Pod. Docker

REPLICASET

DEPLOYMENT

SERVICE

Service discovery

SKAFFOLD

HEALTH CHECK

BLUEGREEN

INGRESS

STATEFULSETS

CONFIGURATION

HELM

HELM-DEP

Minikube

Установить minikube можно по ссылке
https://kubernetes.io/ru/docs/tasks/tools/install-minikube/

На MacOS X можно установить с помощью brew install minikube
Также устанавливаем kubectl: brew install kubectl

Установить kubectl можно по ссылке
https://kubernetes.io/ru/docs/tasks/tools/install-kubectl/

Кому интересно, могут посмотреть инструменты для работы с кубиком на Макоси - https://medium.com/@mo_keefe/a-kubernetes-development-workflow-for-macos-8c41669a4518

После установки можно посмотреть на версию кубика:

```
➜  ~ minikube version
minikube version: v1.9.0
commit: 48fefd43444d2f8852f527c78f0141b377b1e42a
```

```
➜  ~
```

Стартуем миникуб:

```
➜  ~ minikube start --cpus=6 --memory=6g --vm-driver=vmware
🎉  minikube 1.9.2 is available! Download it:
https://github.com/kubernetes/minikube/releases/tag/v1.9.2
💡  To disable this notice, run: 'minikube config set
WantUpdateNotification false'

🙄  minikube v1.9.0 on Darwin 10.15.1
    ▪ MINIKUBE_ACTIVE_DOCKERD=minikube
✨  Using the vmware driver based on user configuration
🔥  Creating vmware VM (CPUs=6, Memory=6144MB, Disk=20000MB) ...
🐳  Preparing Kubernetes v1.18.0 on Docker 19.03.8 ...
🌟  Enabling addons: default-storageclass, storage-provisioner
🏄  Done! kubectl is now configured to use "minikube"
➜  ~
```

Я использовал `vmware`, потому что она постабильнее опенсорсного `hyperkit`. Но крайне важно понимать, что `vwmware` платная.

Создается виртуальная машина и на ней разворачивается кластер кубернетеса из одной ноды. После установки `minikube` автоматически прописывает правильный конфиг для `kubectl`, так чтобы он работал с миникубом

Посмотрим ноды. `Minikube` - это кластер из одной машины, которая является мастером

```
➜  ~ kubectl get nodes
NAME       STATUS   ROLES    AGE   VERSION
minikube   Ready    master   99s   v1.18.0
```

Все приложения и сервисы в кластере находятся в отдельных `namespaces`. По умолчанию активен `namespace default`, в котором нет никаких ресурсов. Также есть несколько системных `namespace`-ов.

Создадим новый `namespace myapp`.

```
➜  ~ kubectl get namespaces
NAME              STATUS   AGE
default           Active   107s
kube-node-lease   Active   108s
```

```
kube-public       Active   108s
kube-system       Active   108s
➜  ~ kubectl create namespace myapp
namespace/myapp created
```

Чтобы во всех командах каждый раз явным образом не прописывать namespace, мы его сделаем namespace-ом по умолчанию. (Можно было использовать утилиту kubens)

```
➜  ~ kubectl config set-context --current --namespace=myapp
Context "minikube" modified.
```

Посмотрим все поды во всех неймспейсах (-A)

```
➜  ~ kubectl get pods -A
NAMESPACE     NAME                               READY   STATUS
RESTARTS   AGE
kube-system   coredns-66bff467f8-d6464           1/1     Running   0
3m24s
kube-system   coredns-66bff467f8-l6dqc           1/1     Running   0
3m24s
kube-system   etcd-minikube                      1/1     Running   0
3m25s
kube-system   kube-apiserver-minikube            1/1     Running   0
3m25s
kube-system   kube-controller-manager-minikube   1/1     Running   0
3m25s
kube-system   kube-proxy-wwq5l                   1/1     Running   0
3m24s
kube-system   kube-scheduler-minikube            1/1     Running   0
3m25s
kube-system   storage-provisioner                1/1     Running   0
3m30s
```

В macOS удобно следить за изменениями в namespace в реальном времени.

```
➜  ~ brew install watch
➜  ~ watch kubectl get all
```

Либо можно использовать флаг -w для kubectl

```
➜  ~ kubectl get pod -w
```

Единственным ограничением является, что такой флаг работает только с одним ресурсом.

Pod. Docker

Подробная информация про Pod.
https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/

Что такое под? Под - это один или несколько контейнеров, запускаемых и удаляемых, как единое целое, а также у них один сетевой неймспейс, поэтому они могут обращаться друг к другу по локалхосту. Некоторый аналог “процесса”. Чаще всего таким “процессом” является приложение.

Чтобы запустить свой сервис в кубике, его необходимо обернуть в контейнер. На ноде кубернетеса запущен свой демон Docker (отличный от того, который запущен на десктопе). Поэтому, чтобы image был доступен докеру можно 1) использовать внешний реестр image - dockerhub, selfhosted nexus и т.д. 2) билдить докером с ноды. 2) вариант возможен только в случае локальной разработки, для продовой не подходит, но чтобы понимать, как оно внутри устроено, давайте пойдем этим методом.

```
➜  ~ docker ps
Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is
the docker daemon running?
➜  ~
```

Получаем доступы к демону докера внутри миникубика.

```
➜  ~ minikube docker-env
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.176.128:2376"
export DOCKER_CERT_PATH="/Users/test/.minikube/certs"
export MINIKUBE_ACTIVE_DOCKERD="minikube"
```

```
# To point your shell to minikube's docker-daemon, run:
# eval $(minikube -p minikube docker-env)
➜  ~
```

```
➜  ~ export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.176.128:2376"
export DOCKER_CERT_PATH="/Users/test/.minikube/certs"
export MINIKUBE_ACTIVE_DOCKERD="minikube"
➜  ~
```

Теперь можно видеть, какие контейнеры уже запущены внутри кубика. Это системные сервисы.

```
➜  ~ docker ps
CONTAINER ID        IMAGE                  COMMAND
CREATED             STATUS              PORTS               NAMES
f5f6204145c8        67da37a9a360           "/coredns -conf /etc…"   7
minutes ago       Up 7 minutes
k8s_coredns_coredns-66bff467f8-l6dqc_kube-system_eae00aa8-4948-4280-8ce6-c
85695ca9c85_0
4c6031a49eb0        67da37a9a360           "/coredns -conf /etc…"   7
minutes ago       Up 7 minutes
k8s_coredns_coredns-66bff467f8-d6464_kube-system_e568412c-100d-48a3-81f7-9
c790fd1b76b_0
13bacc1e4ee1        k8s.gcr.io/pause:3.2   "/pause"                 7
minutes ago       Up 7 minutes
k8s_POD_coredns-66bff467f8-l6dqc_kube-system_eae00aa8-4948-4280-8ce6-c8569
5ca9c85_0
```

Давайте теперь соберем image простого приложения на питоне:

```
➜  hello-py git:(master) ✗ ls
```

```
Dockerfile       app.py           requirements.txt
```

```
➜  hello-py git:(master) ✗ ccat app.py
import os
import json

from flask import Flask
app = Flask(__name__)

@app.route("/health")
def health():
    return '{"status": "ok"}'

@app.route("/")
def hello():
    return 'Hello world from ' + os.environ['HOSTNAME'] + ' !'

if __name__ == "__main__":
    app.run(host='0.0.0.0',port='80')
```

```
➜  hello-py git:(master) ✗ ccat Dockerfile
FROM python:3.5-onbuild

EXPOSE 8000

ENTRYPOINT ["python", "/usr/src/app/app.py"]
➜  hello-py git:(master) ✗
```

```
➜  hello-py git:(master) ✗ ccat requirements.txt
Flask==1.1.2
➜  hello-py git:(master) ✗
```

Запускаем сборку image с тэгом hello-py:v1

```
➜  hello-py git:(master) ✗ docker build -t hello-py:v1 .
Sending build context to Docker daemon  64.51kB
Step 1/3 : FROM python:3.5-onbuild
```

```
3.5-onbuild: Pulling from library/python
d660b1f15b9b: Pull complete
46dde23c37b3: Pull complete
6ebaeb074589: Pull complete
e7428f935583: Pull complete
c8e35ae8bd3e: Pull complete
87155e60716b: Pull complete
d67c7146ff6c: Pull complete
8ea67584e567: Pull complete
2811c7fd9008: Pull complete
Digest:
sha256:aadb2dc6adfa3aeb43ab02a75d8a7e0a011a1b5dc6af0d04ba26835950ffc0ed
Status: Downloaded newer image for python:3.5-onbuild
# Executing 3 build triggers
 ---> Running in 7125824bd821
```

….

```
Step 3/3 : ENTRYPOINT ["python", "/usr/src/app/app.py"]
 ---> Running in 961f007ec9b5
Removing intermediate container 961f007ec9b5
 ---> b1a20d7c1da5
Successfully built b1a20d7c1da5
Successfully tagged hello-py:v1
```

Имеем image, который доступен миникубику, теперь мы можем запустить контейнер с этим image на кластере кубернетес (в нашем случае миникуб).

Сначала описываем спецификацию пода в виде yaml файла.

```
➜  pod cat pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: hello-demo
  labels:
    app: hello-demo
spec:
  containers:
  - name: hello-demo
    image: hello-py:v1
    ports:
```

```
      - containerPort: 80
```

Просим кубик создать ресурсы из файла.

```
➜  pod kubectl apply -f pod.yaml
pod/hello-demo created
➜  pod
```

Теперь мы можем увидеть, что наш под запущен в нашем неймспейсе. Если не прописывать немспейс в манифесте или не передавать через параметр `kubectl -n`, то будет задеплоен в тот немспейс, который в кубконфиге.

```
Every 2,0s: kubectl get pods      MacBook-Pro-2.local: Sun Apr  5 23:33:03
2020

NAME         READY   STATUS    RESTARTS   AGE
hello-demo   1/1     Running   0          60s
```

Можем детально посмотреть на основные параметры созданного пода.

```
➜  pod kubectl describe pod hello-demo
Name:         hello-demo
Namespace:    myapp
Priority:     0
Node:         minikube/192.168.176.128
Start Time:   Sun, 05 Apr 2020 23:32:03 +0300
Labels:       app=hello-demo
Annotations:  Status:  Running
IP:           172.17.0.4
IPs:
  IP:  172.17.0.4
Containers:
  hello-demo:
    Container ID:
docker://2d150fe910aec4d8b64ad7f57229b580a5f45be200ec4021ad23b02c40bd33b4
    Image:          hello-py:v1
```

```
    Image ID:
docker://sha256:b1a20d7c1da5b9d1b86da1a4598cf53c0415bfcf5c02a8952dbdd273cf
07e075
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Sun, 05 Apr 2020 23:32:03 +0300
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from
default-token-dxblc (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-dxblc:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-dxblc
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  92s   default-scheduler  Successfully assigned
myapp/hello-demo to minikube
  Normal  Pulled     92s   kubelet, minikube  Container image
"hello-py:v1" already present on machine
  Normal  Created    92s   kubelet, minikube  Created container hello-demo
  Normal  Started    92s   kubelet, minikube  Started container hello-demo
```

Для каждого пода выделяется отдельный ip адрес на ноде. Но доступен он только на самой ноде, снаружи ноды (в нашем случае виртуалки) его увидеть нельзя.

Заходим на виртуалку.

```
➜  pod minikube ssh
                         _             _
            _         _ ( )           ( )
  ___ ___  (_)  ___  (_)| |/')  _   _ | |_      __
/' _ ` _ `\| |/' _ `\| || , <  ( ) ( )| '_`\  /'__`\
| ( ) ( ) || || ( ) || || |\`\ | (_) || |_) )(  ___/
(_) (_) (_)(_)(_) (_)(_)(_) (_)`\___/'(_,__/'`\____)
```

И смотрим, что на этом ip и порту действительно отвечает наш сервис.

```
$ wget -qO- http://172.17.0.4:80/
Hello world from hello-demo !$
$
```

Выходим и удаляем под.

```
$ logout
➜  pod kubectl delete pod hello-demo
pod "hello-demo" deleted
➜  pod
```

Под переходит в состояние “Удаляюсь”

```
Every 2,0s: kubectl get pods      MacBook-Pro-2.local: Sun Apr  5 23:36:38
2020

NAME         READY   STATUS        RESTARTS   AGE
hello-demo   1/1     Terminating   0          4m35s
```

```
Every 2,0s: kubectl get pods      MacBook-Pro-2.local: Sun Apr  5 23:37:32
2020

No resources found in myapp namespace.
```

И через какое-то время совсем исчезает.

Под-ы в чистом виде практически никто не использует, потому что за подами надо приглядывать. Если под упадет, он сам по себе не поднимется. Кроме того, для отказоустойчивости мы захотим иметь несколько инстансов одного и того же сервиса, и если упадет нода (например), нам нужно, чтобы количество инстансов осталось прежним.

Эту проблему в некотором смысле решает такая сущность (ресурс) кубернетеса, как `replicaset`

REPLICASET

https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/

Давайте создадим `replicaset` для нашего сервиса.
Для начала составим описание ресурса в `yaml` формате

```
➜  replicaset ls
rs.yaml
```

```
➜  replicaset cat rs.yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
    name: hello-rs-demo
spec:
    replicas: 3
    selector:
       matchLabels:
          app: hello-demo
    template:
       metadata:
          labels:
             app: hello-demo
       spec:
          containers:
          - name: hello-demo
            image: hello-py:v1
            ports:
              - containerPort: 80
➜  replicaset
```

replicas - количество реплик.
selector - какие именно поды контролирует эта реплика сет
template - это шаблон пода, который будет создаваться в случае необходимости репликасетом.

Применим эту конфигурацию

```
➜  replicaset kubectl apply -f rs.yaml
replicaset.apps/hello-rs-demo created
➜  replicaset
```

Видим, что запустилось 3 пода. И также в качестве ресурса внутри неймспейса появилась реплика сет.

```
Every 2,0s: kubectl get all       MacBook-Pro-2.local: Sun Apr  5 23:41:23
2020

NAME                         READY   STATUS    RESTARTS   AGE
pod/hello-rs-demo-f52xv      1/1     Running   0          19s
pod/hello-rs-demo-g9qcx      1/1     Running   0          19s
pod/hello-rs-demo-qnzg5      1/1     Running   0          19s

NAME                               DESIRED   CURRENT   READY   AGE
replicaset.apps/hello-rs-demo      3         3         3       19s
```

Если мы удалим один из подов.

```
➜  replicaset kubectl delete pod hello-rs-demo-f52xv
pod "hello-rs-demo-f52xv" deleted
```

Репликасет практически сразу же поднимет еще один под, чтобы их количество совпадало с 3.

```
NAME                         READY   STATUS        RESTARTS   AGE
pod/hello-rs-demo-f52xv      1/1     Terminating   0          52s
pod/hello-rs-demo-g9qcx      1/1     Running       0          52s
pod/hello-rs-demo-qnzg5      1/1     Running       0          52s
pod/hello-rs-demo-wxfhz      1/1     Running       0          2s

NAME                               DESIRED   CURRENT   READY   AGE
replicaset.apps/hello-rs-demo      3         3         3       52s
```

Давайте теперь посмотрим, как работают метки (labels).
Мы видим, что всем подам были проставлены метки (из шаблона), а реплика сет за этими метками “смотрит”

```
➜  replicaset kubectl get pods --show-labels
NAME                  READY   STATUS    RESTARTS   AGE     LABELS
hello-rs-demo-g9qcx   1/1     Running   0          2m12s   app=hello-demo
hello-rs-demo-qnzg5   1/1     Running   0          2m12s   app=hello-demo
hello-rs-demo-wxfhz   1/1     Running   0          82s     app=hello-demo
➜  replicaset
```

Если мы удалим эту метку в одном из подов, то репликасет решит, что у нее всего 2 пода под управлением, и создаст еще один.

```
➜  replicaset kubectl label pod hello-rs-demo-g9qcx app-
pod/hello-rs-demo-g9qcx labeled
➜  replicaset
```

```
Every 2,0s: kubectl get all       MacBook-Pro-2.local: Sun Apr  5 23:44:08
2020

NAME                      READY   STATUS    RESTARTS   AGE
pod/hello-rs-demo-g9qcx   1/1     Running   0          3m4s
pod/hello-rs-demo-plhst   1/1     Running   0          20s
pod/hello-rs-demo-qnzg5   1/1     Running   0          3m4s
pod/hello-rs-demo-wxfhz   1/1     Running   0          2m14s

NAME                            DESIRED   CURRENT   READY   AGE
replicaset.apps/hello-rs-demo   3         3         3       3m4s
```

```
➜  replicaset kubectl get pods --show-labels
NAME                  READY   STATUS    RESTARTS   AGE     LABELS
hello-rs-demo-g9qcx   1/1     Running   0          3m29s   <none>
hello-rs-demo-plhst   1/1     Running   0          45s     app=hello-demo
hello-rs-demo-qnzg5   1/1     Running   0          3m29s   app=hello-demo
hello-rs-demo-wxfhz   1/1     Running   0          2m39s   app=hello-demo
➜  replicaset
```

Это иногда бывает нужно для того, чтобы увезти зависшую поду на дебаг.

```
Every 2,0s: kubectl get all       MacBook-Pro-2.local: Sun Apr  5 23:45:03
2020

NAME                         READY   STATUS        RESTARTS   AGE
pod/hello-rs-demo-g9qcx      1/1     Terminating   0          3m59s
pod/hello-rs-demo-plhst      1/1     Running       0          75s
pod/hello-rs-demo-qnzg5      1/1     Running       0          3m59s
pod/hello-rs-demo-wxfhz      1/1     Running       0          3m9s

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/hello-rs-demo       3         3         3       3m59s
```

Если мы удаляем ресурс репликасет, то автоматически удаляются и все поды, которые она контролирует.

```
➜  replicaset kubectl delete rs hello-rs-demo
replicaset.apps "hello-rs-demo" deleted
➜  replicaset
```

```
Every 2,0s: kubectl get all       MacBook-Pro-2.local: Sun Apr  5 23:46:47
2020

NAME                         READY   STATUS        RESTARTS   AGE
pod/hello-rs-demo-plhst      1/1     Terminating   0          2m59s
pod/hello-rs-demo-qnzg5      1/1     Terminating   0          5m43s
pod/hello-rs-demo-wxfhz      1/1     Terminating   0          4m53s
```

```
➜  replicaset kubectl get all
No resources found in myapp namespace.
➜  replicaset
```

DEPLOYMENT

https://kubernetes.io/docs/concepts/workloads/controllers/deployment/ - вот здесь есть подробная документация по деплойментам.

Но replicaset-ом напрямую обычно мало кто пользуется. Потому что обновление версии image (обновление сервиса) становится тяжелым. Для того, чтобы было удобно накатывать новую версию и в случае чего откатывать существует такой контролер как Deployment, именно его настройкой в 99% и занимается разработчик.

```
➜  deployment ls
deployment.yaml hello-py
```

Давайте посмотрим, как выглядит описание ресурса Deployment

```
➜  deployment cat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
    name: hello-deployment
spec:
    replicas: 2
    selector:
       matchLabels:
          app: hello-py-app
    template:
       metadata:
          labels:
             app: hello-py-app
       spec:
          containers:
          - name: hello-py
            image: hello-py:v1
            ports:
              - name: web
                containerPort: 80
➜  deployment
```

Настройки очень похожи на репликасет.

После применения деплоймента

```
➜  deployment kubectl apply -f deployment.yaml
deployment.apps/hello-deployment created
➜  deployment
```

Можно увидеть, что деплоймент на самом деле создает репликасет, который уже создает поды.

```
➜  deployment kubectl get all
NAME                                    READY   STATUS    RESTARTS   AGE
pod/hello-deployment-58fcff8c8f-6gm2k   1/1     Running   0          23s
pod/hello-deployment-58fcff8c8f-dw5nl   1/1     Running   0          23s

NAME                               READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hello-deployment   2/2     2            2           23s

NAME                                          DESIRED   CURRENT   READY
AGE
replicaset.apps/hello-deployment-58fcff8c8f   2         2         2
23s
➜  deployment
```

Меняем код (убрал один пробел) и собираем новый image.

```
➜  hello-py git:(master) ✗ vim app.py
➜  hello-py git:(master) ✗ docker build -t hello-py:v2 .
Sending build context to Docker daemon  64.51kB
Step 1/3 : FROM python:3.5-onbuild
# Executing 3 build triggers
 ---> Using cache
 ---> Using cache
 ---> c0cc27dacef1
Step 2/3 : EXPOSE 8000
 ---> Running in cda21630e864
Removing intermediate container cda21630e864
 ---> 1bfe9ceba5f5
```

```
Step 3/3 : ENTRYPOINT ["python", "/usr/src/app/app.py"]
 ---> Running in 4eb88becdc19
Removing intermediate container 4eb88becdc19
 ---> c2878cffbb0b
Successfully built c2878cffbb0b
Successfully tagged hello-py:v2
➜  hello-py git:(master) ✗
```

Давайте обновим image в деплойменте

```
➜  deployment kubectl set image deployment/hello-deployment
hello-py=hello-py:v2 --record
deployment.apps/hello-deployment image updated
```

И что мы видим

```
➜  deployment kubectl get all
NAME                                    READY   STATUS        RESTARTS
AGE
pod/hello-deployment-58fcff8c8f-6gm2k   0/1     Terminating   0
9m5s
pod/hello-deployment-58fcff8c8f-dw5nl   0/1     Terminating   0
9m5s
pod/hello-deployment-8548bb8cc8-k8d9b   1/1     Running       0
37s
pod/hello-deployment-8548bb8cc8-tnnt9   1/1     Running       0
36s

NAME                               READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hello-deployment   2/2     2            2           9m5s

NAME                                          DESIRED   CURRENT   READY
AGE
replicaset.apps/hello-deployment-58fcff8c8f   0         0         0
9m5s
replicaset.apps/hello-deployment-8548bb8cc8   2         2         2
37s
➜  deployment
```

Видим, что деплоймент создал еще одну репликасет, а старую заскейлил до `replicas 0`.

Теперь можно посмотреть историю по `deployment/`

```
➜  deployment kubectl rollout history deployment/hello-deployment
deployment.apps/hello-deployment
REVISION  CHANGE-CAUSE
1         <none>
2         kubectl set image deployment/hello-deployment
hello-py=hello-py:v2 --record=true

➜  deployment
```

Если мы откатим деплоймент

```
➜  deployment kubectl rollout undo deployment hello-deployment
deployment.apps/hello-deployment rolled back
```

то можно видеть, как старая репликасет наоборот заскейлилась до 2, а новая - до 0.

```
➜  deployment kubectl get all
NAME                                     READY   STATUS        RESTARTS
AGE
pod/hello-deployment-58fcff8c8f-jrmbq    1/1     Running       0
3s
pod/hello-deployment-58fcff8c8f-rgz4r    1/1     Running       0
5s
pod/hello-deployment-8548bb8cc8-k8d9b    1/1     Terminating   0
3m59s
pod/hello-deployment-8548bb8cc8-tnnt9    1/1     Terminating   0
3m58s

NAME                               READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hello-deployment   2/2     2            2           12m

NAME                                          DESIRED   CURRENT   READY
AGE
replicaset.apps/hello-deployment-58fcff8c8f   2         2         2
12m
```

```
replicaset.apps/hello-deployment-8548bb8cc8   0         0         0
3m59s
➜  deployment
```

```
➜  deployment kubectl get all
NAME                                     READY   STATUS    RESTARTS   AGE
pod/hello-deployment-58fcff8c8f-jrmbq    1/1     Running   0          52s
pod/hello-deployment-58fcff8c8f-rgz4r    1/1     Running   0          54s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hello-deployment    2/2     2            2           13m

NAME                                           DESIRED   CURRENT   READY
AGE
replicaset.apps/hello-deployment-58fcff8c8f    2         2         2
13m
replicaset.apps/hello-deployment-8548bb8cc8    0         0         0
4m48s
➜  deployment
```

В случае необходимости можно заскейлить деплоймент.

```
➜  deployment kubectl scale deployment hello-deployment --replicas=4
deployment.apps/hello-deployment scaled
```

```
➜  deployment kubectl get all
NAME                                     READY   STATUS    RESTARTS   AGE
pod/hello-deployment-58fcff8c8f-jrmbq    1/1     Running   0          115s
pod/hello-deployment-58fcff8c8f-nptqw    1/1     Running   0          3s
pod/hello-deployment-58fcff8c8f-nrkq4    1/1     Running   0          3s
pod/hello-deployment-58fcff8c8f-rgz4r    1/1     Running   0          117s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hello-deployment    4/4     4            4           14m

NAME                                           DESIRED   CURRENT   READY
AGE
replicaset.apps/hello-deployment-58fcff8c8f    4         4         4
14m
```

```
replicaset.apps/hello-deployment-8548bb8cc8   0         0         0
5m51s
➜  deployment
```

Или задаунскейлить.

```
➜  deployment kubectl scale deployment hello-deployment --replicas=2
deployment.apps/hello-deployment scaled
```

```
➜  deployment kubectl get all
NAME                                     READY   STATUS        RESTARTS
AGE
pod/hello-deployment-58fcff8c8f-jrmbq    1/1     Running       0
2m28s
pod/hello-deployment-58fcff8c8f-nptqw    1/1     Terminating   0
36s
pod/hello-deployment-58fcff8c8f-nrkq4    1/1     Terminating   0
36s
pod/hello-deployment-58fcff8c8f-rgz4r    1/1     Running       0
2m30s

NAME                               READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hello-deployment   2/2     2            2           14m

NAME                                          DESIRED   CURRENT   READY
AGE
replicaset.apps/hello-deployment-58fcff8c8f   2         2         2
14m
replicaset.apps/hello-deployment-8548bb8cc8   0         0         0
6m24s
➜  deployment
```

SERVICE

Деплоймент - это контроллер, который поднимает несколько подов, несколько “процессов”, но как в эти сервисы ходить? Как балансировать нагрузку?

Для того, чтобы другие сервисы имели к группе подов доступ как целому сервису, а не разрозненной группе инстансов, в кубернетес есть такая сущность (ресурс), как сервис.

```
➜  service ls
deployment.yaml hello-py        service.yaml
```

```
➜  service cat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
    name: hello-deployment
spec:
    replicas: 2
    selector:
       matchLabels:
          app: hello-py-app
    template:
       metadata:
          labels:
             app: hello-py-app
       spec:
          containers:
          - name: hello-py
            image: hello-py:v1
            ports:
              - name: web
                containerPort: 80
```

```
➜  service cat service.yaml
apiVersion: v1
kind: Service
metadata:
  name: hello-service
spec:
  selector:
    app: hello-py-app
  ports:
```

```
    - protocol: TCP
      port: 9000
      targetPort: web
  type: NodePort
➜  service
```

У сервиса есть имя. И именно это имя будет использоваться для обращения к этому сервису, внутри кубернетеса. Селектор лейблов определяет набор подов, к которым сервис будет перенаправлять запросы. При это необязательно это будут поды одного приложения или одного деплоймента. Поды могут существовать в рамках разных деплойментов, но если их выбирает селектор, мы все-равно воспринимаем, как один селектор. Такая механика например используется для канареечных деплоев, когда в проде выкачено две версии одного приложения.

Порты определяют, как будут пробрасываться. `targetPort` - это порт в контейнере. `port` - это что выставляет наружу сервис.

Применим манифесты.

```
➜  service kubectl apply -f deployment.yaml -f service.yaml
deployment.apps/hello-deployment unchanged
service/hello-service created
```

```
➜  service kubectl get all
NAME                                 READY   STATUS    RESTARTS   AGE
pod/hello-deployment-58fcff8c8f-jrmbq   1/1     Running   0          11m
pod/hello-deployment-58fcff8c8f-rgz4r   1/1     Running   0          11m

NAME                    TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
service/hello-service   NodePort   10.102.143.105   <none>
9000:30517/TCP   24s

NAME                               READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hello-deployment   2/2     2            2           23m

NAME                                          DESIRED   CURRENT   READY
AGE
то то replicaset.apps/hello-deployment-58fcff8c8f   2         2         2
23m
```

```
replicaset.apps/hello-deployment-8548bb8cc8   0         0         0
15m
➜  service
```

type NodePort - это значит, что кубернетес один из портов на ноде забиндит за этим сервисом, и обращаясь по этому порту, вы будете попадать в указанный сервис. Крайне редко используется в продакшне, и очень часто используется для дебага.

В миникубе есть команда, которая позволяет получить адрес сервиса по имени сервиса и неймспесу в котором он находится

```
➜  service minikube service hello-service --url  -n myapp
http://192.168.176.128:30517
➜  service
```

Если сделаем курл на адрес health, получим ответ.

```
➜  service curl http://192.168.176.128:30517/health
{"status": "ok"}%
➜  service
```

```
➜  service curl http://192.168.176.128:30517/
Hello world from hello-deployment-58fcff8c8f-jrmbq !%
➜  service
```

Еще можно за порт-форвардить сервис на локальную машину с помощью kubectl.

```
➜  service kubectl port-forward svc/hello-service 10000:9000
Forwarding from 127.0.0.1:10000 -> 80
Forwarding from [::1]:10000 -> 80
Handling connection for 10000
Handling connection for 10000
Handling connection for 10000
^C%
➜  service
```

Запросы с локальной машины на порт 10000 будут отправлять в указанный сервис на порт 9000 (многие с этим путаются)

```
➜  ~ curl http://localhost:10000/
Hello world from hello-deployment-58fcff8c8f-rgz4r !%
```

```
➜  ~ curl http://localhost:10000/health
{"status": "ok"}%
➜  ~
```

Service discovery

Чтобы другие сервисы внутри кубика могли обращаться к сервису используется 2 механизма.

1. С помощью переменых окружения.
   Запустим throw away под из образа busybox и посмотрим, какие там есть env переменные:

```
➜  service kubectl run -it --rm busybox --image=busybox
If you don't see a command prompt, try pressing enter.
/ # env | grep HELLO
HELLO_SERVICE_PORT_9000_TCP_ADDR=10.102.143.105
HELLO_SERVICE_PORT_9000_TCP_PORT=9000
HELLO_SERVICE_PORT_9000_TCP_PROTO=tcp
HELLO_SERVICE_SERVICE_HOST=10.102.143.105
HELLO_SERVICE_PORT_9000_TCP=tcp://10.102.143.105:9000
HELLO_SERVICE_SERVICE_PORT=9000
HELLO_SERVICE_PORT=tcp://10.102.143.105:9000
/ # wget -qO- http://hello-service:9000/
Hello world from hello-deployment-58fcff8c8f-jrmbq !/ #
```

2. Мы можем обращаться по доменному имени сервиса. Для сервисов внутри неймспеса это просто имя сервиса. Для сервисов в других неймспейсах serivce.ns. Но можно использовать fqdn внутри кластера.

Как видим, отвечают разные поды. Таким образом балансинг на стороне service осуществляется. Есть механизм реализации через iptables, либо ipvs.

```
/ # wget -qO- http://hello-service:9000/
Hello world from hello-deployment-58fcff8c8f-rgz4r !/ #
/ # wget -qO- http://hello-service:9000/
Hello world from hello-deployment-58fcff8c8f-jrmbq !/ #
/ # wget -qO- http://hello-service:9000/
Hello world from hello-deployment-58fcff8c8f-jrmbq !/ #
/ # wget -qO- http://hello-service:9000/
Hello world from hello-deployment-58fcff8c8f-jrmbq !/ #
/ # wget -qO- http://hello-service:9000/
Hello world from hello-deployment-58fcff8c8f-rgz4r !/ #
/ #
```

SKAFFOLD

https://skaffold.dev/ - вот здесь есть документация.

Для удобной разработки, чтобы каждый раз не менять версии image и не собирать докеры с помощью команд, существует утилита skaffold.

```
➜  skaffold ls
deployment.yaml hello-py        service.yaml
```

В файле, который создает skaffold указано, как собирать артефакты, и как их деплоить

```
➜  skaffold skaffold init
apiVersion: skaffold/v2alpha4
kind: Config
metadata:
  name: skaffold
build:
  artifacts:
  - image: hello-py
    context: hello-py
deploy:
```

```
  kubectl:
    manifests:
    - deployment.yaml
    - service.yaml

Do you want to write this configuration to skaffold.yaml? [y/n]: y
Configuration skaffold.yaml was written
You can now run [skaffold build] to build the artifacts
or [skaffold run] to build and deploy
or [skaffold dev] to enter development mode, with auto-redeploy
There is a new version (1.7.0) of Skaffold available. Download it at
https://storage.googleapis.com/skaffold/releases/latest/skaffold-darwin-am
d64
```

то автоматически собирается образ, тэгируется и этот тэг подменяется в `deployment` манифестах

Также есть возможность пушить в собранный `image` в удаленный `registry`.

Отредактируем.

```
➜  skaffold vim hello-py/app.py
➜  skaffold ccat hello-py/app.py
import os
import json

from flask import Flask
app = Flask(__name__)

@app.route("/health")
def health():
    return '{"status": "ok"}'

@app.route("/version")
def health():
    return '{"version": "0.2"}'

@app.route("/")
def hello():
    return 'Hello world from ' + os.environ['HOSTNAME'] + '!'
```

```
if __name__ == "__main__":
    app.run(host='0.0.0.0',port='80')
➜  skaffold
```

Когда запускаем run

```
➜  skaffold skaffold run
Generating tags...
 - hello-py -> hello-py:fe90391-dirty
Checking cache...
 - hello-py: Not found. Building
Found [minikube] context, using local docker daemon.
Building [hello-py]...
Sending build context to Docker daemon  53.25kB
Step 1/3 : FROM python:3.5-onbuild
# Executing 3 build triggers
 ---> Using cache
 ---> Running in d668d596401d
Collecting Flask==1.1.2 (from -r requirements.txt (line 1))
  Downloading
https://files.pythonhosted.org/packages/f2/28/2a03252dfb9ebf377f40fba6a784
1b47083260bf8bd8e737b0c6952df83f/Flask-1.1.2-py2.py3-none-any.whl (94kB)
Collecting itsdangerous>=0.24 (from Flask==1.1.2->-r requirements.txt
(line 1))
  Downloading
https://files.pythonhosted.org/packages/76/ae/44b03b253d6fade317f32c24d100
b3b35c2239807046a4c953c7b89fa49e/itsdangerous-1.1.0-py2.py3-none-any.whl
Collecting Werkzeug>=0.15 (from Flask==1.1.2->-r requirements.txt (line
1))
  Downloading
https://files.pythonhosted.org/packages/cc/94/5f7079a0e00bd6863ef8f1da6387
21e9da21e5bacee597595b318f71d62e/Werkzeug-1.0.1-py2.py3-none-any.whl
(298kB)
Collecting click>=5.1 (from Flask==1.1.2->-r requirements.txt (line 1))
  Downloading
https://files.pythonhosted.org/packages/dd/c0/4d8f43a9b16e289f36478422031b
8a63b54b6ac3b1ba605d602f10dd54d6/click-7.1.1-py2.py3-none-any.whl (82kB)
Collecting Jinja2>=2.10.1 (from Flask==1.1.2->-r requirements.txt (line
1))
```

```
  Downloading
https://files.pythonhosted.org/packages/27/24/4f35961e5c669e96f6559760042a
55b9bcfcdb82b9bdb3c8753dbe042e35/Jinja2-2.11.1-py2.py3-none-any.whl
(126kB)
Collecting MarkupSafe>=0.23 (from Jinja2>=2.10.1->Flask==1.1.2->-r
requirements.txt (line 1))
  Downloading
https://files.pythonhosted.org/packages/6e/57/d40124076756c19ff2269678de7a
e25a14ebbb3f6314eb5ce9477f191350/MarkupSafe-1.1.1-cp35-cp35m-manylinux1_x8
6_64.whl
Installing collected packages: itsdangerous, Werkzeug, click, MarkupSafe,
Jinja2, Flask
Successfully installed Flask-1.1.2 Jinja2-2.11.1 MarkupSafe-1.1.1
Werkzeug-1.0.1 click-7.1.1 itsdangerous-1.1.0
 ---> 683359c3862f
Step 2/3 : EXPOSE 8000
 ---> Running in 92af33dd7b49
 ---> 1de81bd3a8cd
Step 3/3 : ENTRYPOINT ["python", "/usr/src/app/app.py"]
 ---> Running in 1a3aaadf9373
 ---> 97eec43e1002
Successfully built 97eec43e1002
Successfully tagged hello-py:fe90391-dirty
Tags used in deployment:
 - hello-py ->
hello-py:97eec43e100220cd281c3a2306fd77632707d96813a54553cf9e2ae89ef8e88b
   local images can't be referenced by digest. They are tagged and
referenced by a unique ID instead
Starting deploy...
 - deployment.apps/hello-deployment configured
 - service/hello-service configured
You can also run [skaffold run --tail] to get the logs
There is a new version (1.7.0) of Skaffold available. Download it at
https://storage.googleapis.com/skaffold/releases/latest/skaffold-darwin-am
d64
```

```
➜  skaffold kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
hello-deployment-7c4f579c85-bmx8n   0/1     Error     2          24s
```

```
hello-deployment-d98964d7-fn2d7     1/1     Running   0          3m16s
hello-deployment-d98964d7-lr772     1/1     Running   0          3m14s
➜  skaffold
```

Посмотрим логи пода. Мы допустили ошибку, когда отредактировали.

```
➜  skaffold kubectl logs hello-deployment-7c4f579c85-bmx8n
Traceback (most recent call last):
  File "/usr/src/app/app.py", line 11, in <module>
    @app.route("/version")
  File "/usr/local/lib/python3.5/site-packages/flask/app.py", line 1315,
in decorator
    self.add_url_rule(rule, endpoint, f, **options)
  File "/usr/local/lib/python3.5/site-packages/flask/app.py", line 98, in
wrapper_func
    return f(self, *args, **kwargs)
  File "/usr/local/lib/python3.5/site-packages/flask/app.py", line 1284,
in add_url_rule
    "existing endpoint function: %s" % endpoint
AssertionError: View function mapping is overwriting an existing endpoint
function: health
```

Исправляем ошибку

```
➜  skaffold vim hello-py/app.py
➜  skaffold ccat hello-py/app.py
import os
import json

from flask import Flask
app = Flask(__name__)

@app.route("/health")
def health():
    return '{"status": "ok"}'

@app.route("/version")
def version():
    return '{"version": "0.2"}'
```

```python
@app.route("/")
def hello():
    return 'Hello world from ' + os.environ['HOSTNAME'] + '!'

if __name__ == "__main__":
    app.run(host='0.0.0.0',port='80')
```

Собираем и деплоим.

```
➜  skaffold skaffold run
Generating tags...
 - hello-py -> hello-py:fe90391-dirty
Checking cache...
 - hello-py: Not found. Building
Found [minikube] context, using local docker daemon.
Building [hello-py]...
Sending build context to Docker daemon  53.25kB
Step 1/3 : FROM python:3.5-onbuild
# Executing 3 build triggers
 ---> Using cache
 ---> Using cache
 ---> 892fb5e18624
Step 2/3 : EXPOSE 8000
 ---> Running in a25ac5dded0f
 ---> 56b8afee1acc
Step 3/3 : ENTRYPOINT ["python", "/usr/src/app/app.py"]
 ---> Running in 6642d6c9f063
 ---> cfb65f0dce84
Successfully built cfb65f0dce84
Successfully tagged hello-py:fe90391-dirty
Tags used in deployment:
 - hello-py ->
hello-py:cfb65f0dce847a2aaa67618d4307d280786ce961d5609ff5948138731e9f1cac
   local images can't be referenced by digest. They are tagged and
referenced by a unique ID instead
Starting deploy...
 - deployment.apps/hello-deployment configured
 - service/hello-service configured
You can also run [skaffold run --tail] to get the logs
```

```
There is a new version (1.7.0) of Skaffold available. Download it at
https://storage.googleapis.com/skaffold/releases/latest/skaffold-darwin-am
d64
```

По урлу видим, что у нас новая версия. А руками в манифесты мы ничего не меняли.

```
➜  skaffold curl http://192.168.176.128:30517/version
{"version": "0.2"}%
➜  skaffold
```

Теперь можно и удалить.

```
➜  skaffold skaffold delete
Cleaning up...
 - deployment.apps "hello-deployment" deleted
 - service "hello-service" deleted
There is a new version (1.7.0) of Skaffold available. Download it at
https://storage.googleapis.com/skaffold/releases/latest/skaffold-darwin-am
d64

➜  skaffold
```

HEALTH CHECK

И так снова запустим skaffold run

```
➜  probes ls
deployment.yaml hello-py        service.yaml    skaffold.yaml
```

```
➜  probes skaffold run
Generating tags...
 - hello-py -> hello-py:fe90391-dirty
Checking cache...
 - hello-py: Not found. Building
```

```
Found [minikube] context, using local docker daemon.
Building [hello-py]...
Sending build context to Docker daemon  53.25kB
Step 1/3 : FROM python:3.5-onbuild
# Executing 3 build triggers
 ---> Using cache
 ---> Using cache
 ---> 1b78aa5af8e5
Step 2/3 : EXPOSE 8000
 ---> Running in 826bdf9ce2b6
 ---> ff3521e07efb
Step 3/3 : ENTRYPOINT ["python", "/usr/src/app/app.py"]
 ---> Running in 1a00575da7d8
 ---> c402291825b1
Successfully built c402291825b1
Successfully tagged hello-py:fe90391-dirty
Tags used in deployment:
 - hello-py ->
hello-py:c402291825b10474e3457cf255b35957bece58af312ef27721a7f74182623656
   local images can't be referenced by digest. They are tagged and
referenced by a unique ID instead
Starting deploy...
 - deployment.apps/hello-deployment configured
 - service/hello-service configured
You can also run [skaffold run --tail] to get the logs
There is a new version (1.7.0) of Skaffold available. Download it at
https://storage.googleapis.com/skaffold/releases/latest/skaffold-darwin-am
d64
```

Все ресурсы снова на месте:

```
➜  probes kubectl get all
NAME                                     READY   STATUS    RESTARTS   AGE
pod/hello-deployment-5758fd9786-vntrc    1/1     Running   0          15s
pod/hello-deployment-5758fd9786-x58gv    1/1     Running   0          17s

NAME                    TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
service/hello-service   NodePort   10.102.207.240   <none>
9000:30049/TCP   61s
```

```
NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hello-deployment      2/2     2            2           61s

NAME                                            DESIRED   CURRENT   READY
AGE
replicaset.apps/hello-deployment-5758fd9786     2         2         2
17s
replicaset.apps/hello-deployment-58bb558495     0         0         0
61s
➜  probes
```

Добавим задержку при старте приложения и посмотрим, как будет отвечать сервис, в случае, когда мы будем передеплоивать приложение

```
➜  probes vim hello-py/app.py
```

```
➜  probes ccat hello-py/app.py
import os
import json
import time

from flask import Flask
app = Flask(__name__)

@app.route("/health")
def health():
    return '{"status": "ok"}'

@app.route("/version")
def version():
    return '{"version": "0.2"}'

@app.route("/")
def hello():
    return 'Hello world from ' + os.environ['HOSTNAME'] + '!'

if __name__ == "__main__":
    time.sleep(7) # emulating java app
    app.run(host='0.0.0.0',port='80')
```

```
➜  probes
```

Получим сервис.

```
➜  probes minikube service hello-service --url -n myapp
http://192.168.176.128:30049
➜  probes
```

Запускаем цикл проверки:

```
➜  ~ while true ; do curl http://192.168.176.128:30049/health ; echo
'\n'; sleep 1; done
{"status": "ok"}

{"status": "ok"}
```

Обновим деплоймент

```
➜  probes skaffold run --tail
Generating tags...
 - hello-py -> hello-py:fe90391-dirty
Checking cache...
 - hello-py: Not found. Building
Found [minikube] context, using local docker daemon.
Building [hello-py]...
Sending build context to Docker daemon  53.25kB
Step 1/3 : FROM python:3.5-onbuild
# Executing 3 build triggers
 ---> Using cache
 ---> Using cache
 ---> 9376e51f76a2
Step 2/3 : EXPOSE 8000
 ---> Running in 3295693e4867
 ---> 8d70c97e8363
Step 3/3 : ENTRYPOINT ["python", "/usr/src/app/app.py"]
 ---> Running in 4d48f5ea045c
 ---> 8643179cedfc
Successfully built 8643179cedfc
Successfully tagged hello-py:fe90391-dirty
```

```
Tags used in deployment:
 - hello-py ->
hello-py:8643179cedfc09a3bd7b083d615382ce9251312e6319bd79d873eb15dff9b271
   local images can't be referenced by digest. They are tagged and
referenced by a unique ID instead
Starting deploy...
 - deployment.apps/hello-deployment configured
 - service/hello-service configured
[hello-deployment-7f8ddcfb-cvhd2 hello-py]  * Serving Flask app "app"
(lazy loading)
[hello-deployment-7f8ddcfb-cvhd2 hello-py]  * Environment: production
[hello-deployment-7f8ddcfb-cvhd2 hello-py]    WARNING: This is a
development server. Do not use it in a production deployment.
[hello-deployment-7f8ddcfb-cvhd2 hello-py]    Use a production WSGI server
instead.
[hello-deployment-7f8ddcfb-cvhd2 hello-py]  * Debug mode: off
[hello-deployment-7f8ddcfb-cvhd2 hello-py]  * Running on
http://0.0.0.0:80/ (Press CTRL+C to quit)
[hello-deployment-7f8ddcfb-zjb5x hello-py]  * Serving Flask app "app"
(lazy loading)
[hello-deployment-7f8ddcfb-zjb5x hello-py]  * Environment: production
[hello-deployment-7f8ddcfb-zjb5x hello-py]    WARNING: This is a
development server. Do not use it in a production deployment.
[hello-deployment-7f8ddcfb-zjb5x hello-py]    Use a production WSGI server
instead.
[hello-deployment-7f8ddcfb-zjb5x hello-py]  * Debug mode: off
[hello-deployment-7f8ddcfb-zjb5x hello-py]  * Running on
http://0.0.0.0:80/ (Press CTRL+C to quit)
[hello-deployment-7f8ddcfb-cvhd2 hello-py] 172.17.0.1 - - [06/Apr/2020
10:54:54] "GET /health HTTP/1.1" 200 -
[hello-deployment-7f8ddcfb-zjb5x hello-py] 172.17.0.1 - - [06/Apr/2020
10:54:55] "GET /health HTTP/1.1" 200 -
[hello-deployment-7f8ddcfb-cvhd2 hello-py] 172.17.0.1 - - [06/Apr/2020
10:54:56] "GET /health HTTP/1.1" 200 -
[hello-deployment-7f8ddcfb-cvhd2 hello-py] 172.17.0.1 - - [06/Apr/2020
10:54:57] "GET /health HTTP/1.1" 200 -
^CThere is a new version (1.7.0) of Skaffold available. Download it at
https://storage.googleapis.com/skaffold/releases/latest/skaffold-darwin-am
d64

➜  probes
```

Видим, что трафик отправляет на поды, в которых контейнер уже запущен, а приложение еще не готово принимать запросы

```
➜  ~ while true ; do curl http://192.168.176.128:30049/health ; echo
'\n'; sleep 1; done
{"status": "ok"}
{"status": "ok"}
{"status": "ok"}
{"status": "ok"}
{"status": "ok"}
{"status": "ok"}
{"status": "ok"}
{"status": "ok"}

curl: (7) Failed to connect to 192.168.176.128 port 30049: Connection
refused

curl: (7) Failed to connect to 192.168.176.128 port 30049: Connection
refused

curl: (7) Failed to connect to 192.168.176.128 port 30049: Connection
refused

curl: (7) Failed to connect to 192.168.176.128 port 30049: Connection
refused

curl: (7) Failed to connect to 192.168.176.128 port 30049: Connection
refused

curl: (7) Failed to connect to 192.168.176.128 port 30049: Connection
refused

curl: (7) Failed to connect to 192.168.176.128 port 30049: Connection
refused

{"status": "ok"}
{"status": "ok"}
{"status": "ok"}
```

Таким образом при редеплое мы получаем недоступность. Чтобы ее избежать в кубернетесе есть пробы

https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/

(**https://kubernetes.io/ru/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/**)

```
➜  probes vim deployment.yaml
➜  probes ccat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
    name: hello-deployment
spec:
    replicas: 2
    selector:
       matchLabels:
          app: hello-py-app
    template:
       metadata:
          labels:
             app: hello-py-app
       spec:
          containers:
          - name: hello-py
            image: hello-py:v1
            ports:
              - name: web
                containerPort: 80
            livenessProbe:
              httpGet:
                port: 80
                path: /
              initialDelaySeconds: 10
              periodSeconds: 5
              timeoutSeconds: 2
            readinessProbe:
              httpGet:
```

```
              port: 80
              path: /health
            initialDelaySeconds: 10
            periodSeconds: 5
```

livenessProbe - это ручка, которая говорит, что под жив и ее опрашивает агент kubelet. В случае если проба провалится, kubelet отрестартит pod. Это нужно для ситуации, когда приложение перестает работать из-за внутренней ошибки (например, блокировки)

readinessProbe - это ручка, которая подтверждает, что под готов для того, чтобы принимать трафик.

```
➜  probes
```

Передеплоим приложение

```
➜  probes skaffold run
Generating tags...
 - hello-py -> hello-py:fe90391-dirty
Checking cache...
 - hello-py: Found Locally
Tags used in deployment:
 - hello-py ->
hello-py:8643179cedfc09a3bd7b083d615382ce9251312e6319bd79d873eb15dff9b271
   local images can't be referenced by digest. They are tagged and
referenced by a unique ID instead
Starting deploy...
 - deployment.apps/hello-deployment configured
 - service/hello-service configured
You can also run [skaffold run --tail] to get the logs
There is a new version (1.7.0) of Skaffold available. Download it at
https://storage.googleapis.com/skaffold/releases/latest/skaffold-darwin-am
d64

➜  probes
```

Обновим код и посмотрим, как будут отвечать:

```
➜  probes vim hello-py/app.py
➜  probes ccat hello-py/app.py
import os
import json
import time

from flask import Flask
app = Flask(__name__)

@app.route("/health")
def health():
    return '{"status": "ok"}'

@app.route("/version")
def version():
    return '{"version": "0.3"}'

@app.route("/")
def hello():
    return 'Hello world from ' + os.environ['HOSTNAME'] + '!'

if __name__ == "__main__":
    time.sleep(7) # emulating java app
    app.run(host='0.0.0.0',port='80')
➜  probes
```

```
➜  probes skaffold run
Generating tags...
 - hello-py -> hello-py:fe90391-dirty
Checking cache...
 - hello-py: Not found. Building
Found [minikube] context, using local docker daemon.
Building [hello-py]...
Sending build context to Docker daemon  53.25kB
Step 1/3 : FROM python:3.5-onbuild
# Executing 3 build triggers
 ---> Using cache
 ---> Using cache
 ---> 3d91586f9243
```

```
Step 2/3 : EXPOSE 8000
 ---> Running in 3e612e6534ad
 ---> 0a860230ca93
Step 3/3 : ENTRYPOINT ["python", "/usr/src/app/app.py"]
 ---> Running in 329016cda61b
 ---> 1488ceb9a3cb
Successfully built 1488ceb9a3cb
Successfully tagged hello-py:fe90391-dirty
Tags used in deployment:
 - hello-py ->
hello-py:1488ceb9a3cbabc3a189406c2dbc2408d0c27054f862fe2fe50f7f1918181b8a
   local images can't be referenced by digest. They are tagged and
referenced by a unique ID instead
Starting deploy...
 - deployment.apps/hello-deployment configured
 - service/hello-service configured
You can also run [skaffold run --tail] to get the logs
There is a new version (1.7.0) of Skaffold available. Download it at
https://storage.googleapis.com/skaffold/releases/latest/skaffold-darwin-am
d64

➜  probes
```

Видим, что в момент деплоя ошибки прошли. Но в один момент времени (во время раскатки) доступны сразу две версии приложения (старая и новая).

```
^C%
➜  ~ while true ; do curl http://192.168.176.128:30049/version ; echo
'\n'; sleep 1; done
{"version": "0.2"}
{"version": "0.2"}
{"version": "0.2"}
{"version": "0.2"}
{"version": "0.2"}
{"version": "0.2"}
{"version": "0.3"}
{"version": "0.2"}
{"version": "0.3"}
{"version": "0.2"}
```

```
{"version": "0.2"}
{"version": "0.3"}
{"version": "0.2"}
{"version": "0.2"}
{"version": "0.3"}
{"version": "0.2"}
{"version": "0.3"}
{"version": "0.3"}
{"version": "0.3"}
```

BLUEGREEN

Если мы хотим избежать такого, то мы можем использовать следующую технику.

Удаляем все и создаем простое приложение.

```
➜  bluegreen kubectl delete all --all
pod "hello-deployment-59545c599d-hwrz6" deleted
pod "hello-deployment-59545c599d-v9q98" deleted
service "hello-service" deleted
deployment.apps "hello-deployment" deleted
replicaset.apps "hello-deployment-5758fd9786" deleted
replicaset.apps "hello-deployment-58bb558495" deleted
replicaset.apps "hello-deployment-59545c599d" deleted
replicaset.apps "hello-deployment-6cb84578d9" deleted
replicaset.apps "hello-deployment-7f8ddcfb" deleted
➜  bluegreen
```

```
➜  bluegreen kubectl get all
No resources found in myapp namespace.
➜  bluegreen
```

```
➜  bluegreen ccat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
    name: hello-deployment
spec:
    replicas: 2
    selector:
       matchLabels:
          app: hello-py-app
    template:
       metadata:
          labels:
             app: hello-py-app
       spec:
          containers:
          - name: hello-py
            image: hello-py:v1
            ports:
              - name: web
                containerPort: 80
```

```
➜  bluegreen ccat service.yaml
apiVersion: v1
kind: Service
metadata:
  name: hello-service
spec:
  selector:
    app: hello-py-app
  ports:
    - protocol: TCP
      port: 9000
      targetPort: web
  type: NodePort
```

```
➜  bluegreen ccat hello-py/app.py
import os
import json

from flask import Flask
app = Flask(__name__)

@app.route("/health")
```

```
def health():
    return '{"status": "ok"}'

@app.route("/version")
def version():
    return '{"version": "1"}'

@app.route("/")
def hello():
    return 'Hello world from ' + os.environ['HOSTNAME'] + '!'

if __name__ == "__main__":
    app.run(host='0.0.0.0',port='80')
➜  bluegreen
```

Собираем образ

```
➜  bluegreen cd hello-py
➜  hello-py git:(master) ✗ docker build -t hello-py:v1 .
Sending build context to Docker daemon  64.51kB
Step 1/3 : FROM python:3.5-onbuild
# Executing 3 build triggers
 ---> Using cache
 ---> Using cache
 ---> c0afe4b47121
Step 2/3 : EXPOSE 8000
 ---> Running in e45199437a3c
Removing intermediate container e45199437a3c
 ---> cd3f18b4a63e
Step 3/3 : ENTRYPOINT ["python", "/usr/src/app/app.py"]
 ---> Running in a4eba4c891db
Removing intermediate container a4eba4c891db
 ---> 8fac278d0496
Successfully built 8fac278d0496
Successfully tagged hello-py:v1
➜  hello-py git:(master) ✗
```

Создаем деплоймент и сервис.

```
➜  hello-py git:(master) ✗ cd ..
```

```
➜  bluegreen kubectl apply -f service.yaml -f deployment.yaml
service/hello-service created
deployment.apps/hello-deployment created
➜  bluegreen
```

```
➜  bluegreen kubectl get all
NAME                                     READY   STATUS    RESTARTS   AGE
pod/hello-deployment-58fcff8c8f-9f82q    1/1     Running   0          25s
pod/hello-deployment-58fcff8c8f-g6h86    1/1     Running   0          25s

NAME                    TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)
AGE
service/hello-service   NodePort   10.108.32.92   <none>
9000:31318/TCP   25s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hello-deployment    2/2     2            2           25s

NAME                                           DESIRED   CURRENT   READY
AGE
replicaset.apps/hello-deployment-58fcff8c8f    2         2         2
25s
➜  bluegreen
```

Проверяем, что все ок.

```
➜  bluegreen minikube service hello-service --url -n myapp
http://192.168.176.128:31318
➜  bluegreen
```

```
^C%
➜  ~ while true ; do curl http://192.168.176.128:31318/version ; echo
'\n'; sleep 1; done
{"version": "1"}

{"version": "1"}

^C%
➜  ~
```

Пишем новую версию приложения

```
➜  bluegreen cd hello-py
➜  hello-py git:(master) ✗ vim app.py
➜  hello-py git:(master) ✗ ccat app.py
import os
import json

from flask import Flask
app = Flask(__name__)

@app.route("/health")
def health():
    return '{"status": "ok"}'

@app.route("/version")
def version():
    return '{"version": "2"}'

@app.route("/")
def hello():
    return 'Hello world from ' + os.environ['HOSTNAME'] + '!'

if __name__ == "__main__":
    app.run(host='0.0.0.0',port='80')
➜  hello-py git:(master) ✗
```

Собираем новый image

```
➜  hello-py git:(master) ✗ docker build -t hello-py:v2 .
Sending build context to Docker daemon  64.51kB
Step 1/3 : FROM python:3.5-onbuild
# Executing 3 build triggers
 ---> Using cache
 ---> Using cache
 ---> a3eae0bd6697
Step 2/3 : EXPOSE 8000
 ---> Running in 9c58694990a2
Removing intermediate container 9c58694990a2
 ---> 813502ccbbc8
```

```
Step 3/3 : ENTRYPOINT ["python", "/usr/src/app/app.py"]
 ---> Running in 3d71b698d137
Removing intermediate container 3d71b698d137
 ---> 92b3364a3617
Successfully built 92b3364a3617
Successfully tagged hello-py:v2
➜  hello-py git:(master) ✗
```

А теперь для новго релиза создаем новый деплоймент:

```
➜  hello-py git:(master) ✗ cd ..
➜  bluegreen vim deployment-new.yaml
➜  bluegreen ccat deployment-new.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
    name: hello-deployment-new
spec:
    replicas: 2
    selector:
       matchLabels:
          app: hello-py-app-new
    template:
       metadata:
          labels:
             app: hello-py-app-new
       spec:
          containers:
          - name: hello-py
            image: hello-py:v2
            ports:
              - name: web
                containerPort: 80
➜  bluegreen
```

И раскатываем новую версию рядом с первой

```
➜  bluegreen kubectl apply -f deployment-new.yaml
deployment.apps/hello-deployment-new created
➜  bluegreen kubectl get all
```

```
NAME                                        READY   STATUS    RESTARTS
AGE
pod/hello-deployment-58fcff8c8f-9f82q       1/1     Running   0
3m37s
pod/hello-deployment-58fcff8c8f-g6h86       1/1     Running   0
3m37s
pod/hello-deployment-new-85848cc995-5hghm   1/1     Running   0
10s
pod/hello-deployment-new-85848cc995-w45qr   1/1     Running   0
10s

NAME                    TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)
AGE
service/hello-service   NodePort   10.108.32.92   <none>
9000:31318/TCP   3m37s

NAME                                    READY   UP-TO-DATE   AVAILABLE
AGE
deployment.apps/hello-deployment        2/2     2            2
3m37s
deployment.apps/hello-deployment-new    2/2     2            2
10s

NAME                                              DESIRED   CURRENT
READY   AGE
replicaset.apps/hello-deployment-58fcff8c8f       2         2         2
3m37s
replicaset.apps/hello-deployment-new-85848cc995   2         2         2
10s
➜  bluegreen
```

Видим, что сервис ведет все запросы на под с селектором app=hello-py-app

```
➜  bluegreen  kubectl describe svc/hello-service
Name:                     hello-service
Namespace:                myapp
Labels:                   <none>
Annotations:              Selector:  app=hello-py-app
Type:                     NodePort
IP:                       10.108.32.92
```

```
Port:                     <unset>  9000/TCP
TargetPort:               web/TCP
NodePort:                 <unset>  31318/TCP
Endpoints:                172.17.0.4:80,172.17.0.5:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
➜  bluegreen
```

Но с помощью patch метода, мы можем изменить ресурс сервиса, чтобы он захватывал поды из нового деплоймента, или из старого:

```
➜  bluegreen kubectl patch svc/hello-service -p '{"spec": {"selector":
{"app": "hello-py-app-new"}}}'
service/hello-service patched
➜  bluegreen kubectl patch svc/hello-service -p '{"spec": {"selector":
{"app": "hello-py-app"}}}'
service/hello-service patched
➜  bluegreen
```

```
^C%
➜  ~ while true ; do curl http://192.168.176.128:31318/version ; echo
'\n'; sleep 1; done
{"version": "1"}
{"version": "1"}
{"version": "1"}
{"version": "1"}
{"version": "2"}
{"version": "2"}
{"version": "2"}
{"version": "2"}
{"version": "1"}
{"version": "1"}
{"version": "1"}
```

```
^C%
➜  ~
```

Видим, что одновременно либо первая, либо вторая версия на проде

INGRESS

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

Конечно, хотелось бы вытаскивать сервисы не просто на какие-то странные порты, а полноценную роутинг. Для этого существует такое понятие, как ingress.

```
➜  ingress ls
deployment.yaml hello-py        service.yaml    skaffold.yaml
```

```
➜  ingress ccat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
    name: hello-deployment
spec:
    replicas: 2
    selector:
       matchLabels:
          app: hello-py-app
    template:
       metadata:
          labels:
             app: hello-py-app
       spec:
          containers:
          - name: hello-py
            image: hello-py:v1
            ports:
              - name: web
                containerPort: 80
```

```
➜  ingress ccat hello-py/app.py
import os
import json

from flask import Flask
app = Flask(__name__)

@app.route("/health")
def health():
    return '{"status": "ok"}'

@app.route("/version")
def health():
    return '{"version": "0.2"}'

@app.route("/")
def hello():
    return 'Hello world from ' + os.environ['HOSTNAME'] + '!'

if __name__ == "__main__":
    app.run(host='0.0.0.0',port='80')
➜  ingress
```

Для простоты поставим ingress контроллер в составе кубика (nginx ingress controller)

```
➜  ingress minikube addons enable ingress
🌟  The 'ingress' addon is enabled
```

```
➜  ingress kubectl get pods -n kube-system
NAME                               READY   STATUS
RESTARTS   AGE
coredns-66bff467f8-d6464           1/1     Running             0
41h
coredns-66bff467f8-l6dqc           1/1     Running             0
41h
etcd-minikube                      1/1     Running             0
41h
```

```
kube-apiserver-minikube                     1/1     Running             0
41h
kube-controller-manager-minikube            1/1     Running             1
41h
kube-proxy-wwq5l                            1/1     Running             0
41h
kube-scheduler-minikube                     1/1     Running             1
41h
nginx-ingress-controller-6d57c87cb9-px26s   0/1     ContainerCreating   0
22s
storage-provisioner                         1/1     Running             0
41h
➜  ingress
```

```
➜  ingress kubectl get pods -n kube-system | grep ingress
nginx-ingress-controller-6d57c87cb9-px26s   0/1     ContainerCreating   0
17s
```

Ингресс - это объект (ресурс), который существует внутри кубика, который читает ingress-controller. Если нет ингресс-контроллера - ресурс можем создать, но он ни на что влиять не будет. Поэтому в начале ставим контроллер (в кластер кубика), потом создаем ингрессы для своих проектов и сервисов.

```
➜  ingress vim ingress.yaml
➜  ingress ccat ingress.yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: hello-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$2
spec:
  rules:
  - host: hello.world
    http:
      paths:
        - path: /myapp($|/)(.*)
          backend:
            serviceName: hello-service
```

```
            servicePort: 9000
➜  ingress
```

Такая конфигурация говорит, чтобы nginx стоял перед всем входящим трафиком и принимал запросы на домен hello.world и все, что приходит на урл /myapp* пересылалось в сервис.

```
➜  ingress kubectl apply -f ingress.yaml
ingress.extensions/hello-ingress created
➜  ingress

➜  ingress kubectl get ing
NAME            CLASS    HOSTS         ADDRESS   PORTS   AGE
hello-ingress   <none>   hello.world             80      6s
➜  ingress
```

Если просто урлу пойти, то будет 404, потому что Host-а нет.

```
➜  ~ curl http://192.168.176.128/myapp
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>openresty/1.15.8.2</center>
</body>
</html>
➜  ~
```

```
➜  ~ curl -H 'Host: hello.world' http://192.168.176.128/myapp
Hello world from hello-deployment-58fcff8c8f-9f82q!%
➜  ~
```

```
➜  ~ curl -H 'Host: hello.world' http://192.168.176.128/myapp/version
{"version": "1"}%
➜  ~
```

```
➜  ~ curl -H 'Host: hello.world' http://192.168.176.128/myapp/health
{"status": "ok"}%
➜  ~
```

STATEFULSETS

https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/ - про statefulsets

https://kubernetes.io/docs/concepts/storage/volumes/ - все про хранилища

https://kubernetes.io/docs/concepts/storage/persistent-volumes/ - все про персистентные хранилища

Все хорошо для `stateless` приложений. Нужно же понимать, что хоть фс своя у контейнеров и есть, но она удаляется, как только убили под. И кроме того, для `stateful` приложений нужен другой флоу деплоя и гарантий. Например, строгий последовательный порядок выполнения, невозможность прицепить один и тот же волюм для двух разных подов (в случае, если одна из нод кластера стала невидима, а потом вернулась) и т.д. Также имена подов не случайны и зараннее известны: `stateful-0, stateful-1, stateful-2. Это может быть нужно, если мы собираем в кластер, например, etcd.`

Сначала удалим все.

```
➜  statefulset kubectl delete all --all
```

Для постгреса сделаем такой `stateful` сет

```
➜  statefulset ls
postgres.yaml
```

```
➜  statefulset ccat postgres.yaml
---
apiVersion: v1
kind: Service
metadata:
  name: postgres
  labels:
    app: postgres
```

```yaml
spec:
  type: NodePort
  ports:
   - port: 5432
  selector:
    app: postgres

---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: postgres-statefulset
spec:
  serviceName: "postgres"
  replicas: 1
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
      - name: postgres
        image: postgres:latest
        ports:
        - containerPort: 5432
          name: postgredb
        env:
          - name: POSTGRES_DB
            value: myapp
          - name: POSTGRES_USER
            value: myuser
          - name: POSTGRES_PASSWORD
            value: passwd
        volumeMounts:
        - name: postgredb
          mountPath: /var/lib/postgresql/data
          subPath: postgres
  volumeClaimTemplates:
```

```
  - metadata:
      name: postgredb
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: standard
      resources:
        requests:
          storage: 1Gi
➜  statefulset
```

VolumeClaimTemplate - это шаблон запроса на persistent volume.

Прямо сейчас нет ни persistent volume (хранилища), ни persistent volume claim (запроса на хранилище)

```
➜  statefulset kubectl get pv
No resources found in myapp namespace.
```

```
➜  statefulset kubectl get pvc
No resources found in myapp namespace.
➜  statefulset
```

Для того, чтобы запросы на хранилище обрабатывались автоматически, существует специальный контроллер - storage provisioner. Он подчищает persistent volume, после удаления запроса, и он выделяет persistent volume и связывает его запросом.

```
➜  statefulset minikube addons list
|-----------------------------|----------|--------------|
|         ADDON NAME          | PROFILE  |    STATUS    |
|-----------------------------|----------|--------------|
| dashboard                   | minikube | disabled     |
| default-storageclass        | minikube | enabled ✅   |
| efk                         | minikube | disabled     |
| freshpod                    | minikube | disabled     |
| gvisor                      | minikube | disabled     |
| helm-tiller                 | minikube | disabled     |
| ingress                     | minikube | enabled ✅   |
```

```
| ingress-dns                 | minikube | disabled     |
| istio                       | minikube | disabled     |
| istio-provisioner           | minikube | disabled     |
| logviewer                   | minikube | disabled     |
| metrics-server              | minikube | disabled     |
| nvidia-driver-installer     | minikube | disabled     |
| nvidia-gpu-device-plugin    | minikube | disabled     |
| registry                    | minikube | disabled     |
| registry-aliases            | minikube | disabled     |
| registry-creds              | minikube | disabled     |
| storage-provisioner         | minikube | enabled ✅   |
| storage-provisioner-gluster | minikube | disabled     |
|-----------------------------|----------|--------------|
```

И так применяем манифест.

```
➜  statefulset kubectl apply -f postgres.yaml
service/postgres created
statefulset.apps/postgres-statefulset created
```

```
➜  statefulset kubectl get all
NAME                         READY   STATUS              RESTARTS   AGE
pod/postgres-statefulset-0   0/1     ContainerCreating   0          8s

NAME               TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)
AGE
service/postgres   NodePort   10.98.23.212   <none>        5432:30786/TCP
8s

NAME                                    READY   AGE
statefulset.apps/postgres-statefulset   0/1     8s
```

Видим, что создался persistent volume и привзяался к pvc

```
➜  statefulset kubectl get pv
```

```
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM
STORAGECLASS   REASON   AGE
pvc-964c4f12-5ae1-4615-bec0-da777033901c   1Gi        RWO
Delete           Bound    myapp/postgredb-postgres-statefulset-0
standard               18s
```

```
➜  statefulset kubectl get pvc
NAME                               STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS   AGE
postgredb-postgres-statefulset-0   Bound
pvc-964c4f12-5ae1-4615-bec0-da777033901c   1Gi        RWO
standard       20s
```

Теперь можем зайти в БД по кредам, которые оставляли в ENV переменных контейнера.

```
➜  statefulset minikube service postgres -n myapp --url
http://192.168.176.128:30786
```

```
➜  statefulset psql -h 192.168.176.128 -p 30786 -U myuser -W myapp
Password:
psql (12.2)
Type "help" for help.

myapp=# \dt
Did not find any relations.
myapp=# create table client (id bigint primary key, name varchar) ;
CREATE TABLE
myapp=# insert into client(id, name) values (1, 'vasya');
INSERT 0 1
myapp=# select * from client;
 id | name
----+-------
  1 | vasya
(1 row)
```

```
myapp=# \q
➜  statefulset
```

Если удалим под, он все-равно создастся, и данные все равно будут доступны.

```
➜  statefulset kubectl delete pod postgres-statefulset-0
pod "postgres-statefulset-0" deleted
```

```
NAME                          READY   STATUS        RESTARTS   AGE
pod/postgres-statefulset-0    0/1     Terminating   0          48s
```

```
➜  statefulset kubectl get pods
NAME                     READY   STATUS    RESTARTS   AGE
postgres-statefulset-0   1/1     Running   0          15s
```

```
➜  statefulset psql -h 192.168.176.128 -p 30786 -U myuser -W myapp
Password:
psql (12.2)
Type "help" for help.

myapp=# select * from client;
 id | name
----+-------
  1 | vasya
(1 row)

myapp=# \q
➜  statefulset
```

CONFIGURATION

https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/

https://kubernetes.io/docs/concepts/configuration/secret/

Теперь давайте сконфигурируем приложение, чтобы оно работало вместе с БД.

```
➜  configuration ls
deployment.yaml hello-py        postgres.yaml   service.yaml
skaffold.yaml
```

```
➜  configuration ccat hello-py/app.py
import os
import json

from flask import Flask

app = Flask(__name__)

config = {
    'DATABASE_URI': os.environ.get('DATABASE_URI', ''),
    'HOSTNAME': os.environ['HOSTNAME'],
    'GREETING': os.environ.get('GREETING', 'Hello'),
}

@app.route("/")
def hello():
    return config['GREETING'] + ' from ' + config['HOSTNAME'] + '!'

@app.route("/config")
def configuration():
    return json.dumps(config)

@app.route('/db')
def db():
    from sqlalchemy import create_engine

    engine = create_engine(config['DATABASE_URI'], echo=True)
    rows = []
    with engine.connect() as connection:
        result = connection.execute("select id, name from client;")
```

```
        rows = [dict(r.items()) for r in result]
    return json.dumps(rows)

if __name__ == "__main__":
    app.run(host='0.0.0.0', port='80', debug=True)
```

```
➜  configuration ccat hello-py/requirements.txt
Flask==1.1.2
Flask-SQLAlchemy
Flask-Migrate
psycopg2
Flask-Script
➜  configuration
```

Укажем в переменных окружения креды для доступа к БД.

```
➜  configuration ccat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
    name: hello-deployment
spec:
    replicas: 2
    selector:
       matchLabels:
          app: hello-py-app
    template:
       metadata:
          labels:
             app: hello-py-app
       spec:
          containers:
          - name: hello-py
            image: hello-py:v1
            env:
              - name: DATABASE_URI
                value: postgresql+psycopg2://myuser:passwd@postgres/myapp
            ports:
              - name: web
```

```
                containerPort: 80
```

```
➜  configuration ccat service.yaml
apiVersion: v1
kind: Service
metadata:
  name: hello-service
spec:
  selector:
    app: hello-py-app
  ports:
    - protocol: TCP
      port: 9000
      targetPort: web
  type: NodePort
➜  configuration
```

```
➜  configuration ccat skaffold.yaml
apiVersion: skaffold/v2alpha4
kind: Config
metadata:
  name: configuration
build:
  artifacts:
  - image: hello-py
    context: hello-py
deploy:
  kubectl:
    manifests:
    - deployment.yaml
    - service.yaml
➜  configuration
```

Билдим и собираем с помощью `skaffold`.

```
➜  configuration skaffold run
Generating tags...
 - hello-py -> hello-py:fe90391-dirty
```

```
Checking cache...
 - hello-py: Found Locally
Tags used in deployment:
 - hello-py ->
hello-py:8aee95eba2c404dc597d213ad0db4fc66a58569f41f031b9d9ce7c534baf91ae
   local images can't be referenced by digest. They are tagged and
referenced by a unique ID instead
Starting deploy...
 - deployment.apps/hello-deployment configured
 - service/hello-service configured
You can also run [skaffold run --tail] to get the logs
There is a new version (1.7.0) of Skaffold available. Download it at
https://storage.googleapis.com/skaffold/releases/latest/skaffold-darwin-am
d64
```

```
➜  configuration kubectl get all
NAME                                   READY   STATUS    RESTARTS   AGE
pod/hello-deployment-8cdb69dbf-k65wd   1/1     Running   0          3m40s
pod/hello-deployment-8cdb69dbf-vhrr8   1/1     Running   0          3m39s
pod/postgres-statefulset-0             1/1     Running   0          86m

NAME                    TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
service/hello-service   NodePort   10.107.60.245   <none>
9000:31399/TCP   25m
service/postgres        NodePort   10.98.23.212    <none>
5432:30786/TCP   95m

NAME                               READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hello-deployment   2/2     2            2           27m

NAME                                          DESIRED   CURRENT   READY
AGE
replicaset.apps/hello-deployment-856df94dc    0         0         0
27m
replicaset.apps/hello-deployment-8cdb69dbf    2         2         2
3m40s

NAME                                    READY   AGE
statefulset.apps/postgres-statefulset   1/1     95m
```

```
➜  configuration minikube service -n myapp --url hello-service
http://192.168.176.128:31399
➜  configuration
```

Посмотрим, что отдает сервис.

```
➜  ~ curl -s http://192.168.176.128:31399/
Hello from hello-deployment-8cdb69dbf-k65wd!%
➜  ~
```

```
➜  ~ curl -s http://192.168.176.128:31399/config  | jq
{
  "GREETING": "Hello",
  "DATABASE_URI": "postgresql+psycopg2://myuser:passwd@postgres/myapp",
  "HOSTNAME": "hello-deployment-8cdb69dbf-vhrr8"
}
```

И он отдает те данные, которые мы создали в БД .

```
➜  ~ curl -s http://192.168.176.128:31399/db | jq
[
  {
    "id": 1,
    "name": "vasya"
  }
]
```

И если обновим данные в таблице БД

```
➜  configuration psql -h 192.168.176.128 -p 30786 -U myuser -W myapp
Password:
psql (12.2)
Type "help" for help.

myapp=# insert into client(id, name) values (2, 'petya');
```

```
INSERT 0 1
myapp=# \q
➜  configuration
```

То увидим их в сервисе.

```
➜  ~ curl -s http://192.168.176.128:31399/db | jq
[
  {
    "id": 1,
    "name": "vasya"
  },
  {
    "id": 2,
    "name": "petya"
  }
]
➜  ~
```

Давайте теперь попробуем обновить env переменые у деплоймента.

```
➜  configuration kubectl set env deploy hello-deployment GREETING=Aloha
deployment.apps/hello-deployment env updated
```

Это сразу же приведет к пересозданию подов.

```
➜  configuration kubectl get pods
NAME                                READY   STATUS        RESTARTS   AGE
hello-deployment-849fcc74db-76xvc   1/1     Running       0          4s
hello-deployment-849fcc74db-xfh7l   1/1     Running       0          3s
hello-deployment-8cdb69dbf-k65wd    0/1     Terminating   0          10m
hello-deployment-8cdb69dbf-vhrr8    0/1     Terminating   0          10m
postgres-statefulset-0              1/1     Running       0          93m
```

```
➜  ~ curl -s http://192.168.176.128:31399/config | jq
{
```

```
  "GREETING": "Aloha",
  "HOSTNAME": "hello-deployment-849fcc74db-xfh71",
  "DATABASE_URI": "postgresql+psycopg2://myuser:passwd@postgres/myapp"
}
➜  ~
```

Вообще говоря хранить конфиги в деплойменте не очень хорошо. Потому что иногда хочется поменять конфиг без изменения деплоймента (например, если манифесты стороннего приложения). И следуя 12 факторным приложениям, для разных сред мы должны иметь один деплоймент, но разные конфигурации.

Чтобы конфигурации отделить от деплоймента, используется сущность ConfigMap в кубернетесе.

Перенесем конфигурации в config map.

```
➜  configuration vim app-config.yaml
➜  configuration ccat app-config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: hello-config
data:
  DATABASE_URI: postgresql+psycopg2://myuser:passwd@postgres/myapp
  GREETING: Bonjour
➜  configuration
```

И будем ссылаться на эти значения в деплойменте.

```
➜  configuration ccat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
    name: hello-deployment
spec:
    replicas: 2
    selector:
       matchLabels:
          app: hello-py-app
```

```
   template:
      metadata:
         labels:
            app: hello-py-app
      spec:
         containers:
         - name: hello-py
           image: hello-py:v1
           env:
             - name: DATABASE_URI
               valueFrom:
                 configMapKeyRef:
                   name: hello-config
                   key: DATABASE_URI
             - name: GREETING
               valueFrom:
                 configMapKeyRef:
                   name: hello-config
                   key: GREETING
           ports:
             - name: web
               containerPort: 80
➜  configuration
```

Добавляем `app-config.yaml` и убираем `postgresql.yaml` из скаффолда и запускаем сборку и деплой.

```
➜  configuration ccat skaffold.yaml
apiVersion: skaffold/v2alpha4
kind: Config
metadata:
  name: configuration
build:
  artifacts:
  - image: hello-py
    context: hello-py
deploy:
  kubectl:
    manifests:
    - app-config.yaml
    - service.yaml
```

```
    - deployment.yaml
➜  configuration
```

И получаем ошибку, потому что valueFrom заменили на value.

```
➜  configuration kubectl apply -f deployment.yaml
The Deployment "hello-deployment" is invalid:
spec.template.spec.containers[0].env[1].valueFrom: Invalid value: "": may
not be specified when `value` is not empty
```

В этом случае костыльно убиваем деплоймент

```
➜  configuration skaffold delete
Cleaning up...
 - configmap "hello-config" deleted
 - service "hello-service" deleted
 - deployment.apps "hello-deployment" deleted
There is a new version (1.7.0) of Skaffold available. Download it at
https://storage.googleapis.com/skaffold/releases/latest/skaffold-darwin-am
d64
```

```
➜  configuration kubectl get all
NAME                          READY   STATUS    RESTARTS   AGE
pod/postgres-statefulset-0    1/1     Running   0          3h19m

NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
service/postgres     NodePort    10.98.23.212    <none>        5432:30786/TCP
3h27m

NAME                                     READY   AGE
statefulset.apps/postgres-statefulset    1/1     3h27m
➜  configuration
```

И снова запускаем

```
➜  configuration skaffold run
Generating tags...
 - hello-py -> hello-py:fe90391-dirty
```

```
Checking cache...
 - hello-py: Found Locally
Tags used in deployment:
 - hello-py ->
hello-py:5f59785aca2337c09cc9cd7fb00bf7df6e42190484dc7cb10448a7ec4ed2aaa3
   local images can't be referenced by digest. They are tagged and
referenced by a unique ID instead
Starting deploy...
 - configmap/hello-config created
 - service/hello-service created
 - deployment.apps/hello-deployment created
You can also run [skaffold run --tail] to get the logs
There is a new version (1.7.0) of Skaffold available. Download it at
https://storage.googleapis.com/skaffold/releases/latest/skaffold-darwin-am
d64

➜  configuration
```

```
➜  configuration kubectl get all
NAME                                   READY   STATUS    RESTARTS   AGE
pod/hello-deployment-766cdc78f8-l5jch   1/1     Running   0          26s
pod/hello-deployment-766cdc78f8-s5g5t   1/1     Running   0          26s
pod/postgres-statefulset-0              1/1     Running   0          3h19m

NAME                    TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)
AGE
service/hello-service   NodePort   10.99.177.89   <none>
9000:31301/TCP   26s
service/postgres        NodePort   10.98.23.212   <none>
5432:30786/TCP   3h28m

NAME                               READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hello-deployment   2/2     2            2           26s

NAME                                         DESIRED   CURRENT   READY
AGE
replicaset.apps/hello-deployment-766cdc78f8   2         2         2
26s

NAME                                   READY   AGE
```

```
statefulset.apps/postgres-statefulset   1/1     3h28m
➜  configuration
```

Делаем запрос и проверяем, что все хорошо передалось.

```
➜  configuration minikube service hello-service -n myapp --url
http://192.168.176.128:31301
```

```
➜  ~ curl -s http://192.168.176.128:31301/config | jq
{
  "HOSTNAME": "hello-deployment-766cdc78f8-l5jch",
  "GREETING": "Bonjour",
  "DATABASE_URI": "postgresql+psycopg2://myuser:passwd@postgres/myapp"
}
➜  ~
```

Держать пароли в конфигах очень не секурно, поэтому для сенсетив данных используют секреты `Secrets`. По сути это тот же конфигмап, просто значения заэнкожены в `base64`. По сути это избавляет только от случайного подглядывания.

```
➜  configuration echo
'postgresql+psycopg2://myuser:passwd@postgres/myapp' | base64
cG9zdGdyZXNxbCtwc3ljb3BnMjovL215dXNlcjpwYXNzd2RAcG9zdGdyZXMvbXlhcHAK
```

```
➜  configuration ccat app-config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: hello-config
data:
  GREETING: Bonjour
---
apiVersion: v1
kind: Secret
metadata:
  name: hello-secret
```

```
type: Opaque
data:
  DATABASE_URI:
cG9zdGdyZXNxbCtwc3ljb3BnMjovL215dXNlcjpwYXNzd2RAcG9zdGdyZXMvbXlhcHAK
➜  configuration
```

```
➜  configuration ccat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
    name: hello-deployment
spec:
    replicas: 2
    selector:
       matchLabels:
          app: hello-py-app
    template:
       metadata:
          labels:
             app: hello-py-app
       spec:
          containers:
          - name: hello-py
            image: hello-py:v1
            env:
              - name: DATABASE_URI
                valueFrom:
                  secretKeyRef:
                    name: hello-secret
                    key: DATABASE_URI
              - name: GREETING
                valueFrom:
                  configMapKeyRef:
                    name: hello-config
                    key: GREETING
            ports:
              - name: web
                containerPort: 80
```

Редеплоим и проверяем, что все нормально собралось.

```
➜  configuration skaffold run
Generating tags...
 - hello-py -> hello-py:fe90391-dirty
Checking cache...
 - hello-py: Found Locally
Tags used in deployment:
 - hello-py ->
hello-py:5f59785aca2337c09cc9cd7fb00bf7df6e42190484dc7cb10448a7ec4ed2aaa3
   local images can't be referenced by digest. They are tagged and
referenced by a unique ID instead
Starting deploy...
 - configmap/hello-config configured
 - secret/hello-secret configured
 - service/hello-service configured
 - deployment.apps/hello-deployment configured
You can also run [skaffold run --tail] to get the logs
There is a new version (1.7.0) of Skaffold available. Download it at
https://storage.googleapis.com/skaffold/releases/latest/skaffold-darwin-am
d64

➜  configuration
```

```
➜  ~ curl -s http://192.168.176.128:31301/config | jq
{
  "HOSTNAME": "hello-deployment-5d8d9cb585-kp6lh",
  "DATABASE_URI": "postgresql+psycopg2://myuser:passwd@postgres/myapp\n",
  "GREETING": "Bonjour"
}
```

```
➜  ~ curl -s http://192.168.176.128:31301/db | jq
[
  {
    "id": 1,
    "name": "vasya"
  },
```

```
  {
    "id": 2,
    "name": "petya"
  }
]
➜  ~
```

В целом набор статичных манифестов не так хорошо работает. Например, если мы хотим зарелизить два сервиса рядом, нам для них придется писать 2 разных деплоя, которые будут по сути одинаковыми.

Если рассматривать кубернетес как операционную систему, а ресурсы и манифесты, как ассемблер, то не хватает пакетного менеджера. Который бы мог разруливать зависимости, и собирать приложение из n-ого количества сервисов.

На самом деле такой менеджер есть и он называется `helm` (тут примеры на `helm3`)

HELM

```
➜  helm ls
app-config.yaml hello-py        service.yaml
deployment.yaml postgres.yaml   skaffold.yaml
```

Пакеты в нотации хелм называются чартами. Давайте создадим хелм чарт для нашего небольшого приложения.

```
➜  helm helm create hello-chart
Creating hello-chart
```

```
➜  helm cd hello-chart
➜  hello-chart
➜  hello-chart ls
Chart.yaml  charts      templates   values.yaml
➜  hello-chart
```

Поменяем стандартные шаблоны, так, чтобы они соответстовали нашим манифестам

```
➜  helm tree hello-chart
hello-chart
├── Chart.yaml
├── charts
├── templates
│   ├── NOTES.txt
│   ├── _helpers.tpl
│   ├── config.yaml
│   ├── deployment.yaml
│   ├── service.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml

3 directories, 8 files
➜  helm
```

В `Chart.yaml` описываем наше приложение.

```
➜  helm ccat hello-chart/Chart.yaml
apiVersion: v2
name: hello-chart
description: A Helm chart for Kubernetes

type: application

version: 0.3.0
appVersion: 0.3.0
➜  helm
```

в `values.yaml` описываем настройки приложения и их дефолтные значения .

```
➜  helm ccat hello-chart/values.yaml
replicaCount: 2
```

```
image:
  repository: hello-py

service:
  type: NodePort
  port: 9000

externalPostgresql:
  postgresqlUsername: myuser
  postgresqlPassword: passwd
  postgresqlDatabase: myapp
  postgresqlHost: "postgres"
  postgresqlPort: "5432"
➜  helm
```

В шаблонах собственно лежат шаблонизированные манифесты приложения.

```
➜  helm ccat hello-chart/templates/service.yaml
apiVersion: v1
kind: Service
metadata:
  name: {{ include "hello-chart.fullname" . }}
  labels:
    {{- include "hello-chart.labels" . | nindent 4 }}
spec:
  type: {{ .Values.service.type }}
  ports:
    - port: {{ .Values.service.port }}
      targetPort: http
      protocol: TCP
      name: web
  selector:
    {{- include "hello-chart.selectorLabels" . | nindent 4 }}
➜  helm
```

В деплойменте мы еще добавили аннотацию про `config.yaml` - это такой способ передеплоить приложение с пересозданием подов, если поменяли конфигмапу.

```
➜  helm ccat hello-chart/templates/deployment.yaml
apiVersion: apps/v1
```

```
kind: Deployment
metadata:
  name: {{ include "hello-chart.fullname" . }}
  labels:
    {{- include "hello-chart.labels" . | nindent 4 }}
spec:
  replicas: {{ .Values.replicaCount }}
  selector:
    matchLabels:
      {{- include "hello-chart.selectorLabels" . | nindent 6 }}
  template:
    metadata:
      labels:
        {{- include "hello-chart.selectorLabels" . | nindent 8 }}
      annotations:
        checksum/config: {{ include (print $.Template.BasePath
"/config.yaml") . | sha256sum }}
    spec:
      containers:
        - name: {{ .Chart.Name }}
          image: "{{ .Values.image.repository }}:{{ .Chart.AppVersion }}"
          ports:
            - name: http
              containerPort: 80
              protocol: TCP
          env:
            - name: DATABASE_URI
              valueFrom:
                secretKeyRef:
                  name: hello-secret
                  key: DATABASE_URI
            - name: GREETING
              valueFrom:
                configMapKeyRef:
                  name: hello-config
                  key: GREETING
➜  helm
```

```
➜  helm ccat hello-chart/templates/config.yaml
apiVersion: v1
```

```
kind: ConfigMap
metadata:
  name: hello-config
data:
  GREETING: Bonjour
---
apiVersion: v1
kind: Secret
metadata:
  name: hello-secret
type: Opaque
data:
  DATABASE_URI: {{ printf "postgresql+psycopg2://%s:%s@%s:%s/%s"
.Values.externalPostgresql.postgresqlUsername
.Values.externalPostgresql.postgresqlPassword
.Values.externalPostgresql.postgresqlHost
.Values.externalPostgresql.postgresqlPort
.Values.externalPostgresql.postgresqlDatabase  | b64enc | quote }}
➜  helm
```

В секретах используем старый добрый base64 encoder

```
➜  helm ccat hello-chart/templates/NOTES.txt
TO BE DONE
➜  helm
```

```
➜  helm kubectl get all
NAME                          READY   STATUS    RESTARTS   AGE
pod/postgres-statefulset-0    1/1     Running   0          31h

NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
service/postgres     NodePort    10.98.23.212    <none>        5432:30786/TCP
31h

NAME                                     READY   AGE
statefulset.apps/postgres-statefulset    1/1     31h
➜  helm
```

Перед установкой по-настоящему посмотрим, как применились наши переменные, и какие получились в итоге манифесты.

```
➜  helm helm install myapp ./hello-chart --dry-run
NAME: myapp
LAST DEPLOYED: Thu Apr  9 00:09:09 2020
NAMESPACE: myapp
STATUS: pending-install
REVISION: 1
HOOKS:
---
# Source: hello-chart/templates/tests/test-connection.yaml
apiVersion: v1
kind: Pod
metadata:
  name: "myapp-hello-chart-test-connection"
  labels:
    helm.sh/chart: hello-chart-0.3.0
    app.kubernetes.io/name: hello-chart
    app.kubernetes.io/instance: myapp
    app.kubernetes.io/version: "0.3.0"
    app.kubernetes.io/managed-by: Helm
  annotations:
    "helm.sh/hook": test-success
spec:
  containers:
    - name: wget
      image: busybox
      command: ['wget']
      args: ['myapp-hello-chart:9000']
  restartPolicy: Never
MANIFEST:
---
# Source: hello-chart/templates/config.yaml
apiVersion: v1
kind: Secret
metadata:
  name: hello-secret
type: Opaque
data:
```

```
  DATABASE_URI:
"cG9zdGdyZXNxbCtwc3ljb3BnMjovL215dXNlcjpwYXNzd2RAcG9zdGdyZXM6NTQzMi9teWFwc
A=="
---
# Source: hello-chart/templates/config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: hello-config
data:
  GREETING: Bonjour
---
# Source: hello-chart/templates/service.yaml
apiVersion: v1
kind: Service
metadata:
  name: myapp-hello-chart
  labels:
    helm.sh/chart: hello-chart-0.3.0
    app.kubernetes.io/name: hello-chart
    app.kubernetes.io/instance: myapp
    app.kubernetes.io/version: "0.3.0"
    app.kubernetes.io/managed-by: Helm
spec:
  type: NodePort
  ports:
    - port: 9000
      targetPort: http
      protocol: TCP
      name: web
  selector:
    app.kubernetes.io/name: hello-chart
    app.kubernetes.io/instance: myapp
---
# Source: hello-chart/templates/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-hello-chart
  labels:
    helm.sh/chart: hello-chart-0.3.0
```

```
    app.kubernetes.io/name: hello-chart
    app.kubernetes.io/instance: myapp
    app.kubernetes.io/version: "0.3.0"
    app.kubernetes.io/managed-by: Helm
spec:
  replicas: 2
  selector:
    matchLabels:
      app.kubernetes.io/name: hello-chart
      app.kubernetes.io/instance: myapp
  template:
    metadata:
      labels:
        app.kubernetes.io/name: hello-chart
        app.kubernetes.io/instance: myapp
      annotations:
        checksum/config:
8c4f1f22c8f31476c892030d7d4bdcebdb793d8802b9058fec733bc604bf03d7
    spec:
      containers:
        - name: hello-chart
          image: "hello-py:0.3.0"
          ports:
            - name: http
              containerPort: 80
              protocol: TCP
          env:
            - name: DATABASE_URI
              valueFrom:
                secretKeyRef:
                  name: hello-secret
                  key: DATABASE_URI
            - name: GREETING
              valueFrom:
                configMapKeyRef:
                  name: hello-config
                  key: GREETING

NOTES:
TO BE DONE
```

Давайте установим чарт. Релиз назовем myapp.

```
➜  helm helm install myapp ./hello-chart
NAME: myapp
LAST DEPLOYED: Thu Apr  9 00:09:49 2020
NAMESPACE: myapp
STATUS: deployed
REVISION: 1
NOTES:
TO BE DONE
➜  helm
```

```
➜  helm kubectl get all
NAME                                     READY   STATUS    RESTARTS   AGE
pod/myapp-hello-chart-57dfb8f5b6-625jk   1/1     Running   0          44s
pod/myapp-hello-chart-57dfb8f5b6-r8vqn   1/1     Running   0          44s
pod/postgres-statefulset-0               1/1     Running   0          31h

NAME                         TYPE       CLUSTER-IP       EXTERNAL-IP
PORT(S)          AGE
service/myapp-hello-chart    NodePort   10.107.102.133   <none>
9000:32528/TCP   45s
service/postgres             NodePort   10.98.23.212     <none>
5432:30786/TCP   31h

NAME                                 READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/myapp-hello-chart    2/2     2            2           45s

NAME                                           DESIRED   CURRENT   READY
AGE
replicaset.apps/myapp-hello-chart-57dfb8f5b6   2         2         2
45s

NAME                                   READY   AGE
statefulset.apps/postgres-statefulset  1/1     31h
➜  helm
```

```
➜  ~ curl -s http://192.168.176.128:32528/db | jq
[
  {
    "name": "vasya",
    "id": 1
  },
  {
    "name": "petya",
    "id": 2
  }
]
➜  ~
```

Теперь можно смело удалять захардкоженные манифесты.

```
➜  helm rm -fr app-config.yaml deployment.yaml service.yaml
```

```
➜  helm ls
hello-chart   hello-py      postgres.yaml
```

HELM-DEP

Для того, чтобы добавить в наше приложение зависимость, а хелм это позволяет, очистим namespace.

Очищаем неймспейс и удаляем pvc, чтобы не захламляли место.

```
➜  helm-dep helm uninstall myapp
release "myapp" uninstalled
```

```
➜  helm-dep kubectl delete all --all
pod "postgres-statefulset-0" deleted
```

```
service "postgres" deleted
statefulset.apps "postgres-statefulset" deleted
➜  helm-dep kubectl get pvc
NAME                                STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS   AGE
postgredb-postgres-statefulset-0   Bound
pvc-964c4f12-5ae1-4615-bec0-da777033901c   1Gi        RWO
standard       41h
➜  helm-dep kubectl delete pvc postgredb-postgres-statefulset-0
persistentvolumeclaim "postgredb-postgres-statefulset-0" deleted
➜  helm-dep kubectl get pv
No resources found in myapp namespace.
➜  helm-dep
```

Добавляем в чарте в зависимости postgresql. version - это версия чарта, а не приложения.

```
➜  helm-dep ccat hello-chart/Chart.yaml
apiVersion: v2
name: hello-chart
description: A Helm chart for Kubernetes

type: application

version: 0.4.0
appVersion: 0.3.0

dependencies:
  - name: postgresql
    version: 8.x.x
    repository: https://charts.bitnami.com/bitnami
    condition: postgresql.enabled
    tags:
      - myapp-database
➜  helm-dep
```

Устанавливаем зависимости. Они складываются в директорию charts/

```
➜  helm-dep helm dependency update ./hello-chart
```

```
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "cetic" chart repository
...Successfully got an update from the "bitnami" chart repository
...Successfully got an update from the "stable" chart repository
Update Complete. ⎈Happy Helming!⎈
Saving 1 charts
Downloading postgresql from repo https://charts.bitnami.com/bitnami
Deleting outdated charts
➜  helm-dep
```

```
➜  helm-dep tree .
.
├── hello-chart
│   ├── Chart.lock
│   ├── Chart.yaml
│   ├── charts
│   │   └── postgresql-8.7.2.tgz
│   ├── templates
│   │   ├── NOTES.txt
│   │   ├── _helpers.tpl
│   │   ├── config.yaml
│   │   ├── deployment.yaml
│   │   ├── service.yaml
│   │   └── tests
│   │       └── test-connection.yaml
│   └── values.yaml
└── hello-py
    ├── Dockerfile
    ├── app.py
    └── requirements.txt

5 directories, 13 files
```

Для того, чтобы изменить параметры сабчарта `postgresql`, мы должны использовать парметр `postgresql`, в котором переопределить `Values` для этого пакета (в соответствующей этому пакету структуре и именам переменных)

```
➜  helm-dep vim  hello-chart/values.yaml
➜  helm-dep
```

```
➜  helm-dep ccat hello-chart/values.yaml
replicaCount: 2

image:
  repository: hello-py

service:
  type: NodePort
  port: 9000

postgresql:
  enabled: true
  postgresqlUsername: myuser
  postgresqlPassword: passwd
  postgresqlDatabase: myapp
  service:
    port: "5432"
```

Соответствующим образом переписываем конфиги

```
➜  helm-dep ccat  hello-chart/templates/config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: hello-config
data:
  GREETING: Bonjour
---
apiVersion: v1
kind: Secret
metadata:
  name: hello-secret
type: Opaque
data:
  DATABASE_URI: {{ printf "postgresql+psycopg2://%s:%s@%s:%s/%s"
.Values.postgresql.postgresqlUsername
.Values.postgresql.postgresqlPassword (include "postgresql.fullname" .)
```

```
.Values.postgresql.service.port .Values.postgresql.postgresqlDatabase  |
b64enc | quote }}
```

Для того, чтобы обращаться по имени хоста добавляем шаблон `postgresql.fullname`. (Только его, остальное из дефолтного чарта при создании)

```
➜  helm-dep ccat hello-chart/templates/_helpers.tpl
{{/* vim: set filetype=mustache: */}}
{{/*
Expand the name of the chart.
*/}}
{{- define "hello-chart.name" -}}
{{- default .Chart.Name .Values.nameOverride | trunc 63 | trimSuffix "-"
-}}
{{- end -}}

{{/*
Create a default fully qualified app name.
We truncate at 63 chars because some Kubernetes name fields are limited to
this (by the DNS naming spec).
If release name contains chart name it will be used as a full name.
*/}}
{{- define "hello-chart.fullname" -}}
{{- if .Values.fullnameOverride -}}
{{- .Values.fullnameOverride | trunc 63 | trimSuffix "-" -}}
{{- else -}}
{{- $name := default .Chart.Name .Values.nameOverride -}}
{{- if contains $name .Release.Name -}}
{{- .Release.Name | trunc 63 | trimSuffix "-" -}}
{{- else -}}
{{- printf "%s-%s" .Release.Name $name | trunc 63 | trimSuffix "-" -}}
{{- end -}}
{{- end -}}
{{- end -}}

{{/*
Create chart name and version as used by the chart label.
*/}}
{{- define "hello-chart.chart" -}}
```

```
{{- printf "%s-%s" .Chart.Name .Chart.Version | replace "+" "_" | trunc 63
| trimSuffix "-" -}}
{{- end -}}

{{/*
Common labels
*/}}
{{- define "hello-chart.labels" -}}
helm.sh/chart: {{ include "hello-chart.chart" . }}
{{ include "hello-chart.selectorLabels" . }}
{{- if .Chart.AppVersion }}
app.kubernetes.io/version: {{ .Chart.AppVersion | quote }}
{{- end }}
app.kubernetes.io/managed-by: {{ .Release.Service }}
{{- end -}}

{{/*
Selector labels
*/}}
{{- define "hello-chart.selectorLabels" -}}
app.kubernetes.io/name: {{ include "hello-chart.name" . }}
app.kubernetes.io/instance: {{ .Release.Name }}
{{- end -}}


Create a default fully qualified app name.
We truncate at 63 chars because some Kubernetes name fields are limited to
this (by the DNS naming spec).
*/}}
{{- define "postgresql.fullname" -}}
{{- printf "%s-%s" .Release.Name "postgresql" | trunc 63 | trimSuffix "-"
-}}
{{- end -}}
➜  helm-dep
```

Теперь можно устанавливать релиз `myapp` и нашего чарта.

```
➜  helm-dep helm install myapp ./hello-chart
NAME: myapp
```

```
LAST DEPLOYED: Thu Apr  9 10:30:15 2020
NAMESPACE: myapp
STATUS: deployed
REVISION: 1
NOTES:
TO BE DONE
```

```
➜  helm-dep kubectl get all
NAME                                        READY   STATUS    RESTARTS   AGE
pod/myapp-hello-chart-84d58b59b7-jvz6z      1/1     Running   0
5m56s
pod/myapp-hello-chart-84d58b59b7-pxhw6      1/1     Running   0
5m56s
pod/myapp-postgresql-0                      1/1     Running   0
5m56s

NAME                                  TYPE        CLUSTER-IP
EXTERNAL-IP   PORT(S)          AGE
service/myapp-hello-chart             NodePort    10.96.214.45    <none>
9000:31750/TCP   5m56s
service/myapp-postgresql              ClusterIP   10.109.207.51   <none>
5432/TCP         5m56s
service/myapp-postgresql-headless     ClusterIP   None            <none>
5432/TCP         5m56s

NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/myapp-hello-chart       2/2     2            2           5m56s

NAME                                               DESIRED   CURRENT   READY
AGE
replicaset.apps/myapp-hello-chart-84d58b59b7       2         2         2
5m56s

NAME                                  READY   AGE
statefulset.apps/myapp-postgresql     1/1     5m56s
➜  helm-dep
```

```
➜  ~ curl -s http://192.168.176.128:30495/config | jq
```

```
{
  "HOSTNAME": "myapp-hello-chart-84d58b59b7-sp5vg",
  "GREETING": "Bonjour",
  "DATABASE_URI":
"postgresql+psycopg2://myuser:passwd@myapp-postgresql:5432/myapp"
}
```

Обращаемся и видим ошибку.

```
➜  ~ curl  http://192.168.176.128:30495/db
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>sqlalchemy.exc.ProgrammingError:
(psycopg2.errors.UndefinedTable) relation &quot;client&quot; does not
exist
LINE 1: select id, name from client;
                             ^
```

JOB
Давайте сделаем миграцию

```
➜  helm-dep vim initdb.yaml
```

```
➜  helm-dep ccat initdb.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: "myapp"
spec:
  template:
    metadata:
      name: "myapp"
    spec:
      restartPolicy: Never
      containers:
      - name: post-install-job
```

```
        image: postgres:latest
        env:
          - name: POSTGRES_PASSWORD
            value: "passwd"
        command:
          - sh
          - "-c"
          - |
            psql postgres://myuser:passwd@myapp-postgresql/myapp <<'EOF'
              create table client (id bigint primary key, name varchar);
              insert into client (id, name) values (1, 'Konstantin');
            EOF

  backoffLimit: 0
➜  helm-dep
```

```
➜  helm-dep kubectl apply -f initdb.yaml
job.batch/myapp created
```

```
➜  helm-dep kubectl logs pod/myapp-cprv4
CREATE TABLE
INSERT 0 1
```

```
➜  ~ curl -s http://192.168.176.128:30495/db  | jq
[
  {
    "name": "Konstantin",
    "id": 1
  }
]
➜  ~
```

Все мы закончили знакомство с Kubernetes!