Homework: Kubernetes

- Домашнее задание принимается как Pull Request в основной репозиторий
- Имя ветки hw-kubernetes-01
- Наличие всех созданных файлов в каталоге k8s
- Описание выполненных действий в README.MD корня репозитория.
- Короткое описание в Description Pull Request-а.
- Reviewer: sergeykudelin

Установка кластера на локальной рабочем месте

- Установим утилиту minikube https://minikube.sigs.k8s.io/docs/
- В зависимости какая у Вас ОС можете выбрать решение по виртуализации https://minikube.sigs.k8s.io/docs/drivers/
  - Hyper-Kit (MacOS)
  - VirtualBox (Windows/MacOS/Linux)
  - Hyper-V (Windows)
- Установим самый свежий kubectl https://kubernetes.io/docs/tasks/tools/
- Сконфигурируем первый кластер с помощью утилиты minikube
  - Создаем first-cluster

```
➜  sergeykudelin-hw: minikube start -p first-cluster

😄  [first-cluster] minikube v1.22.0 on Darwin 11.4
❗  Both driver=hyperkit and vm-driver=hyperkit have been
set.

    Since vm-driver is deprecated, minikube will default to
driver=hyperkit.

    If vm-driver is set in the global config, please run
"minikube config unset vm-driver" to resolve this warning.

✨  Using the hyperkit driver based on user configuration
👍  Starting control plane node first-cluster in cluster
first-cluster
🔥  Creating hyperkit VM (CPUs=2, Memory=4000MB,
Disk=20000MB) ...
🐳  Preparing Kubernetes v1.21.2 on Docker 20.10.6 ...
    ▪ Generating certificates and keys ...
    ▪ Booting up control plane ...
    ▪ Configuring RBAC rules ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-
provisioner:v5
🌟  Enabled addons: storage-provisioner, default-
storageclass
🏄  Done! kubectl is now configured to use "first-cluster"
cluster and "default" namespace by default
```

  - Создаем второй по аналогии с именем second-cluster

```
➜  sergeykudelin-hw: minikube start -p first-cluster
```

- Проверим доступность обоих кластеров при использовании утилиты kubectl

```
➜  sergeykudelin-hw git:(k8s_intro) ✗ kubectl config get-
contexts
  CURRENT   NAME                  CLUSTER
AUTHINFO
NAMESPACE
            first-cluster         first-cluster
first-cluster
default
  *         second-cluster        second-cluster
second-cluster
default
```

- Первый делом переключимся на первый кластере

```
➜  sergeykudelin-hw git:(k8s_intro) ✗ kubectl config use-
context first-cluster
Switched to context "first-cluster".
```

- Как мы видим minikube сделал нам кластер из одной master node-ы, так еще и добавил нужные контексты в настройки kubectl
- Ознакомьтесь с содержимым конфиг файла kubectl что бы составить понимание из каких аттребутов состоит каждый профиль подключение

```
cat ~/.kube/config
```

- Ознакомимся со существующими namespaces-ами в кластере

```
➜  sergeykudelin-hw git:(k8s_intro) ✗ kubectl get ns
NAME              STATUS   AGE
default           Active   11m
kube-node-lease   Active   11m
kube-public       Active   11m
kube-system       Active   11m
```

- Так же можем посмотреть из каких nodes состоит кластер и увидем что из одной, которая так же выполняет функционал master node.

```
➜  sergeykudelin-hw git:(k8s_intro) ✗ kubectl get nodes
NAME            STATUS   ROLES                  AGE
VERSION
first-cluster   Ready    control-plane,master   13m
v1.21.2
```

- Так же ознакомимся с CLusterRole по умолчанию, которые можно использовать что бы не повторять функционал для предоставление прав новым пользователям

```

```

```
➜  sergeykudelin-hw git:(k8s_intro) ✗ kubectl get
clusterrole
NAME
CREATED AT
admin
2021-08-02T19:08:13Z
cluster-admin
2021-08-02T19:08:13Z
...
```

- Попробуем посмотреть просто Role, если они существуют

```
➜  sergeykudelin-hw git:(k8s_intro) ✗ kubectl get role
No resources found in default namespace.
```

- Попробуйте прокомментировать, почему здесь ни чего.
- Давайте проверить системный namespace kube-system

```
➜  sergeykudelin-hw git:(k8s_intro) ✗ kubectl -n kube-
system get role
NAME                                             CREATED AT
extension-apiserver-authentication-reader        2021-08-
02T19:08:13Z
kube-proxy                                       2021-08-
02T19:08:15Z
kubeadm:kubelet-config-1.21                      2021-08-
02T19:08:14Z
kubeadm:nodes-kubeadm-config                     2021-08-
02T19:08:14Z
system::leader-locking-kube-controller-manager   2021-08-
02T19:08:13Z
system::leader-locking-kube-scheduler            2021-08-
02T19:08:13Z
system:controller:bootstrap-signer               2021-08-
02T19:08:13Z
system:controller:cloud-provider                 2021-08-
02T19:08:13Z
system:controller:token-cleaner                  2021-08-
02T19:08:13Z
system:persistent-volume-provisioner             2021-08-
02T19:08:17Z
```

- Как видете существуют Роли действия которых распростроняются только на namespace kube-system
- Проверим наличие NetworkPolicy, они отсутствуют так как kubernetes предоставляет по умолчанию полным доступ между всеми обьектами

```
➜  sergeykudelin-hw git:(k8s_intro) ✗ kubectl get
NetworkPolicy
No resources found in default namespace.
➜  sergeykudelin-hw git:(k8s_intro) ✗ kubectl -n kube-
system get NetworkPolicy
No resources found in kube-system namespace.
```

- Удалим второй кластер в связи не актуальностью для дальнейших действий.

```
➜  sergeykudelin-hw git:(k8s_intro) ✗ minikube delete -p
second-cluster
🔥  Deleting "second-cluster" in hyperkit ...
💀  Removed all traces of the "second-cluster" cluster.
```

- Для ознакомление с сервисами на базе которых построен kubernetes зайдем в виртуальную машину minikube через следующую команду.

```
➜  sergeykudelin-hw git:(k8s_intro) ✗ minikube ssh -p
first-cluster
                         _                 _
                      _ ( )               ( )
___ ___   (_)   ___   (_)| |/')   _    _ | |_       __
/' _ ` _ `\| |/' _ `\| || , <   ( ) ( )| '_`\   /'__`\
| ( ) ( ) || || ( ) || || |\`\ | (_) || |_) )(  ___/
(_) (_) (_)(_)(_) (_)(_)(_) (_)`\___/'(_,__/'`\____)

$
```

- Попробуем найти все сервисы

```
$ ps -ax | grep kube
3350 ?        Ssl    1:52 kube-apiserver --advertise-
address=192.168.64.18 --allow-privileged=true --
authorization-mode=Node,RBAC --client-ca-
file=/var/lib/minikube/certs/ca.crt --enable-admission-
plugins=NamespaceLifecycle,LimitRanger,ServiceAccount,Defau
ltStorageClass,DefaultTolerationSeconds,NodeRestriction,Mut
atingAdmissionWebhook,ValidatingAdmissionWebhook,ResourceQu
ota --enable-bootstrap-token-auth=true --etcd-
cafile=/var/lib/minikube/certs/etcd/ca.crt --etcd-
certfile=/var/lib/minikube/certs/apiserver-etcd-client.crt
--etcd-keyfile=/var/lib/minikube/certs/apiserver-etcd-
client.key --etcd-servers=https://127.0.0.1:2379 --
insecure-port=0 --kubelet-client-
certificate=/var/lib/minikube/certs/apiserver-kubelet-
client.crt --kubelet-client-
key=/var/lib/minikube/certs/apiserver-kubelet-client.key --
kubelet-preferred-address-
types=InternalIP,ExternalIP,Hostname --proxy-client-cert-
file=/var/lib/minikube/certs/front-proxy-client.crt --
proxy-client-key-file=/var/lib/minikube/certs/front-proxy-
client.key --requestheader-allowed-names=front-proxy-client
--requestheader-client-ca-
file=/var/lib/minikube/certs/front-proxy-ca.crt --
requestheader-extra-headers-prefix=X-Remote-Extra- --
requestheader-group-headers=X-Remote-Group --requestheader-
username-headers=X-Remote-User --secure-port=8443 --
service-account-
issuer=https://kubernetes.default.svc.cluster.local --
service-account-key-file=/var/lib/minikube/certs/sa.pub --
service-account-signing-key-
file=/var/lib/minikube/certs/sa.key --service-cluster-ip-
range=10.96.0.0/12 --tls-cert-
```

```
file=/var/lib/minikube/certs/apiserver.crt --tls-private-
key-file=/var/lib/minikube/certs/apiserver.key

3421 ?        Ssl    0:26 etcd --advertise-client-
urls=https://192.168.64.18:2379 --cert-
file=/var/lib/minikube/certs/etcd/server.crt --client-cert-
auth=true --data-dir=/var/lib/minikube/etcd --initial-
advertise-peer-urls=https://192.168.64.18:2380 --initial-
cluster=first-cluster=https://192.168.64.18:2380 --key-
file=/var/lib/minikube/certs/etcd/server.key --listen-
client-
urls=https://127.0.0.1:2379,https://192.168.64.18:2379 --
listen-metrics-urls=http://127.0.0.1:2381 --listen-peer-
urls=https://192.168.64.18:2380 --name=first-cluster --
peer-cert-file=/var/lib/minikube/certs/etcd/peer.crt --
peer-client-cert-auth=true --peer-key-
file=/var/lib/minikube/certs/etcd/peer.key --peer-trusted-
ca-file=/var/lib/minikube/certs/etcd/ca.crt --proxy-
refresh-interval=70000 --snapshot-count=10000 --trusted-ca-
file=/var/lib/minikube/certs/etcd/ca.crt

3515 ?        Ssl    0:04 kube-scheduler --authentication-
kubeconfig=/etc/kubernetes/scheduler.conf --authorization-
kubeconfig=/etc/kubernetes/scheduler.conf --bind-
address=127.0.0.1 --
kubeconfig=/etc/kubernetes/scheduler.conf --leader-
elect=false --port=0

3569 ?        Ssl    0:38 kube-controller-manager --
allocate-node-cidrs=true --authentication-
kubeconfig=/etc/kubernetes/controller-manager.conf --
authorization-kubeconfig=/etc/kubernetes/controller-
manager.conf --bind-address=127.0.0.1 --client-ca-
file=/var/lib/minikube/certs/ca.crt --cluster-
cidr=10.244.0.0/16 --cluster-name=mk --cluster-signing-
cert-file=/var/lib/minikube/certs/ca.crt --cluster-signing-
key-file=/var/lib/minikube/certs/ca.key --
controllers=*,bootstrapsigner,tokencleaner --
kubeconfig=/etc/kubernetes/controller-manager.conf --
leader-elect=false --port=0 --requestheader-client-ca-
file=/var/lib/minikube/certs/front-proxy-ca.crt --root-ca-
file=/var/lib/minikube/certs/ca.crt --service-account-
private-key-file=/var/lib/minikube/certs/sa.key --service-
cluster-ip-range=10.96.0.0/12 --use-service-account-
credentials=true

3846 ?        Ssl    0:51
/var/lib/minikube/binaries/v1.21.2/kubelet --bootstrap-
kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --
config=/var/lib/kubelet/config.yaml --container-
runtime=docker --hostname-override=first-cluster --
kubeconfig=/etc/kubernetes/kubelet.conf --node-
ip=192.168.64.18

4346 ?        Ssl    0:00 /usr/local/bin/kube-proxy --
config=/var/lib/kube-proxy/config.conf --hostname-
override=first-cluster
```

```
12646 pts/0    S+     0:00 grep kube
$
```

- Примерно так запускают сервисы на базе которых построен kubernetes)))
- Выходим из виртуального сервер

Начнем подготовку нашего приложения RealWorld для deployment в кластер

- Создадим дополнительный каталог k8s
- Ранее когда мы создавали docker образ frontend части, мы столкнули с проблемой что значение переменной **REACT_APP_BACKEND_URL** необходимо было определять при сборке приложение. Что не есть быть хорошо, так как нам надо иметь возможность определять данное значение при deployment-е а не build. Воспользуем менее красивым вариантом, но не требующий вмешивать в код приложения, будем собирать **docker image с nodejs + приложение с исходниками**, что позволит запускать приложение и слушать переменную REACT_APP_BACKEND_URL из переменного окружения.
- Переименует файл Dockerfile в директории /frontend в Dockerfile.previous
- Создадим новый DockerFile со следующим содержимым

```
FROM node:12
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

COPY package.json /usr/src/app/
RUN npm install && npm cache clean -f
COPY . /usr/src/app

CMD [ "npm", "start" ]
```

- Соберите образ и опубликуйте его в Вашем публичном репозитории DockerHub с новым tag-ом.

Пишем манифесты которые позволят нам обьяснить kubernetes как запускать наше приложение

Для начала нам надо запустить сами приложения. Из лекции мы знаем что лучше всего для этого подойдет сущность deployment

- Создаем файл манифеста для нашего frontend компоненты, только замените на значения image на Ваше, опубликованное в DockerHub

```
➜  k8s git:(k8s_intro) ✗ cat deployment_frontend.yml
---
apiVersion: apps/v1
kind: Deployment
metadata:
name: fe-realworld
labels:
    app: realworld
    type: frontend
```

```
spec:
replicas: 2
selector:
    matchLabels:
    app: realworld
    type: frontend
template:
    metadata:
    labels:
        app: realworld
        type: frontend
    spec:
    containers:
    - name: fe-realworld
        image: sergeykudelin/hillel-frontend:0.0.7
        env:
        - name: REACT_APP_BACKEND_URL
        value: "http://backend.realworld.local.io/api"
        ports:
        - containerPort: 4100
        imagePullPolicy: Always
```

- Создаем файл манифеста для нашего backend компоненты

```
➜  k8s git:(k8s_intro) ✗ cat deployment_backend.yml
---
apiVersion: apps/v1
kind: Deployment
metadata:
name: be-realworld
labels:
    app: realworld
    type: backend
spec:
replicas: 2
selector:
    matchLabels:
    app: realworld
    type: backend
template:
    metadata:
    labels:
        app: realworld
        type: backend
    spec:
    containers:
    - name: be-realworld
        image: sergeykudelin/hillel-backend:0.0.1
        ports:
        - containerPort: 8081
        imagePullPolicy: Always
        env:
        - name: PORT
        value: "8081"
        - name: NODE_ENV
        value: "production"
        - name: MONGO_DB_URI
```

```
        value: "mongodb://mongo/conduit"
        - name: SECRET
        value: "secret"%
```

- Остается база данных, для нее выберим отдельный тип сущности который идеально подходит для StateFul приложений.
    - Но есть ПРОБЛЕМА (задача) для нее необходимо предварительно Persistent Volume который будет выступать хранилищем каталога /data/db
    - Но есть и другая ПРОБЛЕМА (задача) нам надо какой Peristent Volume который автоматом бы создавался и управлялся kubernetes. Воспользуемся расширение Kubernetes и добавим в наш кластер CSI (Container Storage Interface) который на базе дисковорой подсистемы нашего хоста, будет сам выделять место и создавать Persistent Volume

```
➜  k8s git:(k8s_intro) ✗ minikube addons enable csi-
hostpath-driver -p first-cluster
❗  [WARNING] For full functionality, the 'csi-hostpath-
driver' addon requires the 'volumesnapshots' addon to be
enabled.

You can enable 'volumesnapshots' addon by running:
'minikube addons enable volumesnapshots'

    ▪ Using image k8s.gcr.io/sig-storage/csi-resizer:v1.1.0
    ▪ Using image k8s.gcr.io/sig-
storage/hostpathplugin:v1.6.0
    ▪ Using image k8s.gcr.io/sig-storage/csi-
snapshotter:v4.0.0
    ▪ Using image k8s.gcr.io/sig-storage/csi-
attacher:v3.1.0
    ▪ Using image k8s.gcr.io/sig-storage/csi-external-
health-monitor-agent:v0.2.0
    ▪ Using image k8s.gcr.io/sig-storage/csi-external-
health-monitor-controller:v0.2.0
    ▪ Using image k8s.gcr.io/sig-storage/csi-node-driver-
registrar:v2.0.1
    ▪ Using image k8s.gcr.io/sig-
storage/livenessprobe:v2.2.0
    ▪ Using image k8s.gcr.io/sig-storage/csi-
provisioner:v2.1.0
🔎  Verifying csi-hostpath-driver addon...
🌟  The 'csi-hostpath-driver' addon is enabled
```

    - Теперь у нас есть возможность написать файл манифеста для создание Persistent Volume с использование Persistent Volume Claim и Storage Class освобождая нас ок дополнительных действий по созданию и подключению.

```
➜  k8s git:(k8s_intro) ✗ cat pvc_mongo.yml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
name: mongo-pvc
spec:
accessModes:
```

```
- ReadWriteOnce
resources:
    requests:
    storage: 1Gi
storageClassName: csi-hostpath-sc
```

- Следующий этап напишем файл манифеста для mongodb

```
➜  k8s git:(k8s_intro) ✗ cat statefulset_mongo.yml
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
name: mongo-realworld
labels:
    app: realworld
    type: mongo
spec:
serviceName: mongo
replicas: 1
selector:
    matchLabels:
    app: realworld
    type: mongo
template:
    metadata:
    labels:
        app: realworld
        type: mongo
    spec:
    containers:
    - name: mongo-realworld
        image: mongo:latest
        ports:
        - containerPort: 27017
        imagePullPolicy: Always
        volumeMounts:
        - name: mongo-pvc
        mountPath: /data/db
volumeClaimTemplates:
- metadata:
    name: mongo-pvc
    spec:
    accessModes:
    - ReadWriteOnce
    resources:
        requests:
        storage: 1Gi
    storageClassName: csi-hostpath-sc
```

Попробуем запустить наши контейнера, ДОЛЖНЫ хотя бы не падать))

- Запускаем mongodb
    - Создаем PVC

```
➜  k8s git:(k8s_intro) ✗ kubectl apply -f pvc_mongo.yml
persistentvolumeclaim/mongo-pvc created
```

- Проверяем корректность создания

```
➜  k8s git:(k8s_intro) ✗kubectl get pvc
NAME        STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS      AGE
mongo-pvc   Bound    pvc-931fdaff-4d91-4893-9c4b-
e72510af30f4   1Gi        RWO            csi-hostpath-sc
31s

➜  k8s git:(k8s_intro) ✗ kubectl get pv
NAME                                       CAPACITY
ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM
STORAGECLASS      REASON   AGE
pvc-931fdaff-4d91-4893-9c4b-e72510af30f4   1Gi        RWO
Delete           Bound    default/mongo-pvc   csi-hostpath-
sc           44s
```

- Создаем StatefulSet с mongodb

```
➜  k8s git:(k8s_intro) ✗ kubectl apply -f
statefulset_mongo.yml
statefulset.apps/mongo-realworld created
➜  k8s git:(k8s_intro) ✗ kubectl get statefulset
NAME             READY   AGE
mongo-realworld   0/1     13s
```

- Проверяем через время, пока READY не будет 1/1
- Состояние controller-а можно посмотреть через describe

```
➜  k8s git:(k8s_intro) ✗  kubectl describe statefulset
mongo-realworld
Name:               mongo-realworld
Namespace:          default
CreationTimestamp:  Mon, 02 Aug 2021 23:08:18 +0300
Selector:           app=realworld,type=mongo
Labels:             app=realworld
                    type=mongo
Annotations:        <none>
Replicas:           1 desired | 1 total
Update Strategy:    RollingUpdate
Partition:        0
Pods Status:        0 Running / 1 Waiting / 0 Succeeded / 0
Failed
Pod Template:
Labels:  app=realworld
        type=mongo
Containers:
mongo-realworld:
    Image:        mongo:latest
    Port:         27017/TCP
    Host Port:    0/TCP
    Environment:  <none>
```

```
    Mounts:
    /data/db from mongo-pvc (rw)
Volumes:  <none>
Volume Claims:
Name:          mongo-pvc
StorageClass:  csi-hostpath-sc
Labels:        <none>
Annotations:   <none>
Capacity:      1Gi
Access Modes:  [ReadWriteOnce]
Events:
Type    Reason            Age   From
Message
----    ------            ----  ----                    ---
----
Normal  SuccessfulCreate  105s  statefulset-controller
create Claim mongo-pvc-mongo-realworld-0 Pod mongo-
realworld-0 in StatefulSet mongo-realworld success
Normal  SuccessfulCreate  105s  statefulset-controller
create Pod mongo-realworld-0 in StatefulSet mongo-realworld
successful
```

- Проверяем повторно наш statefulset

```
➜  k8s git:(k8s_intro) ✗ kubectl get statefulset
NAME              READY   AGE
mongo-realworld   1/1     3m7s
```

- Так как наша БД готова, можем поднимать backend, но НЕТ!!! Как же наш backend найдет mongodb в нашем кластере. Ответ необходимо создать **service** которые помогут опубликовать наши приложения в рамках Kubernetes
  - Создаем файл манифеста для mongo

```
➜  k8s git:(k8s_intro) ✗ cat svc_mongo.yml
apiVersion: v1
kind: Service
metadata:
name: mongo
labels:
    app: realworld
    type: mongo
spec:
ports:
- port: 27017
    name: mongo
clusterIP: None
selector:
    app: realworld
    type: mongo
```

  - Применяем манифест

```
➜  k8s git:(k8s_intro) ✗ kubectl apply -f svc_mongo.yml
service/mongo created
```

- Проверяем создание и доступность в рамках kubernetes network

```
➜  k8s git:(k8s_intro) ✗ kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP
PORT(S)     AGE
kubernetes   ClusterIP   10.96.0.1    <none>
443/TCP     67m
mongo        ClusterIP   None         <none>
27017/TCP   46s
```

- Как видим для mongodb был создам service без выделения дополнительного ClusterIP.

- Пробуем применять deployment для нашего backend приложения

```
➜  k8s git:(k8s_intro) ✗ kubectl apply -f
deployment_backend.yml
deployment.apps/be-realworld created
```

- Проверяем его наличие, вроде есть и даже не падают))

```
➜  k8s git:(k8s_intro) ✗ kubectl get deployment
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
be-realworld   2/2     2            2           29s
```

- Пробуем применять deployment для нашего frontend приложения, АКЦЕНТИРУЮ внимание образ должен быть указан именно тот который запускает приложение из исходников. Так как нам надо иметь возможность задавать REACT_APP_BACKEND_URL в будущем.

```
➜  k8s git:(k8s_intro) ✗ kubectl apply -f
deployment_frontend.yml
deployment.apps/fe-realworld created
```

- Пробуем проверить наличие контейнеров и их успешный запуск (ВОЗМОЖНО потребует время для загрузки Вашего docker image), ждем и проверяем

```
➜  k8s git:(k8s_intro) ✗ kubectl get deployment
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
be-realworld   2/2     2            2           5m25s
fe-realworld   2/2     2            2           2m24s
```

Пробуем связать наш frontend & backend

- Как помним frontend-приложение наше, является SPA и рендериться на стороне клиента и общается с backend с его адреса. Это значит что нам надо предоставить доступ из вне кластера!!! Для этого отлично подойдет INGRESS. Но по умолчанию в kubernetes отсутствует ingress-controller (если его конечно не предустановил какой то провайдер, например облачный). В нашем случае мы может доустановить его как плагин в minikube.

```
    ➜  k8s git:(k8s_intro) ✗ minikube addons enable ingress -p
first-cluster
```

```
    ▪ Using image k8s.gcr.io/ingress-nginx/controller:v0.44.0
    ▪ Using image docker.io/jettech/kube-webhook-certgen:v1.5.1
    ▪ Using image docker.io/jettech/kube-webhook-certgen:v1.5.1
🔎  Verifying ingress addon...
🌟  The 'ingress' addon is enabled
```

- По что бы создать ingress для нашего frontend & backend, они должны быть опубликованы внутри кластер через service, создадим и опубликуем их.
  - Создаем файл манифеста для frontend

```
➜  k8s git:(k8s_intro) ✗ cat svc_frontend.yml
---
apiVersion: v1
kind: Service
metadata:
name: fe-realworld
labels:
    app: realworld
    type: frontend
spec:
selector:
    app: realworld
    type: frontend
ports:
- protocol: TCP
    port: 80
    targetPort: 4100
```

  - Создаем файл манифеста для backend

```
➜  k8s git:(k8s_intro) ✗ cat svc_backend.yml
---
apiVersion: v1
kind: Service
metadata:
name: be-realworld
labels:
    app: realworld
    type: backend
spec:
selector:
    app: realworld
    type: backend
ports:
- protocol: TCP
    port: 8081
    targetPort: 8081
```

  - Применяем оба

```
➜  k8s git:(k8s_intro) ✗ kubectl apply -f svc_frontend.yml
service/fe-realworld created
➜  k8s git:(k8s_intro) ✗ kubectl apply -f svc_backend.yml
service/be-realworld created
```

- Проверяем их наличие

```
➜  k8s git:(k8s_intro) ✗ kubectl get service
NAME           TYPE        CLUSTER-IP     EXTERNAL-IP
PORT(S)     AGE
be-realworld   ClusterIP   10.107.72.81   <none>
8081/TCP    34s
fe-realworld   ClusterIP   10.110.5.185   <none>
80/TCP      40s
kubernetes     ClusterIP   10.96.0.1      <none>
443/TCP     84m
mongo          ClusterIP   None           <none>
27017/TCP   17m
```

- Напишем файл манифеста ingress для frontend

```
➜  k8s git:(k8s_intro) ✗ cat ingress_frontend.yml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
name: fe-realworld
spec:
rules:
- host: "realworld.local.io"
    http:
    paths:
    - pathType: Prefix
        path: "/"
        backend:
        service:
            name: fe-realworld
            port:
            number: 80
```

- Напишем файл манифеста ingress для backend

```
➜  k8s git:(k8s_intro) ✗ cat ingress_backend.yml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
name: be-realworld
spec:
rules:
- host: "backend.realworld.local.io"
    http:
    paths:
    - pathType: Prefix
        path: "/"
        backend:
        service:
            name: be-realworld
            port:
            number: 8081
```

- Применяем оба манифеста

```
➜  k8s git:(k8s_intro) ✗ kubectl apply -f
ingress_frontend.yml
ingress.networking.k8s.io/fe-realworld configured
➜  k8s git:(k8s_intro) ✗ kubectl apply -f
ingress_backend.yml
ingress.networking.k8s.io/be-realworld created
```

- Проверяем наличи

```
➜  k8s git:(k8s_intro) ✗ kubectl get ingress
NAME           CLASS    HOSTS
ADDRESS         PORTS   AGE
be-realworld   <none>   backend.realworld.local.io
192.168.64.18   80      82s
fe-realworld   <none>   realworld.local.io
192.168.64.18   80      6m24s
➜  k8s git:(k8s_intro) ✗
```

- Проверим адрес нашей виртуальной машины где работает кластер, который должен совпадать с адресом назначенный в ingress

```
➜  k8s git:(k8s_intro) ✗ minikube ip -p first-cluster
192.168.64.18
```

- Существует ПРОБЛЕМА (задача) наш хост ничего не знает об внутренней адресации kubernetes cluster, по этому добавим доменные имена указанные в ingress в наш /etc/hosts файла на хоста.

```
➜  k8s git:(k8s_intro) ✗ cat /etc/hosts | grep local.io
192.168.64.18 realworld.local.io
192.168.64.18 backend.realworld.local.io
192.168.64.18 adminmongo.realworld.local.io
```

Финально смотрим все запущенные pod-ы.

- Проверяем pods

```
➜  k8s git:(k8s_intro) ✗ kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
be-realworld-6f49664975-jpmkf     1/1     Running   0          29m
be-realworld-6f49664975-xsmj7     1/1     Running   0          29m
fe-realworld-fdf94f948-9v27d      1/1     Running   0          26m
fe-realworld-fdf94f948-ptq7c      1/1     Running   0          26m
mongo-realworld-0                 1/1     Running   0          39m
```

- Проверяем services

```
➜  k8s git:(k8s_intro) ✗ kubectl get services
NAME           TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)
AGE
be-realworld   ClusterIP   10.107.72.81   <none>        8081/TCP
16m
fe-realworld   ClusterIP   10.110.5.185   <none>        80/TCP
```

```
16m
kubernetes     ClusterIP   10.96.0.1      <none>        443/TCP
100m
mongo          ClusterIP   None           <none>
27017/TCP   33m
```

- Проверяем ingress

```
➜  k8s git:(k8s_intro) ✗ kubectl get ingress
NAME           CLASS    HOSTS                       ADDRESS
PORTS   AGE
be-realworld   <none>   backend.realworld.local.io
192.168.64.18   80      8m52s
fe-realworld   <none>   realworld.local.io
192.168.64.18   80      13m
```

- Проверяем наш маленький но ГОРДЫЙ Persisten Volume

```
➜  k8s git:(k8s_intro) ✗ kubectl get pvc
NAME                          STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS      AGE
mongo-pvc                     Bound    pvc-931fdaff-4d91-4893-
9c4b-e72510af30f4   1Gi        RWO            csi-hostpath-sc
43m
mongo-pvc-mongo-realworld-0   Bound    pvc-a749ff6d-7331-4311-
95f0-ec4ece147263   1Gi        RWO
csi-hostpath-sc   41m
➜  k8s git:(k8s_intro) ✗ kubectl get pv
NAME                                       CAPACITY   ACCESS
MODES   RECLAIM POLICY   STATUS   CLAIM
STORAGECLASS      REASON   AGE
pvc-931fdaff-4d91-4893-9c4b-e72510af30f4   1Gi        RWO
Delete           Bound    default/mongo-pvc
csi-hostpath-sc            43m
pvc-a749ff6d-7331-4311-95f0-ec4ece147263   1Gi        RWO
Delete           Bound    default/mongo-pvc-mongo-realworld-0
csi-hostpath-sc            41m
➜  k8s git:(k8s_intro) ✗ kubectl get storageclass
NAME                 PROVISIONER                RECLAIMPOLICY
VOLUMEBINDINGMODE   ALLOWVOLUMEEXPANSION   AGE
csi-hostpath-sc      hostpath.csi.k8s.io        Delete
Immediate           false                  51m
standard (default)   k8s.io/minikube-hostpath   Delete
Immediate           false                  102m
```

ПРОВЕРЯЕМ ПРИЛОЖЕНИЕ

- Обращаемся с хоста на адрес http://realworld.local.io/register и пройдем регистрацию, так как приложен будет выдавать ошибки из некорретность авторизации.
- Пробуем опубликовать новый пост.

Самостоятельное задание (Если не получается можете воспользовать инструкцией дальше)

```
The beginning of emptiness









The ending of emptiness
```

- Создаем deployment

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
name: admin-mongo
labels:
    app: realworld
    type: admin-mongo
spec:
replicas: 1
```

```
selector:
    matchLabels:
    app: realworld
    type: admin-mongo
template:
    metadata:
    labels:
        app: realworld
        type: admin-mongo
    spec:
    containers:
    - name: admin-mongo
        image: adicom/admin-mongo
        ports:
        - containerPort: 8082
        imagePullPolicy: Always
        env:
        - name: PORT
        value: "8082"
        - name: DB_HOST
        value: "mongo"
```

- Создаем service

```
---
apiVersion: v1
kind: Service
metadata:
name: admin-mongo
labels:
    app: realworld
    type: admin-mongo
spec:
selector:
    app: realworld
    type: admin-mongo
ports:
- protocol: TCP
    port: 8082
    targetPort: 8082
```

- Создаем ingress

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
name: admin-mongo
spec:
rules:
- host: "adminmongo.realworld.local.io"
    http:
    paths:
    - pathType: Prefix
        path: "/"
        backend:
        service:
```

```
            name: admin-mongo
            port:
            number: 8082
```

- Применяем манифесты

```
➜  k8s git:(k8s_intro) ✗ kubectl apply -f
deployment_admin_mongo.yml
deployment.apps/admin-mongo created
➜  k8s git:(k8s_intro) ✗ kubectl apply -f svc_admin_mongo.yml
service/admin-mongo created
➜  k8s git:(k8s_intro) ✗ kubectl apply -f
ingress_admin_mongo.yml
ingress.networking.k8s.io/admin-mongo created
```

- ПРОВЕРЯЕМ ПОДКЛЮЧЕНИЕ по соответствующему URL и создаем подключение и проверяем наличие записей в БД.