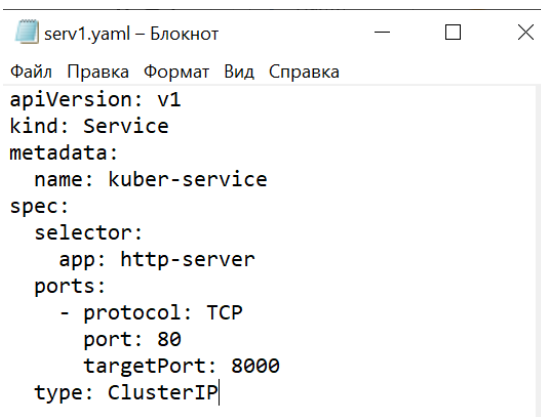


3. Service

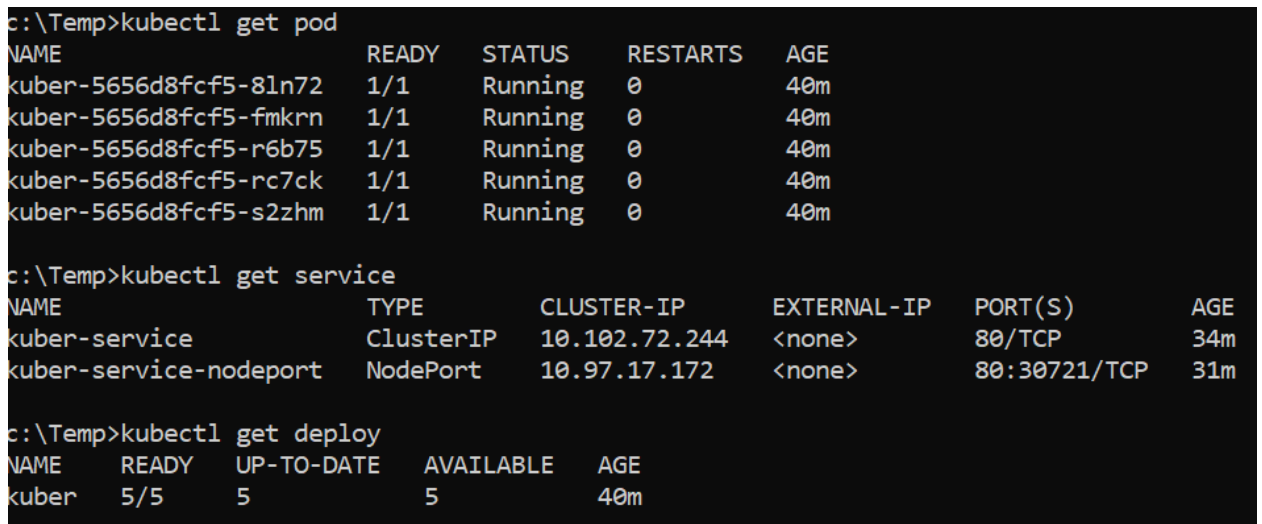
3.a) Service - объект K8s, который создается для того, чтобы сформировать единую постоянную точку входа группы подов, предоставляющее одно и то же приложение. Т.е. каждая служба имеет IP адрес и Port, которые не меняются до тех пор пока служба существует. Мы рассмотрим только ClusterIP и Node. externalname - работа с DNS, LoadBalancer - только Cloud Clusters

ClusterIP (ip доступно только внутри кластера): selector.app должен соответствовать labels в подах. Данный сервис подгружается для предыдущего deployment(ЛР3 deploy2)



```
serv1.yaml - Блокнот
Файл Правка Формат Вид Справка
apiVersion: v1
kind: Service
metadata:
  name: kuber-service
spec:
  selector:
    app: http-server
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8000
  type: ClusterIP
```

Что у нас есть на данный момент



```
c:\Temp>kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE
kuber-5656d8fcf5-8ln72              1/1     Running   0           40m
kuber-5656d8fcf5-fmkrrn             1/1     Running   0           40m
kuber-5656d8fcf5-r6b75              1/1     Running   0           40m
kuber-5656d8fcf5-rc7ck              1/1     Running   0           40m
kuber-5656d8fcf5-s2zhm              1/1     Running   0           40m

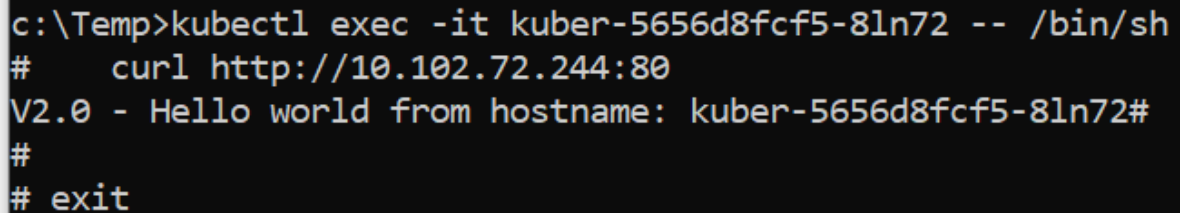
c:\Temp>kubectl get service
NAME                                TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
kuber-service                      ClusterIP      10.102.72.244   <none>        80/TCP           34m
kuber-service-nodeport              NodePort       10.97.17.172    <none>        80:30721/TCP     31m

c:\Temp>kubectl get deploy
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
kuber   5/5     5            5           40m
```

Выполним команду

```
kubectl exec -it kuber-5656d8fcf5-8ln72 -- /bin/sh  
curl http://10.102.72.244:80
```

(Запуск обеих команд позволяет вам проверять и тестировать доступность вашего сервиса изнутри пода, что является полезным для диагностики проблем с доступом или функциональностью вашего приложения)



```
c:\Temp>kubectl exec -it kuber-5656d8fcf5-8ln72 -- /bin/sh  
# curl http://10.102.72.244:80  
V2.0 - Hello world from hostname: kuber-5656d8fcf5-8ln72#  
#  
# exit
```

Рассмотрим подробнее, что происходит

kubectl exec: Это команда Kubernetes, используемая для выполнения команд внутри контейнера, работающего в Pod

-it:

- **-i:** Позволяет поддерживать интерактивный режим, что позволяет взаимодействовать с командной оболочкой (shell) в контейнере.
- **-t:** Создает псевдотерминал, что делает взаимодействие более удобным, выдавая визуальные подсказки, похожие на те, что можно получать непосредственно в терминале компьютера.

kuber-5656d8fcf5-8ln72: имя пода, внутри которого будет выполнена команда. Передаём конкретный под, чтобы получить доступ к его терминалу.

-- /bin/sh: команда, которую хотим выполнить внутри контейнера (в данном случае это командная оболочка sh). Обратите внимание, что **--**, он нужен для указания конца опций командной строки для **kubectl exec**, чтобы определить, что последующие элементы являются командой для выполнения внутри контейнера.

Когда выполняется эта команда, то мы получаем доступ к оболочке `sh` внутри контейнера `kuber-5656d8fcf5-8ln72`. Теперь можно выполнять любые команды Linux внутри этого контейнера.

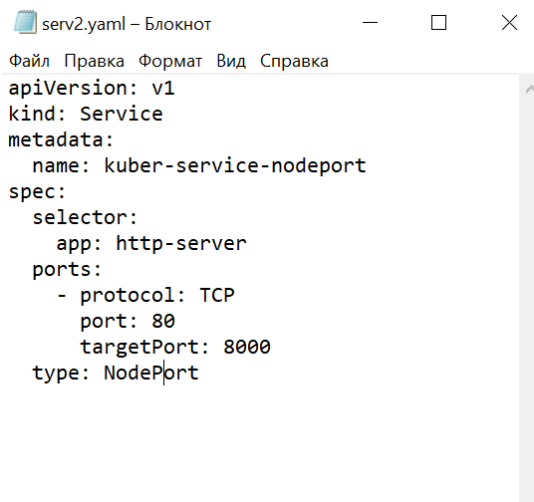
`curl`: утилита командной строки для выполнения HTTP-запросов. Она может быть использована для отправки различных видов запросов к URL.

`http://10.102.72.244:80`: URL, к которому будет произведён запрос. В данном случае это ClusterIP адрес нашего сервиса на порту 80.

Таким образом, если выполнить `curl http://10.102.72.244:80`, то отправим HTTP-запрос к сервису, который работает в кластере Kubernetes, и ожидаем получить ответ от нашего приложения, работающего на данном сервисе, что и происходит

```
c:\Temp>kubectl exec -it kuber-5656d8fcf5-8ln72 -- /bin/sh
# curl http://10.102.72.244:80
V2.0 - Hello world from hostname: kuber-5656d8fcf5-8ln72#
#
# exit
```

3.b) NodePort (доступ извне): Два примера, что будет, если автоматически nodeport предоставит порт (30721 в данном случае) и если зададим мы (30080).
Диапазон от 30000 до 32767



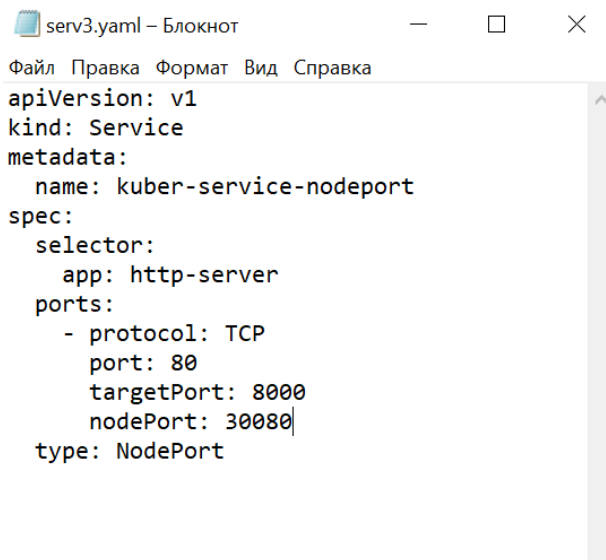
```
serv2.yaml - Блокнот
Файл  Правка  Формат  Вид  Справка
apiVersion: v1
kind: Service
metadata:
  name: kuber-service-nodeport
spec:
  selector:
    app: http-server
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8000
  type: NodePort
```

```
c:\Temp>kubectl get service
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kuber-service	ClusterIP	10.102.72.244	<none>	80/TCP	34m
kuber-service-nodeport	NodePort	10.97.17.172	<none>	80:30721/TCP	31m



V2.0 - Hello world from hostname: kuber-5656d8fcf5-s2zhm



```
serv3.yaml - Блокнот
Файл Правка Формат Вид Справка
apiVersion: v1
kind: Service
metadata:
  name: kuber-service-nodeport
spec:
  selector:
    app: http-server
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8000
      nodePort: 30080
  type: NodePort
```

← → ↻ ⓘ localhost:30080

/2.0 - Hello world from hostname: kuber-5656d8fcf5-r6b75

3.с)

Пример с 2 контейнерами и 2 сервисами.

YAML-файл определяет Deployment, который будет поддерживать три реплики подов, каждый из которых будет запускать два контейнера: один с PHP, слушающий порт 80, и другой с Tomcat, слушающий порт 8080. Это позволяет создать многоконтейнерные приложения в Kubernetes, где разные контейнеры могут выполнять различные задачи или служить различным логическим частям приложения.



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deployment-multi
  labels:
    app: my-k8s-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      project: cia
  template:
    metadata:
      labels:
        project: cia
    spec:
      containers:
        - name: my-web
          image: adv4000/k8sphp:latest
          ports:
            - containerPort: 80 #port on Pod
        - name: not-my-web
          image: tomcat:8.5.38
          ports:
            - containerPort: 8080 #port on Pod
```

Пояснения `deploy5.yaml` (часть `Deployment`)

`apiVersion: apps/v1`: версия API, используемая для этого объекта. В данном случае — это `apps/v1`, что является актуальной версией для `Deployment`.

`kind: Deployment`: тип ресурса, который создается. В данном случае это объект `Deployment`, который управляет развертыванием подов (`Pods`) и гарантирует, что заданное количество реплик ваших подов всегда доступно.

`metadata`: Метаданные объекта, такие как имя и метки.

- `name: deployment-multi`: имя объекта `Deployment`.

- `labels`: метки, которые могут использоваться для поиска и управления объектами. В данном случае метка `app` установлена на `my-k8s-deployment`.

`spec`: основная спецификация `Deployment`-а, где это определяет его поведение.

- `replicas: 3`: указывает, что Kubernetes должен поддерживать три реплики (экземпляра) пода. Это значит, что будет запущено три одинаковых пода, чтобы обеспечивать доступность приложения.

`selector`: определяет, какие поды являются частью этого `Deployment`-а.

- `matchLabels`: определяет метки, которые должны совпадать. В данном случае `project: sia`. Это означает, что `Deployment` будет управлять подами, у которых есть эта метка.

`template`: Шаблон для создания подов.

- `metadata`: метаданные, которые будут применены к подам, созданным этим `Deployment`-ом.

- `labels`: метки, которые будут применены к создаваемым подам. Здесь метка `project: sia` идентична той, что используется в селекторе.

`spec (внутри template)`: Спецификация самих подов.

- `containers`: список контейнеров, которые будут запущены в каждом поде.

- Первый контейнер:

- `name: my-web`: имя контейнера.

- `image: adv4000/k8sphp:latest`: Docker-образ, из которого будет создан контейнер. В данном случае это образ PHP.
- `ports`: какие порты будут открыты в контейнере.
- `containerPort: 80`: контейнер будет слушать порт 80.

- Второй контейнер:
 - `name: not-my-web`: имя контейнера.
 - `image: tomcat:8.5.38`: Docker-образ для контейнера Tomcat версии 8.5.38.
 - `ports`: какие порты будут открыты в контейнере:
 - `containerPort: 8080`: контейнер будет слушать порт 8080.

##продолжение deploy5.yaml

Эта часть описывает сервис под названием multi-service, который направляет трафик на поды с меткой project: sia. Сервис доступен извне по порту 30001 для приложения, слушающего порт 80, и по порту 30002 для другого приложения, слушающего порт 8000, которое будет доступно на порту 8888 сервиса



```
deploy5.yaml – Блокнот
Файл  Правка  Формат  Вид  Справка

ports:
- containerPort: 8080 #port on Pod

---
apiVersion: v1
kind: Service
metadata:
  name: multi-service
  labels:
    env: prod
    owner: user
spec:
  type: NodePort
  selector:
    project: cia
  ports:
    - name: my-web-app-listener
      protocol: TCP
      port: 80
      targetPort: 80
      nodePort: 30001
    - name: not-my-web-app-listener
      protocol: TCP
      port: 8888
      targetPort: 8000
      nodePort: 30002
```

Пояснения deploy5.yaml (часть Service)

apiVersion: v1: версия API Kubernetes, используемая для этого объекта. В данном случае это v1, что актуально для ресурсов типа Service.

kind: Service: тип ресурса, который создается.

metadata: метаданные объекта

- **name: multi-service:** имя объекта сервиса.
- **labels:** метки, которые могут использоваться для поиска и управления объектами, а также для применения селекторов. В этом случае установлены метки `env: prod` и `owner: user`.

spec: Основная спецификация сервиса, где определяется его поведение.

- **type: NodePort**: сервис будет доступен через тип NodePort. Это означает, что сервис будет доступен на определённом порту на каждом узле (node) кластера, а Kubernetes будет перенаправлять трафик к соответствующим подам. Это позволяет получать доступ к сервису извне кластера.

selector: указывает, какие поды этот сервис будет обслуживать.

- **project: cia**: сервис будет направлять трафик на поды, которые имеют метку project: cia. Это фактически является связующим звеном между сервисом и подами, которые он обслуживает.

ports: список портов, на которых сервис будет слушать.

- Первый порт:

- **name: my-web-app-listener**: имя порта.
- **protocol: TCP**: протокол, используемый этим портом (TCP).
- **port: 80**: порт, который будет использоваться для доступа к сервису. Это порт, на котором другие сервисы или клиенты будут отправлять запросы к сервису.
- **targetPort: 80**: порт на подах, на который будет перенаправлен трафик. Это означает, что трафик, приходящий на порт 80 сервиса, будет направлен на порт 80 в подах.
- **nodePort: 30001**: порт, на котором этот сервис будет доступен на каждом узле кластера. Вы можете подключиться к сервису, отправив запрос на любой узел кластера по этому порту.

- Второй порт:

- **name: not-my-web-app-listener**: Имя этого порта.
- **protocol: TCP**: используемый протокол (TCP).
- **port: 8888**: порт, используемый для доступа к этому сервису.
- **targetPort: 8000**: порт на подах, на который будет перенаправлен трафик. Это означает, что трафик, приходящий на порт 8888 сервиса, будет направлен на порт 8000 в подах.
- **nodePort: 30002**: порт, на котором этот сервис будет доступен на каждом узле кластера.

```
c:\Temp>kubectl apply -f deploy5.yaml
deployment.apps/deployment-multi unchanged
service/multi-service created

c:\Temp>kubectl get svc
NAME                TYPE        CLUSTER-IP      EXTERNAL-IP  PORT(S)          AGE
kuber-service       ClusterIP   10.102.72.244   <none>       80/TCP           90m
kuber-service-nodeport NodePort    10.97.17.172    <none>       80:30080/TCP     88m
multi-service       NodePort    10.104.173.136  <none>       80:30001/TCP,8888:30002/TCP 7m34s
```

← → ↻ ⚠ Не защищено 10.24.29.27:30001

Hello from Kubernetes


Server IP Address is: 10.42.0.239

10.24.29.27:30002

Home Documentation Configuration Examples Wiki

Apache Tomcat/8.5.38

If you're seeing this, you've succeeded



Recommended Reading:
[Security Considerations HOW-TO](#)
[Manager Application HOW-TO](#)
[Clustering/Session Replication HOW-TO](#)

4. Liveness, readiness, Volume

Эти элементы помогают управлять состоянием приложений и их жизненным циклом в контейнеризированной среде

4.a) LivenessProbe (Проверки живучести) предназначены для определения, находится ли контейнер в состоянии, когда его следует перезапустить. Если проверка живучести не проходит, Kubernetes перезапускает контейнер. Это полезно, например, в случаях, когда приложение зависло или перестало отвечать.. Есть 3 механизма проверок: exec, http get, TCP.

deploy-liv.yaml – Блокнот

Файл Правка Формат Вид Справка

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ubuntu
  labels:
    app: ubuntu
spec:
  replicas: 1
  selector:
    matchLabels:
      app: ubuntu
  template:
    metadata:
      labels:
        app: ubuntu
    spec:
      containers:
        - name: ubuntu
          image: ubuntu
          args:
            - /bin/sh
            - -c
            - touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600
          livenessProbe:
            exec:
              command:
                - cat
                - /tmp/healthy
            initialDelaySeconds: 5
            periodSeconds: 5
            timeoutSeconds: 1
            successThreshold: 1
            failureThreshold: 3
```

initialDelaySeconds: 5 # кол-во сек от старта контейнера до запуска LivenessProbe

periodSeconds: 5 # кол-во секунд между пробами

timeoutSeconds: 1 # кол-во секунд ожидания пробы

successThreshold: 1 # мин кол-во проверок, чтобы проба считалась неудачной

failureThreshold: 3 # сколько раз сделает команду, чтобы считать контейнер умершим и перезапустить

Создаем образ ubuntu, создаем внутри файл healthy, после чего через время удалим его. Проверка будет типа exes (команды), которая будет проверять cat существует ли файл или нет. Если код равен 0 - то все хорошо, если 1 - ошибка.

```
c:\Temp>kubectl apply -f deploy-liv.yaml
deployment.apps/ubuntu created
```

```
c:\Temp>kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
deployment-multi-5d9c6d866c-bhv6h	2/2	Running	0	10m
deployment-multi-5d9c6d866c-n9hlm	2/2	Running	0	10m
deployment-multi-5d9c6d866c-tn7mq	2/2	Running	0	10m
ubuntu-989c968bb-v92wk	1/1	Running	0	16s

```
c:\Temp>kubectl exec -it ubuntu-989c968bb-v92wk -- /bin/bash
```

```
root@ubuntu-989c968bb-v92wk:/# ls -a /tmp
```

```
.. healthy
```

```
root@ubuntu-989c968bb-v92wk:/# ls -la /tmp
```

```
total 8
```

```
drwxrwxrwt 1 root root 4096 Nov 14 05:44 .
```


```
drwxr-xr-x 1 root root 4096 Nov 14 05:43 ..
```

							Pods(test)[1]	
NAME ↑	PF	READY	STATUS	RESTARTS	CPU	MEM	%CPU/R	%
ubuntu-7d478f46fc-dv9qh	●	1/1	Running	3	2	0	n/a	

```
Successfully pulled image "ubuntu" in 1.151s (1.151s including waiting). Image size: 29754422 bytes.
Successfully pulled image "ubuntu" in 1.126s (1.126s including waiting). Image size: 29754422 bytes.
Liveness probe failed: cat: /tmp/healthy: No such file or directory
Container ubuntu failed liveness probe, will be restarted
Pulling image "ubuntu"
Created container ubuntu
Successfully pulled image "ubuntu" in 1.157s (1.157s including waiting). Image size: 29754422 bytes.
Started container ubuntu
```

```
c:\Temp>kubectl delete deploy ubuntu
deployment.apps "ubuntu" deleted
```

4.b) ТСП проверка подключения. Если подключение успешно установлено - то все хорошо. Здесь специально в livenessProbe идет ошибочный порт 8001 (а не 8000), чтобы показать, что контейнер перезагрузится.

 deploy-liv-tcp.yaml – Блокнот — □

Файл Правка Формат Вид Справка

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kuber-tcp
  labels:
    app: kuber
spec:
  replicas: 1
  selector:
    matchLabels:
      app: http-server-tcp
  template:
    metadata:
      labels:
        app: http-server-tcp
    spec:
      containers:
      - name: kuber-app
        image: bakavets/kuber:v1.0
        ports:
        - containerPort: 8000
        livenessProbe:
          tcpSocket:
            port: 8001
          initialDelaySeconds: 15
          periodSeconds: 10
          timeoutSeconds: 1
          successThreshold: 1
          failureThreshold: 3
```

```
---
apiVersion: v1
kind: Service
metadata:
  name: kuber-service-tcp
spec:
  selector:
    app: http-server-tcp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8000
    nodePort: 30002
  type: NodePort
```

```
c:\Temp>kubectl apply -f deploy-liv-tcp.yaml
deployment.apps/kuber-tcp created
service/kuber-service-tcp created
```

```
c:\Temp>kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
kuber-tcp-659f475685-td4k8	1/1	Running	0	5s

```
c:\Temp>kubectl describe pod kuber-tcp-659f475685-td4k8
```

```
Name:          kuber-tcp-659f475685-td4k8
Namespace:     test
Priority:       0
```

```
Message
```


```
-----
```

```
Successfully assigned test/kuber-tcp-659f475685-td4k8 to docker-desktop
Container image "bakavets/kuber:v1.0" already present on machine
Created container kuber-app
Started container kuber-app
Liveness probe failed: dial tcp 10.1.0.103:8001: connect: connection refused
```

```
c:\Temp>kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
kuber-tcp-659f475685-td4k8	1/1	Running	0	31s

4.c) HttpGet делает запрос на порт. Если сервис не отвечает или передает код с ошибкой - контейнер автоматически перезапустится. Тут образ после 5 обращений к сервису выдает ошибку. Так было специально сделано для проверки

 deploy-liv-http.yaml – Блокнот

Файл Правка Формат Вид Справка

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kuber-http
  labels:
    app: kuber
spec:
  replicas: 1
  selector:
    matchLabels:
      app: http-server-http
  template:
    metadata:
      labels:
        app: http-server-http
    spec:
      containers:
        - name: kuber-app
          image: bakavets/kuber:v1.0-unhealthy
          ports:
            - containerPort: 8000
          livenessProbe:
            httpGet:
              path: /healthcheck
              port: 8000
            initialDelaySeconds: 5
            periodSeconds: 5
```

```
---
apiVersion: v1
kind: Service
metadata:
  name: kuber-service-http
spec:
  selector:
    app: http-server-http
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8000
      nodePort: 30002
  type: NodePort
```

```
mnagapetyan@mnagapetyan-ThinkPad-P1-Gen-6:~/ABX$ kubectl apply -f liv_http_dep.yaml
deployment.apps/kuber-http created
service/kuber-service-http created
mnagapetyan@mnagapetyan-ThinkPad-P1-Gen-6:~/ABX$ kubectl get po
NAME                                READY   STATUS             RESTARTS   AGE
kuber-http-84b99dbf5f-b2cxf         0/1     ContainerCreating   0           4s
```



```
c:\Temp>kubectl apply -f deploy-liv-http.yaml
deployment.apps/kuber-http created
service/kuber-service-http created
```

```
c:\Temp>kubectl get pod
NAME                                READY   STATUS             RESTARTS   AGE
kuber-http-84b99dbf5f-1x18b        0/1     ContainerCreating   0           3s
```

```
c:\Temp>kubectl describe pod kuber-http-84b99dbf5f-1x18b
Name:                                kuber-http-84b99dbf5f-1x18b
Namespace:                          test
Priority:                             0
```

```
Events:
  Type     Reason      Age   From          Message
  ----     -
Normal    Scheduled   63s   default-scheduler   Successfully assigned test/kuber-http-84b99dbf5f-1x18b to docker-desktop
Normal    Pulling     62s   kubelet          Pulling image "bakavets/kuber:v1.0-unhealthy"
Normal    Pulled      10s   kubelet          Successfully pulled image "bakavets/kuber:v1.0-unhealthy" in 52.337s (52.33
112 bytes.
Normal    Created     8s    kubelet          Created container kuber-app
Normal    Started     8s    kubelet          Started container kuber-app
```

```
c:\Temp>kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE
kuber-http-84b99dbf5f-1x18b        1/1     Running   0           72s
```

```
c:\Temp>kubectl delete -f deploy-liv-http.yaml
deployment.apps "kuber-http" deleted
service "kuber-service-http" deleted
```

4.d) Readiness - используются для определения, готово ли приложение принимать трафик. Если проверка готовности не проходит, Kubernetes не отправляет на этот контейнер запросы от сервиса, то есть идентично прошлому, только liveness отвечает за живучесть контейнера (просто его перезапускает), то Readiness понимает, когда регистрировать порт к сервису, чтобы начать принимать трафик.

```
GNU nano 6.2
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kuber-default
  labels:
    app: kuber
spec:
  replicas: 1
  selector:
    matchLabels:
      app: http-server-default
  template:
    metadata:
      labels:
        app: http-server-default
    spec:
      containers:
        - name: kuber-app
          image: bakavets/kuber:v1.0
          ports:
            - containerPort: 8000
---
apiVersion: v1
kind: Service
metadata:
  name: kuber-default-service
spec:
  selector:
    app: http-server-default
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8000
      nodePort: 30004
  type: NodePort
```

Создадим 2 файл с теми же label, только он уже будет “бракованный”, как и с LivenessProbe.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kuber-http-readinessprobe
  labels:
    app: kuber
spec:
  replicas: 1
  selector:
    matchLabels:
      app: http-server-default
  template:
    metadata:
      labels:
        app: http-server-default
    spec:
      containers:
        - name: kuber-app
          image: bakavets/kuber:v1.0-unhealthy
          ports:
            - containerPort: 8000
          readinessProbe:
            httpGet:
              path: /healthcheck
              port: 8000
            initialDelaySeconds: 5
            periodSeconds: 5
          livenessProbe:
            httpGet:
              path: /healthcheck
              port: 8000
            initialDelaySeconds: 5
            periodSeconds: 5
```

Сперва запускается здоровый под, вследствие чего трафик поступал на него. После регистрации 2 - уже и 2 файл принимает трафик, но так как тот упал, то снова трафик вернулся к 1.

← → ↻ ⚠ Не защищено 10.24.29.27:30004

V1.0 - Hello world from hostname: kuber-default-678c8898b-9nfxc

← → ↻ ⚠ Не защищено 10.24.29.27:30004

Hello world from hostname: kuber-http-readinessprobe-5fc675c967-65rkv

← → ↻ ⚠ Не защищено 10.24.29.27:30004

V1.0 - Hello world from hostname: kuber-default-678c8898b-9nfxc

4.e) Volume: предоставляют способ хранения данных, который переживает перезапуск контейнеров и обеспечивает совместное использование данных между контейнерами.

Самым простым типом является EmptyDir. Выделение пространства на ноде, где будут запускаться поды. Указываем сперва volume с названием и типом. Дальше указываем, куда мы хотим встроить в nginx (/cache-2 - будет создана автоматически) + имя созданного volume (cache-volume)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kuber
  labels:
    app: kuber
spec:
  replicas: 3
  selector:
    matchLabels:
      app: http-server
  template:
    metadata:
      labels:
        app: http-server
    spec:
      containers:
        - name: kuber-app-1
          image: bakavets/kuber
          ports:
            - containerPort: 8000
          volumeMounts:
            - mountPath: /cache-1
              name: cache-volume
        - name: nginx
          image: nginx
          ports:
            - containerPort: 80
          volumeMounts:
            - mountPath: /cache-2
              name: cache-volume
      volumes:
        - name: cache-volume
          emptyDir: {}
```

```
mnagapetyan@mnagapetyan-ThinkPad-P1-Gen-6:~/ABX$ kubectl apply -f volume.yaml
deployment.apps/kuber created
```

```
mnagapetyan@mnagapetyan-ThinkPad-P1-Gen-6:~/ABX$ kubectl get po
NAME                                READY    STATUS    RESTARTS   AGE
kuber-8544d666b-bmxgb               2/2     Running   0           43s
kuber-8544d666b-mhbdh               2/2     Running   0           46s
kuber-8544d666b-rv6w6               2/2     Running   0           39s
```

```
mnagapetyan@mnagapetyan-ThinkPad-P1-Gen-6:~/ABX$ kubectl exec -it kuber-8544d666b-bmxgb -c kuber-app-1 -- /bin/bash
root@kuber-8544d666b-bmxgb:/# ls -la
total 84
drwxr-xr-x 1 root root 4096 Oct 19 16:08 .
drwxr-xr-x 1 root root 4096 Oct 19 16:08 ..
drwxr-xr-x 1 root root 4096 Sep 10 2020 bin
drwxr-xr-x 2 root root 4096 Jul 10 2020 boot
drwxrwxrwx 2 root root 4096 Oct 19 16:08 cache-1
drwxr-xr-x 5 root root 360 Oct 19 16:08 dev
drwxr-xr-x 1 root root 4096 Oct 19 16:08 etc
```

```
root@kuber-8544d666b-bmxgb:/# cd cache-1/
root@kuber-8544d666b-bmxgb:/cache-1# echo "Hello world" > hello.txt
root@kuber-8544d666b-bmxgb:/cache-1# ls -la
total 12
drwxrwxrwx 2 root root 4096 Oct 19 16:10 .
drwxr-xr-x 1 root root 4096 Oct 19 16:08 ..
-rw-r--r-- 1 root root 12 Oct 19 16:10 hello.txt
root@kuber-8544d666b-bmxgb:/cache-1#
```

```
mnagapetyan@mnagapetyan-ThinkPad-P1-Gen-6:~/ABX$ kubectl exec -it kuber-8544d666b-bmxgb -c nginx -- /bin/bash
root@kuber-8544d666b-bmxgb:/# ls
bin  cache-2  docker-entrypoint.d  etc  lib  media  opt  root  sbin  sys  usr
boot  dev  docker-entrypoint.sh  home  lib64  mnt  proc  run  srv  tmp  var
root@kuber-8544d666b-bmxgb:/# cd cache-2/
root@kuber-8544d666b-bmxgb:/cache-2# ls -la
total 12
drwxrwxrwx 2 root root 4096 Oct 19 16:10 .
drwxr-xr-x 1 root root 4096 Oct 19 16:08 ..
-rw-r--r-- 1 root root 12 Oct 19 16:10 hello.txt
root@kuber-8544d666b-bmxgb:/cache-2# cat hello.txt
Hello world
root@kuber-8544d666b-bmxgb:/cache-2#
```

Если поставить путь к существующему файлу, то директория станет пустой. Т.е затирает существующие файлы.