

ДЗ: Сетевое взаимодействие Pod, сервисы

Выполнение ДЗ II План работы

Данное задание выполняется в `minikube`, версии не ниже 1.0.0. Для MacOS рекомендуется использовать драйвер VM HyperKit (подключение описано [тут](#))

Файлы манифестов для заданий "без звездочки" доступны на GitHub [здесь](#).

Не забудьте про ведение файла `README` и оформление PR.

Работа выполняется в ветке `kubernetes-networks`.

Работа с тестовым веб-приложением

- Добавление проверок Pod
- Создание объекта Deployment
- Добавление сервисов в кластер (ClusterIP)
- Включение режима балансировки IPVS

Доступ к приложению извне кластера

- Установка MetalLB в Layer2-режиме
- Добавление сервиса `LoadBalancer`
- Установка Ingress-контроллера и прокси `ingress-nginx`
- Создание правил `Ingress`

Добавление проверок Pod

- Откройте файл с описанием Pod из предыдущего ДЗ (`kubernetes-intro/web-pod.yml`)
- Добавьте в описание пода `readinessProbe` (можно добавлять его сразу после указания образа контейнера):

```
1  ...
2  spec:
3    containers:
4      - name: web
5        image: thatsme/web:1.2
6        # --- BEGIN ---
7        readinessProbe:           # Добавим проверку готовности
8          httpGet:                # веб-сервера отдавать
9            path: /index.html     # КОНТЕНТ
10           port: 80
11        # --- END ---
12  ...
```

Добавление проверок Pod

- Запустите наш под командой `kubectl apply -f web-pod.yml`

```
$ kubectl apply -f web-pod.yml  
pod/web created
```

- Теперь выполните команду `kubectl get pod/web` и убедитесь, что под перешел в состояние Running

```
$ kubectl get pod/web  
NAME    READY   STATUS    RESTARTS   AGE  
web     1/1     Running   0           5m47s
```

Добавление проверок Pod

- Теперь сделайте команду `kubectl describe pod/web` (вывод объемный, но в нем много интересного)
 - Посмотрите в конце листинга на список `Conditions`:

```
Conditions:
Type          Status
Initialized   True
Ready         False
ContainersReady False
PodScheduled  True
```

- Также посмотрите на список событий, связанных с Pod:

```
Warning Unhealthy 27h (x19 over 28h) kubelet, minikube Readiness probe
failed: Get http://172.17.0.4:80/index.html: dial tcp 172.17.0.4:80:
connect: connection refused
```

Добавление проверок Pod

Из листинга выше видно, что проверка готовности контейнера завершается неудачно. Это неудивительно - веб-сервер в контейнере слушает порт **8000** (по условиям первого ДЗ).

Пока мы не будем исправлять эту ошибку, а добавим другой вид проверок: `livenessProbe`.

- Самостоятельно добавьте в манифест проверку состояния веб-сервера. Например, так:

```
livenessProbe:  
  tcpSocket: { port: 8000 }
```

- Запустите Pod с новой конфигурацией

Добавление проверок Pod

Вопрос для самопроверки:

1. Почему следующая конфигурация валидна, но не имеет смысла?

```
livenessProbe:  
  exec:  
    command:  
      - 'sh'  
      - '-c'  
      - 'ps aux | grep my_web_server_process'
```

2. Бывают ли ситуации, когда она все-таки имеет смысл?

Создание Deployment

Скорее всего, в процессе изменения конфигурации Pod, вы столкнулись с неудобством обновления конфигурации пода через `kubectl` (и уже нашли ключик `--force`).

В любом случае, для управления несколькими однотипными подами такой способ не очень подходит. Создадим **Deployment**, который упростит обновление конфигурации пода и управление группами подов.

- Для начала, создайте новую папку `kubernetes-networks` в вашем репозитории
- В этой папке создайте новый файл `web-deploy.yaml`

Создание Deployment

Начнем заполнять наш файл-манифест для Deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web      # Название нашего объекта Deployment
spec:
  replicas: 1    # Начнем с одного пода
  selector:     # Укажем, какие поды относятся к нашему Deployment:
    matchLabels: #   – это поды с меткой
      app: web   #   app и ее значением web
  template:     # Теперь зададим шаблон конфигурации пода
```

Теперь в блок `template:` можно перенести конфигурацию Pod из `web-pod.yaml`, убрав строки `apiVersion: v1` и `kind: Pod`.

! Будьте внимательны с отступами!

Создание Deployment II Пример

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web
spec:
  replicas: 1
  selector:
    matchLabels:
      app: web
  template:
    ... omitted PodSpec from web-pod.yaml ...
```

Полный пример можно взять в [этом Gist](#)

Создание Deployment

- Для начала удалим старый под из кластера:

```
kubectl delete pod/web --grace-period=0 --force
```

- И приступим к деплою:

```
cd kubernetes-networks/  
kubectl apply -f web-deploy.yaml
```

- Посмотрим, что получилось:

```
kubectl describe deployment web
```



Создание Deployment II Результат

```
1 Name: web
2 Namespace: default
3 CreationTimestamp: Tue, 16 Jul 2019 18:36:42 +0300
4 Labels: <none>
5 Annotations: deployment.kubernetes.io/revision: 1
6             kubectl.kubernetes.io/last-applied-configuration: ...omitted...
7 Selector: app=web
8 Replicas: 1 desired | 1 updated | 1 total | 0 available | 1 unavailable
9 StrategyType: RollingUpdate
10 MinReadySeconds: 0
11 RollingUpdateStrategy: 25% max unavailable, 25% max surge
12 Pod Template: ... omitted ...
13 Conditions:
14   Type          Status Reason
15   ----          -
16   Available     False  MinimumReplicasUnavailable
17   Progressing   True   ReplicaSetUpdated
18 OldReplicaSets: <none>
19 NewReplicaSet:  web-5bc6b958c5 (1/1 replicas created)
20 Events:
21   Type          Reason          Age    From          Message
22   ----          -
23   Normal        ScalingReplicaSet  5m12s deployment-controller Scaled up replica set web-5bc6b958c5 to 1
```

Создание Deployment

- Поскольку мы не исправили `ReadinessProbe`, то поды, входящие в наш **Deployment**, не переходят в состояние *Ready* из-за неуспешной проверки
- На предыдущем слайде видно, что это влияет на состояние всего **Deployment** (строка `Available` в блоке **Conditions**)
- Теперь самое время исправить ошибку! Поменяйте в файле `web-deploy.yaml` следующие параметры:
 - Увеличьте число реплик до 3 (`replicas: 3`)
 - Исправьте порт в `readinessProbe` на порт **8000**
- Примените изменения командой `kubectl apply -f web-deploy.yaml`

Создание Deployment

Самостоятельная работа

- Теперь проверьте состояние нашего **Deployment** командой `kubectl describe deploy/web`
 - Убедитесь, что условия (Conditions) **Available** и **Progressing** выполняются (в столбце *Status* значение **true**)
- Добавьте в манифест (`web-deploy.yaml`) блок `strategy` (можно сразу перед шаблоном пода)

```
strategy:  
  type: RollingUpdate  
  rollingUpdate:  
    maxUnavailable: 0  
    maxSurge: 100%
```

Создание Deployment

Самостоятельная работа

- Попробуйте разные варианты деплоя с крайними значениями `maxSurge` и `maxUnavailable` (оба 0, оба 100%, 0 и 100%)
- За процессом можно понаблюдать с помощью `kubectl get events --watch` или установить [kubespys](#) и использовать его (`kubespys trace deploy`)

```
$ kubespys trace deploy web
[ADDED extensions/v1beta1/Deployment] default/web
Rolling out Deployment revision 2
✘ Deployment is failing; 0 out of 0 Pods are available: [MinimumReplicasUnavailable] Deployment does not have minimum availability.
✘ Rollout has failed; controller is no longer rolling forward: [ProgressDeadlineExceeded] ReplicaSet "web-5f94444bbb" has timed out progressing.

ROLLOUT STATUS:
- [Current rollout | Revision 2] [ADDED] default/web-5f94444bbb
  ⌚ Waiting for ReplicaSet to attain minimum available Pods (0 available of a 1 minimum)
  - [ContainersNotReady] web-5f94444bbb-8nhdg containers with unready status: [web]

- [Previous ReplicaSet | Revision 1] [ADDED] default/web-58464fb5c9
  ⌚ Waiting for ReplicaSet to scale to 0 Pods (3 currently exist)
  - [ContainersNotReady] web-58464fb5c9-8v775 containers with unready status: [web]
  - [ContainersNotReady] web-58464fb5c9-6kv7h containers with unready status: [web]
  - [ContainersNotReady] web-58464fb5c9-txgwp containers with unready status: [web]
```

Создание Service

Для того, чтобы наше приложение было доступно внутри кластера (а тем более - снаружи), нам потребуется объект типа **Service**. Начнем с самого распространенного типа сервисов - **ClusterIP**.

- **ClusterIP** выделяет для каждого сервиса IP-адрес из особого диапазона (этот адрес виртуален и даже не настраивается на сетевых интерфейсах)
- Когда под внутри кластера пытается подключиться к виртуальному IP-адресу сервиса, то нода, где запущен под меняет *адрес получателя* в сетевых пакетах на *настоящий адрес пода*.
- Нигде в сети, за пределами ноды, виртуальный **ClusterIP** не встречается.

Создание Service II ClusterIP

ClusterIP удобны в тех случаях, когда:

- Нам не надо подключаться к *конкретному* поду сервиса
- Нас устраивается случайное распределение подключений между подами
- Нам нужна стабильная точка подключения к сервису, независимая от подов, нод и DNS-имен

Например:

- Подключения клиентов к кластеру БД (multi-read) или хранилищу
- Простейшая (не совсем, *use IPVS, Luke*) балансировка нагрузки внутри кластера

Создание Service II ClusterIP

Итак, создадим манифест для нашего сервиса в папке `kubernetes-networks`.

- Файл `web-svc-cip.yaml`:

```
apiVersion: v1
kind: Service
metadata:
  name: web-svc-cip
spec:
  selector:
    app: web
  type: ClusterIP
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8000
```

- Применим изменения: `kubectl apply -f web-svc-cip.yaml`

Создание Service II ClusterIP

- Проверим результат (отметьте назначенный CLUSTER-IP):

```
1 $ kubectl get services
2 NAME          TYPE          CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
3 kubernetes    ClusterIP     10.96.0.1       <none>           443/TCP          32h
4 web-svc-cip   ClusterIP     10.102.98.208   <none>           80/TCP           17h
```

- Подключимся к VM Minikube (команда `minikube ssh`):
 - Сделайте `curl http://<CLUSTER-IP>/index.html` - работает!
 - Сделайте `ping <CLUSTER-IP>` - пинга нет
 - Сделайте `arp -an, ip addr show` - нигде нет ClusterIP
 - Сделайте `iptables --list -nv -t nat` - вот где наш кластерный IP!

Создание Service II ClusterIP

- Нужное правило находится в цепочке **KUBE-SERVICES**
- Затем мы переходим в цепочку **KUBE-SVC-.....** - здесь находятся правила "балансировки" между цепочками **KUBE-SEP-.....**
 - SVC - очевидно Service
- В цепочках **KUBE-SEP-.....** находятся конкретные правила перенаправления трафика (через DNAT)
 - SEP - Service Endpoint

Подробное описание можно почитать [ТУТ](#) или перейти на IPVS, там чуть понятнее))

Включение IPVS

Итак, с версии 1.0.0 Minikube поддерживает работу `kube-proxy` в режиме IPVS. Попробуем включить его "наживую".

При запуске нового инстанса Minikube лучше использовать ключ `--extra-config` и сразу указать, что мы хотим IPVS

- Включим IPVS для `kube-proxy`, исправив `ConfigMap` (конфигурация Pod, хранящаяся в кластере)
 - Выполните команду `kubectl --namespace kube-system edit configmap/kube-proxy`
 - Или `minikube dashboard` (далее надо выбрать namespace `kube-system`, *Configs and Storage/Config Maps*)

Включение IPVS

- Теперь найдите в файле конфигурации `kube-proxy` строку `mode: ""`

```
1  apiVersion: v1
2  data:
3    config.conf: |-
4      apiVersion: kubeproxy.config.k8s.io/v1alpha1
5      ...
6      iptables:
7        ...
8      ipvs:
9        ...
10     kind: KubeProxyConfiguration
11     metricsBindAddress: 127.0.0.1:10249
12     mode: ""
13     nodePortAddresses: null
14     ...
```

Включение IPVS

- Измените значение `mode` с пустого на `ipvs` и сохраните изменения

```
...  
mode: "ipvs"  
...
```

- Теперь удалим Pod с `kube-proxy`, чтобы применить новую конфигурацию (он входит в `DaemonSet` и будет запущен автоматически)

```
kubectl --namespace kube-system delete pod --selector='k8s-app=kube-proxy'
```

Описание работы и настройки [IPVS в K8s](#)

Включение IPVS

- После успешного рестарта `kube-proxy` выполним команду `minikube ssh` и проверим, что получилось
- Выполним команду `iptables --list -nv -t nat` в VM Minikube
- Что-то поменялось, но старые цепочки на месте (хотя у них теперь 0 references) 😞
 - `kube-proxy` настроил все по-новому, но не удалил мусор
 - Запуск `kube-proxy --cleanup` в нужном поде - тоже не помогает

```
kubectl --namespace kube-system exec kube-proxy-⟨POD⟩ kube-proxy --cleanup
```

Включение IPVS

Полностью очистим все правила `iptables`:

- Создадим в VM с Minikube файл `/tmp/iptables.cleanup`

```
*nat
-A POSTROUTING -s 172.17.0.0/16 ! -o docker0 -j MASQUERADE
COMMIT

*filter
COMMIT

*mangle
COMMIT
```

- Применим конфигурацию: `iptables-restore /tmp/iptables.cleanup`
- Проверим результат `iptables --list -nv -t nat`

Включение IPVS

- Итак, лишние правила удалены и мы видим только актуальную конфигурацию
 - `kube-proxy` периодически делает полную синхронизацию правил в своих цепочках)
- Как посмотреть конфигурацию IPVS? Ведь в VM нет утилиты `ipvsadm`?
 - В VM выполним команду `toolbox` - в результате мы окажется в контейнере с Fedora
 - Теперь установим `ipvsadm`:

```
dnf install -y ipvsadm && dnf clean all
```

Включение IPVS

- Выполним `ipvsadm --list -n` и среди прочих сервисов найдем наш:

```
$ ipvsadm --list -n
TCP 10.106.18.171:80 rr
  -> 172.17.0.11:8000          Masq    1      0      0
  -> 172.17.0.12:8000          Masq    1      0      0
  -> 172.17.0.13:8000          Masq    1      0      0
```

- Теперь выйдем из контейнера `toolbox` и сделаем `ping` кластерного IP:

```
$ ping -c1 10.106.18.171
PING 10.106.18.171 (10.106.18.171): 56 data bytes
64 bytes from 10.106.18.171: seq=0 ttl=64 time=0.277 ms
--- 10.106.18.171 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.227/0.227/0.277 ms
```

Включение IPVS

Итак, все работает. Но почему пингуется виртуальный IP?

Все просто - он уже не такой виртуальный. Этот IP теперь есть на интерфейсе `kube-ipvs0`:

```
$ ip addr show kube-ipvs0
57: kube-ipvs0: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN group default
    link/ether 16:0b:8d:8c:fe:ac brd ff:ff:ff:ff:ff:ff
    ...
    inet 10.99.228.101/32 brd 10.99.228.101 scope global kube-ipvs0
        valid_lft forever preferred_lft forever
```

Также, правила в `iptables` построены по-другому. Вместо цепочки правил для каждого сервиса, теперь используются хэш-таблицы (`ipset`). Можете посмотреть их, установив утилиту `ipset` в `toolbox`.

Работа с LoadBalancer и Ingress

Установка MetalLB

MetalLB позволяет запустить внутри кластера L4-балансировщик, который будет принимать извне запросы к сервисам и раскидывать их между подами. Установка его проста:

- Установите манифест:

```
kubectl apply -f  
https://raw.githubusercontent.com/google/metallb/v0.8.0/manifests/metallb.yaml
```

! В продуктиве так делать не надо. Сначала стоит скачать файл и разобраться, что там внутри

Установка MetalLB

Проверьте, что были созданы нужные объекты:

```
$ kubectl --namespace metallb-system get all
```

NAME	READY	STATUS	RESTARTS	AGE
pod/controller-547d466688-27z1b	1/1	Running	0	34h
pod/speaker-bc84g	1/1	Running	0	34h

NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE	NODE SELECTOR	AGE
daemonset.apps/speaker	1	1	1	1	1	<none>	34h

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/controller	1/1	1	1	34h

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/controller-547d466688	1	1	1	34h

Установка MetalLB

Теперь настроим балансировщик с помощью `ConfigMap`

- Создайте манифест `metallb-config.yaml` в папке `kubernetes-networks`:

```
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - "172.17.255.1-172.17.255.254"
```

Установка MetalLB

- В конфигурации мы настраиваем:
 - Режим L2 (анонс адресов балансировщиков с помощью ARP)
 - Создаем пул адресов 172.17.255.1-172.17.255.254 - они будут назначаться сервисам с типом `LoadBalancer`
- Теперь можно применить наш манифест: `kubectl apply -f metallb-config.yaml`
- Контроллер подхватит изменения автоматически

MetaLB II Проверка конфигурации

- Сделайте копию файла `web-svc-cip.yaml` в `web-svc-lb.yaml` и откройте его в редакторе.
- Измените имя сервиса и его тип на `LoadBalancer`

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: web-svc-lb
5  spec:
6    selector:
7      app: web
8    type: LoadBalancer
9    ports:
10     - protocol: TCP
11       port: 80
12       targetPort: 8000
```

- Примените манифест

MetalLB II Проверка конфигурации

- Теперь посмотрите логи пода-контроллера MetalLB (подставьте правильное имя!)

```
1 $ kubectl --namespace metallb-system logs pod/controller-XXXXXXXX-XXXXXX
2
3 ... omitted ...
4 {"caller":"main.go:49","event":"startUpdate","msg":"start of service
  update","service":"default/web-svc-lb","ts":"2019-07-19T00:38:52.200615669Z"}
5 {"caller":"service.go:88","event":"ipAllocated","ip":"172.17.255.2","msg":"IP
  address assigned by controller","service":"default/web-svc-lb","ts":"2019-07-
  19T00:38:52.201979269Z"}
6 {"caller":"main.go:93","event":"serviceUpdated","msg":"updated service
  object","service":"default/web-svc-lb","ts":"2019-07-19T00:38:52.220374969Z"}
7 {"caller":"main.go:95","event":"endUpdate","msg":"end of service
  update","service":"default/web-svc-lb","ts":"2019-07-19T00:38:52.220477069Z"}
```

- Обратите внимание на назначенный IP-адрес (или посмотрите его в выводе `kubectl describe svc web-svc-lb`)

MetalLB II Проверка конфигурации

- Если мы попробуем открыть URL `http://<our_LB_address>/index.html`, то... ничего не выйдет.
- Это потому, что сеть кластера изолирована от нашей основной ОС (а ОС не знает ничего о подсети для балансировщиков)
- Чтобы это поправить, добавим статический маршрут
 - В реальном окружении это решается добавлением нужной подсети на интерфейс сетевого оборудования
 - Или использованием L3-режима (что потребует усилий от сетевиков, но более предпочтительно)

MetaLB II Проверка конфигурации

- Найдите IP-адрес виртуалки с Minikube. Например так:

```
1  $ minikube ssh
2      Welcome to Minikube VM!
3  $ ip addr show eth0
4
5  2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state
   UP group default qlen 1000
6  link/ether 26:a8:d8:89:c2:39 brd ff:ff:ff:ff:ff:ff
7  inet 192.168.64.4/24 brd 192.168.64.255 scope global dynamic eth0
8      valid_lft 54714sec preferred_lft 54714sec
9  inet6 fe80::24a8:d8ff:fe89:c239/64 scope link
10     valid_lft forever preferred_lft forever
```

- Добавьте маршрут в вашей ОС на IP-адрес Minikube:

```
$ sudo route add 172.17.255.0/24 192.168.64.4
Password:
add net 172.17.255.0: gateway 192.168.64.4
```

MetalLB II Проверка конфигурации

DISCLAIMER:

Добавление маршрута может иметь другой синтаксис

(например, `ip route add 172.17.255.0/24 via 192.168.64.4`
в ОС Linux)

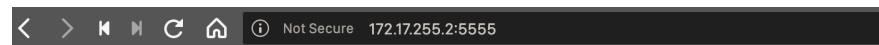
или вообще не сработать (в зависимости от VM Driver в Minikube).

В этом случае, не надо расстраиваться - работу наших сервисов и манифестов можно проверить из консоли Minikube, просто будет не так эффективно.

P.S. - Самый простой способ найти IP виртуалки с minikube -
`minikube ip`

MetalLB II Проверка конфигурации

Если все получилось, то можно открыть в браузере URL с IP-адресом нашего балансировщика и посмотреть, как космические корабли бороздят просторы вселенной.



MetalLB II Проверка конфигурации

Если пообновлять страничку с помощью **Ctrl-F5** (т.е. игнорируя кэш), то будет видно, что каждый наш запрос приходит на другой под. Причем, порядок смены подов - всегда один и тот же.

Так работает IPVS - по умолчанию он использует **rr** (Round-Robin) балансировку.

К сожалению, выбрать алгоритм на уровне манифеста сервиса нельзя. Но когда-нибудь, эта полезная фича [появится](#)

Доступные алгоритмы балансировки описаны [здесь](#) и [здесь](#)

MetalLB II Задание со

Сервисы **LoadBalancer** работают в том числе и с UDP-трафиком.

Задание:

- Сделайте сервис **LoadBalancer**, который откроет доступ к CoreDNS снаружи кластера
- Поскольку DNS работает по TCP и UDP протоколам - учтите это в конфигурации. Оба протокола должны работать по одному и тому же IP-адресу балансировщика.

 Hint

Создание Ingress

Теперь, когда у нас есть балансировщик, можно заняться Ingress-контроллером и прокси:

- неудобно, когда на каждый Web-сервис надо выделять свой IP-адрес
- а еще хочется балансировку по HTTP-заголовкам (sticky sessions)

Для нашего домашнего задания возьмем почти "коробочный" `ingress-nginx` от проекта Kubernetes. Это "достаточно хороший" Ingress для умеренных нагрузок, основанный на OpenResty и пачке Lua-скриптов.

Создание Ingress

- Установка начинается с основного манифеста:

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/static/mandatory.yaml
```

- После установки основных компонентов, в [ИНСТРУКЦИИ](#) рекомендуется применить манифест, который создаст **NodePort**-сервис. Но у нас есть MetalLB, мы можем сделать круче.

Можно сделать просто `minikube addons enable ingress`, но мы не ищем легких путей

Создание Ingress

Создадим файл `nginx-lb.yaml` с конфигурацией `LoadBalancer`-сервиса (работаем в каталоге `kubernetes-networks`):

```
kind: Service
apiVersion: v1
metadata:
  name: ingress-nginx
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
spec:
  externalTrafficPolicy: Local
  type: LoadBalancer
  selector:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
  ports:
    - { name: http, port: 80, targetPort: http }
    - { name: https, port: 443, targetPort: https }
```

Создание Ingress

- Теперь применим созданный манифест и посмотрим на IP-адрес, назначенный ему MetalLB
- Теперь можно сделать пинг на этот IP-адрес и даже `curl`

```
1 <html>
2 <head><title>404 Not Found</title></head>
3 <body>
4 <center><h1>404 Not Found</h1></center>
5 <hr><center>openresty/1.15.8.1</center>
6 </body>
7 </html>
```

Если видим страничку 404 от OpenResty - значит работает! 🙌

Подключение приложение Web к Ingress

- Наш Ingress-контроллер не требует `ClusterIP` для балансировки трафика
- Список узлов для балансировки заполняется из ресурса `Endpoints` нужного сервиса (это нужно для "интеллектуальной" балансировки, привязки сессий и т.п.)
- Поэтому мы можем использовать headless-сервис для нашего веб-приложения.
- Скопируйте `web-svc-cip.yaml` в `web-svc-headless.yaml`
 - измените имя сервиса на `web-svc`
 - добавьте параметр `clusterIP: None`

Создание Headless-сервиса

```
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: web-svc
5 spec:
6   selector:
7     app: web
8   type: ClusterIP
9   clusterIP: None
10  ports:
11    - protocol: TCP
12      port: 80
13      targetPort: 8000
```

- Теперь примените полученный манифест и проверьте, что **ClusterIP** для сервиса **web-svc** действительно не назначен

Создание правил Ingress

Теперь настроим наш ingress-прокси, создав манифест с ресурсом **Ingress** (файл назовите `web-ingress.yaml`):

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: web
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
    paths:
    - path: /web
      backend:
        serviceName: web-svc
        servicePort: 8000
```

Создание правил Ingress

Примените манифест и проверьте, что корректно заполнены **Address** и **Backends**

```
1 $ kubectl describe ingress/web
2
3 Name:                web
4 Namespace:           default
5 Address:              172.17.255.1
6 Default backend:     default-http-backend:80 (<none>)
7 Rules:
8   Host  Path  Backends
9   ----  ---  -
10  *
11      /web  web-svc:8000 (172.17.0.11:8000,172.17.0.12:8000,172.17.0.3:8000)
12 Annotations:
13   kubectl.kubernetes.io/last-applied-configuration: ... omitted ...
14   nginx.ingress.kubernetes.io/rewrite-target: /
15 Events:                <none>
```

Создание правил Ingress

- Теперь можно проверить, что страничка доступна в браузере (http://<LB_IP>/web/index.html)
- Обратите внимание, что обращения к странице тоже балансируются между Podами. Только сейчас это происходит средствами nginx, а не IPVS

Задания со ★

- Простое задание со ★: добавьте доступ к `kubernetes-dashboard` через наш Ingress-прокси (сервис должен быть доступен через префикс `/dashboard`)
- Реализуйте канареечное развертывание (перенаправление части трафика на выделенную группу подов по HTTP-заголовку). Документация [IUI](#). Естественно, что вам понадобятся 1-2 "канареечных" пода.

Проверка ДЗ

- Результаты вашей работы должны быть добавлены в ветку **kubernetes-networks** вашего GitHub репозитория `<YOUR_LOGIN>_platform`
- В **README.md** рекомендуется внести описание того, что сделано
- Создайте Pull Request к ветке **master** (описание PR рекомендуется заполнять)
- Добавьте метку **homework-3** к вашему PR
- В Assignees к PR добавьте преподавателя, который вел лекцию (GitHub логин **virtualshuric**)

Проверка ДЗ

- После того как автоматизированные тесты проверят корректность выполнения ДЗ, необходимо сделать merge ветки **kubernetes-networks** в **master** и закрыть PR
- Если у вас возникли вопросы по ДЗ и необходима консультация преподавателей - после прохождения автотестов добавьте к PR метку **Review Required** и **не мерджите PR** самостоятельно

Проверка ДЗ

Структура репозитория после выполнения домашнего задания:

```
|— .github
|   └─ PULL_REQUEST_TEMPLATE.md
|— .travis.yml
|— LICENSE
|— README.md
└─ kubernetes-intro
    └─ web
        └─ Dockerfile
        └─ web-pod.yaml
└─ kubernetes-networks
    └─ metallb-config.yaml
    └─ nginx-lb.yaml
    └─ web-deploy.yaml
    └─ web-ingress.yaml
    └─ web-svc-cip.yaml
    └─ web-svc-headless.yaml
    └─ web-svc-lb.yaml
```