdecydowanie zalecana jest metoda z p. 5’. Wykonuje się najpierw sprawdzenie poprawności konfiguracji na „poziomie” klastra, a dopiero potem warto wprowadzać modyfikację w ustawieniach DNS (typowo w konfiguracji wpisów w rekordach serwera DNS)

PFSwChO laboratorium 9

Zadanie - w postaci sprawozdania: **OBOWIĄZKOWE** - cz. I

Załóżmy, że w klastrze mają zostać uruchomione dwie aplikacje (dwa Deployment-y) odpowiednio o nazwach: `app-a` oraz `app-b`, każda o dwóch replikach.

Każda z tych dwóch aplikacji ma działać w swojej przestrzeni nazw, odpowiednio: `appns-a` oraz `appns-b`.

W ramach konfiguracji zasad sieciowych (obiekty NetworkPolicy) należy zabronić **wszelkiej** komunikacji pomiędzy obiektami uruchomionymi w przestrzeni nazw `appns-a` oraz `appns-b`

Dostęp z zewnątrz do aplikacji `app-a` oraz `app-b` ma zostać zrealizowany w oparciu o kontroler Ingress Nginx. Należy utworzyć skojarzony z nim obiekt Ingress, który pozwoli aby odpowiednio:

- aplikacja `app-a` była dostępna pod adresem `a.lab9.net`
- aplikacja `app-b` była dostępna pod adresem `b.lab9.net`

W konfiguracji obiektu Ingress musi zostać uwzględniony domyślny backend (przekierowanie do serwisu domyślnego, gdy nie spełniona jest żadna reguła Ingress) wskazujący na Deployment oparty o obraz `hello-app:1.0` z jedną repliką

Sprawozdanie ma zawierać: pliki konfiguracyjne, ich omówienie oraz potwierdzenia poprawności działania