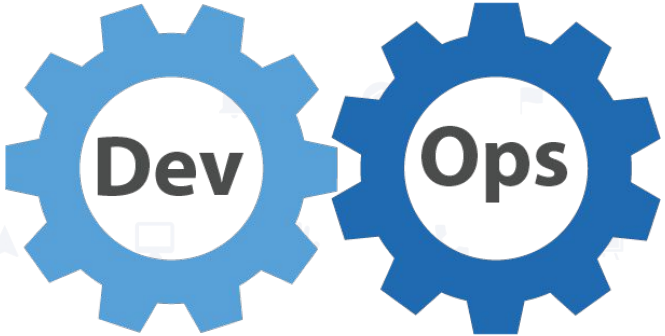
kubernetes

Pod, ReplicaSet, Deploy.



Спикер

Илия Карин

DevOps

iliyakarin.it@gmail.com

in/iliya-karin

- 9 лет в ИТ
- DevOps в Fintech
- DevOps/SRE в Nokia
- Сис. админ в Газпром

Репрезентация Docker

- Отдельно взятые контейнеры
- Легкая погрузка и доставка к клиенту
- Удобно, просто и понятно

Репрезентация Docker-Compose

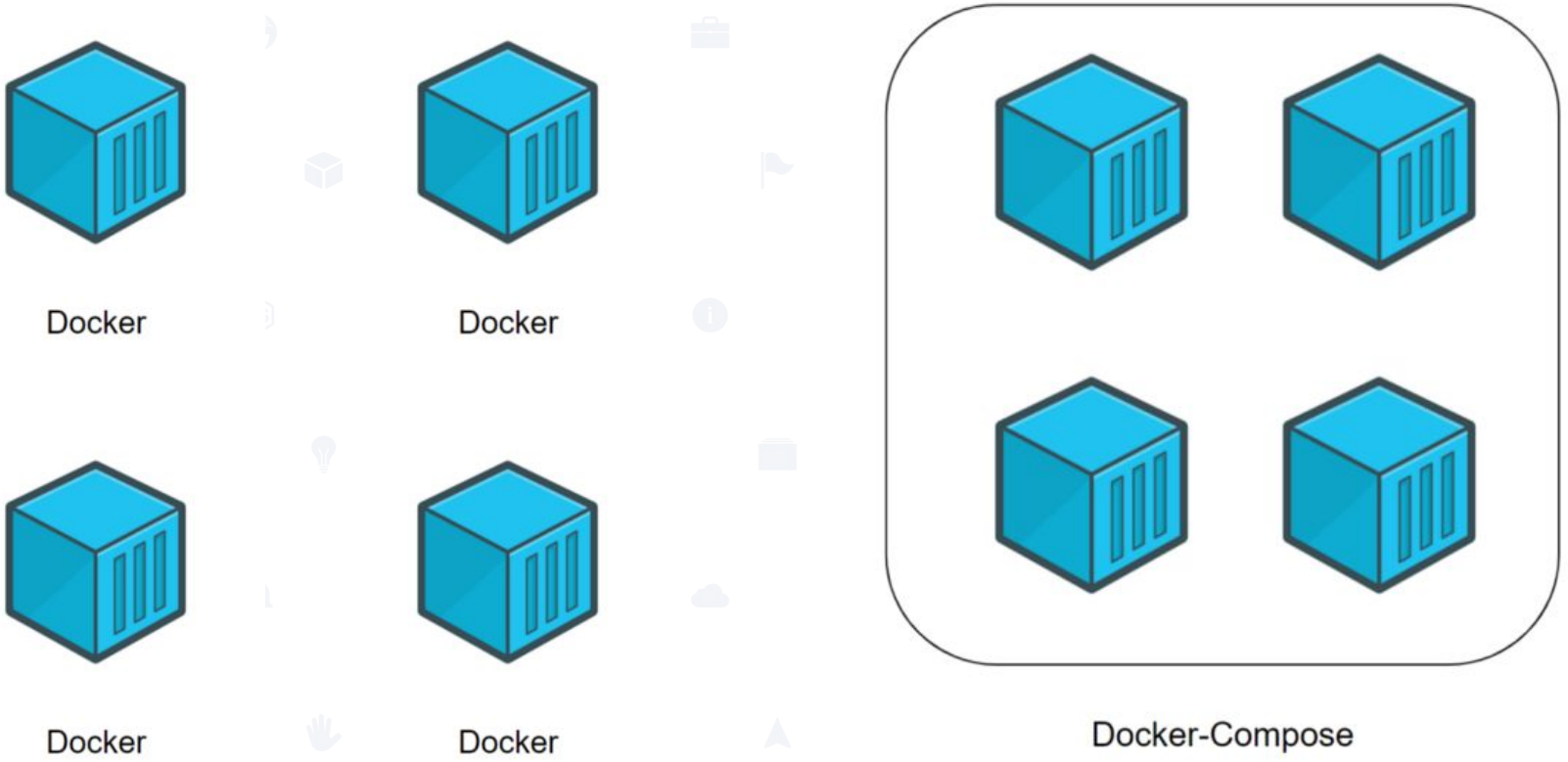
- Связанные группы контейнеров
- Легкая погрузка и доставка к клиенту
- Удобно, просто и понятно

Docker-Compose

- Docker - для управления отдельными контейнерами, из которых состоит приложение.

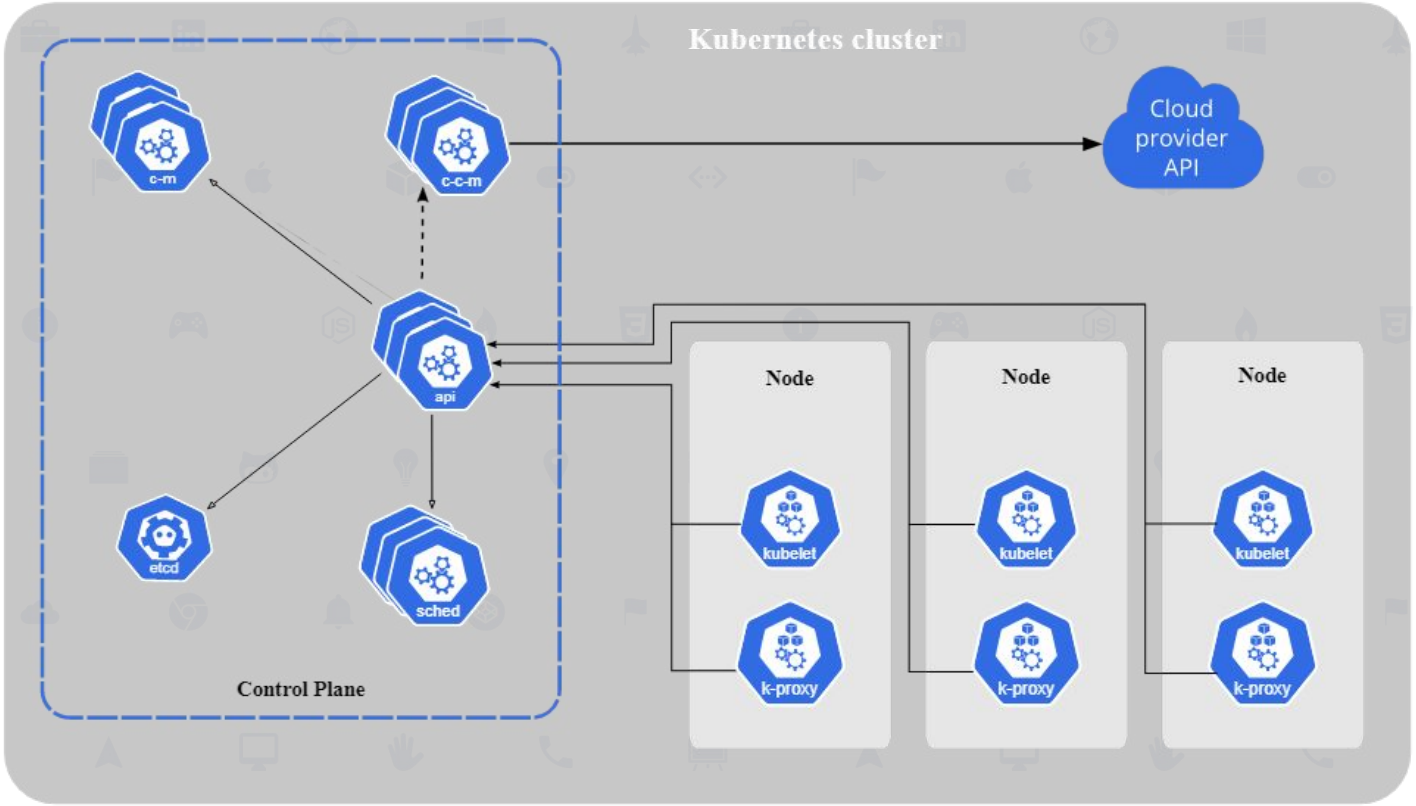
- Docker Compose - для одновременного управления несколькими контейнерами, входящими в состав приложения.

Docker-Compose



Репрезентация kubernetes

Архитектура Kubernetes



Зачем вам не нужен kubernetes

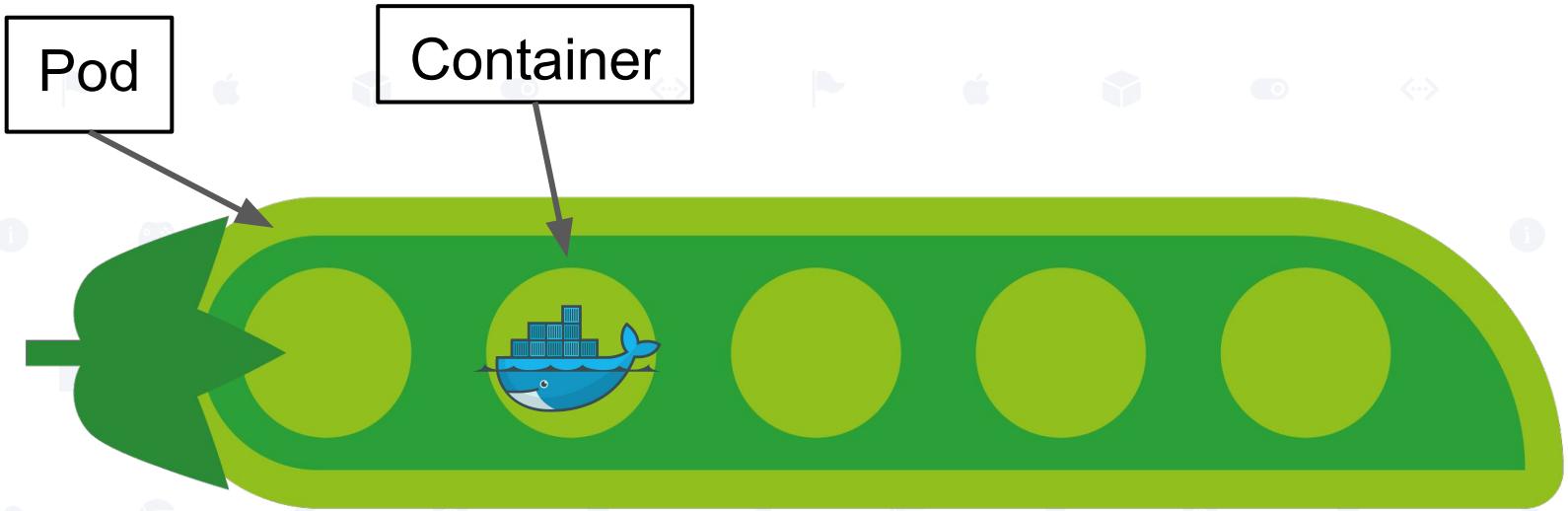
- Просто потому что. У других есть и мы хотим.
- Менеджмент где то слышал что в kubernetes все само станет хорошо.
- У вас есть легаси монолит и вы хотите решить проблемы ПО инфраструктурой.

Что такое Pod?

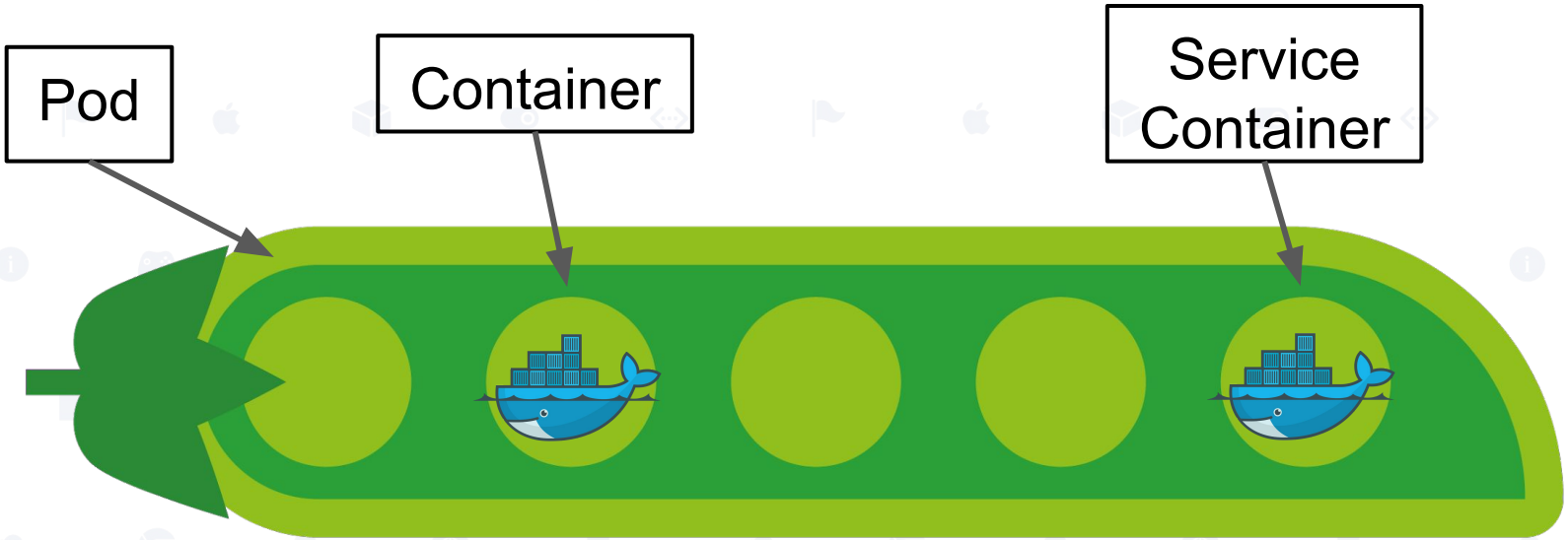
- Группа из одного или более контейнеров
- Единые правила запуска для всей группы
- Общие ресурсы сети и хранения данных
- Как будто отдельный хост

Kubernetes POD

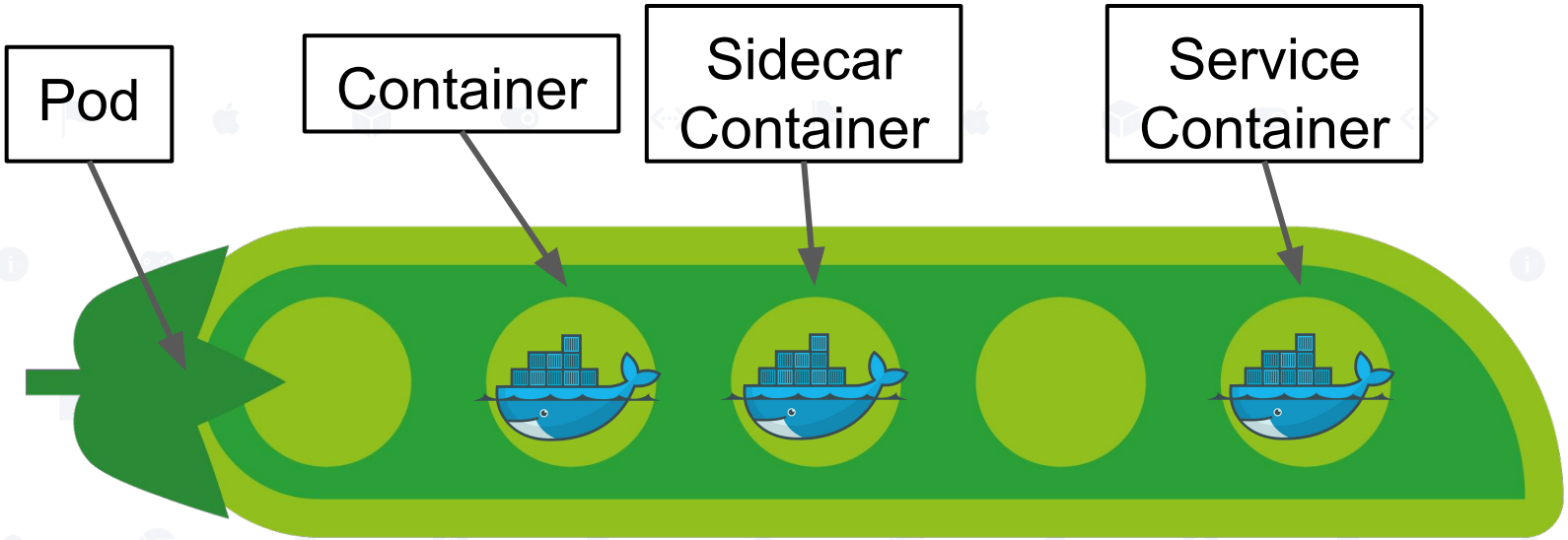
Kubernetes pod



Kubernetes pod



Kubernetes pod



Kubernetes pod

- Pod для k8s является неделимой единицей.
- В Pod может быть несколько контейнеров.
- В Pod помимо вашего контейнера всегда есть service container, обеспечивающий сетевое взаимодействие.
- Ваш вспомогательный (второй, третий и т.д.) контейнер внутри Pod называется sidecar container.

Создание Kubernetes pod

```
---
apiVersion: v1
kind: Pod
metadata:
  name: my-nginx-pod
spec:
  containers:
  - image: nginx:1.12
    name: nginx
    ports:
    - containerPort: 80
```

- **apiVersion** - версия api к которой обращаемся
- **kind** - тип объекта который создаем
- **metadata** - раздел метаданных
  - **name** - имя объекта
- **spec** - раздел спецификации объекта
  - **containers** - описание объекта контейнера
  - **image** - образ из которого создаем контейнер
  - **ports** - описание сетевых портов
    - **containerPort** - какой порт контейнера открываем

Создание Kubernetes pod

Создадим Pod:

```
# kubectl apply -f pod.yml
```

Вывод:

```
pod/my-pod created
```

Проверка работает ли Pod:

```
# kubectl get po
NAME    READY   STATUS    RESTARTS   AGE
my-pod   1/1     Running   0          51s
```

Создание еще одного Kubernetes pod

Меняем имя Pod’a в файле:

```
# kubectl apply -f pod.yml
pod/my-nginx-pod-1 created
```

Проверяем появился ли Pod:

```
# kubectl get po
NAME           READY   STATUS    RESTARTS   AGE
my-nginx-pod-1   1/1     Running   0          6s
my-pod           1/1     Running   0          3m36s
```

Редактирование манифеста Kubernetes pod

```
# kubectl edit pod my-pod
```

Меняем версию nginx на 1.13
Сохраняемся и выходим

Проверяем что pod перезапустился с новыми параметрами:

```
# kubectl get po
NAME             READY   STATUS    RESTARTS   AGE
my-nginx-pod-1   1/1     Running   0          4m34s
my-pod           1/1     Running   1          8m4s
```

Редактирование манифеста Kubernetes pod методом patch

```
# kubectl patch pod my-nginx-pod-1 -p
'{"spec":{"containers":[{"name":"nginx","image":"nginx:1.11"}]}}'
```

Меняем версию nginx на 1.11
Видимо сообщение об успехе:

```
pod/my-nginx-pod-1 patched
```

Проверяем что pod перезапустился с новыми параметрами:

```
# kubectl describe pod my-nginx-pod-1 | grep image
```

```
Normal  Pulled    9m9s    kubelet    Container image "nginx:1.12" already present on machine
Normal  Pulling   2m57s   kubelet    Pulling image "nginx:1.11"
Normal  Pulled    2m36s   kubelet    Successfully pulled image "nginx:1.11" in 20.996876795s
```

Проверка Kubernetes pod

```
# kubectl describe pod my-pod
```

```
Containers:
  nginx:
    Container ID:   docker://85b397437ced5b1b1bb00719db1afbe377342521493b02d72793911689be5bc6
    Image:          nginx:1.13

Events:
  Type    Reason     Age                  From               Message
  ----    ------     ----                 ----               -------
  Normal  Scheduled  41m                  default-scheduler  Successfully assigned default/my-pod to minikube.local
  Normal  Pulling    41m                  kubelet            Pulling image "nginx:1.12"
  Normal  Pulled     41m                  kubelet            Successfully pulled image "nginx:1.12" in 9.073318671s
  Normal  Killing    34m                  kubelet            Container nginx definition changed, will be restarted
  Normal  Pulling    34m                  kubelet            Pulling image "nginx:1.13"
  Normal  Created    33m (x2 over 41m)    kubelet            Created container nginx
  Normal  Started    33m (x2 over 41m)    kubelet            Started container nginx
```

Удаление Kubernetes pod по манифесту

```
# kubectl get po
NAME            READY   STATUS    RESTARTS   AGE
my-nginx-pod-1   1/1     Running   0          51m
my-pod           1/1     Running   1          55m
```

```
# kubectl delete -f pod.yml
pod "my-nginx-pod-1" deleted
```

```
# kubectl get po
NAME    READY   STATUS    RESTARTS   AGE
my-pod   1/1     Running   1          60m
```

Удаление всех Kubernetes pod по манифесту

```
# kubectl get po
NAME              READY   STATUS    RESTARTS   AGE
my-pod              1/1     Running   1          55m
```

```
# kubectl delete pod --all
pod "my-pod" deleted
```

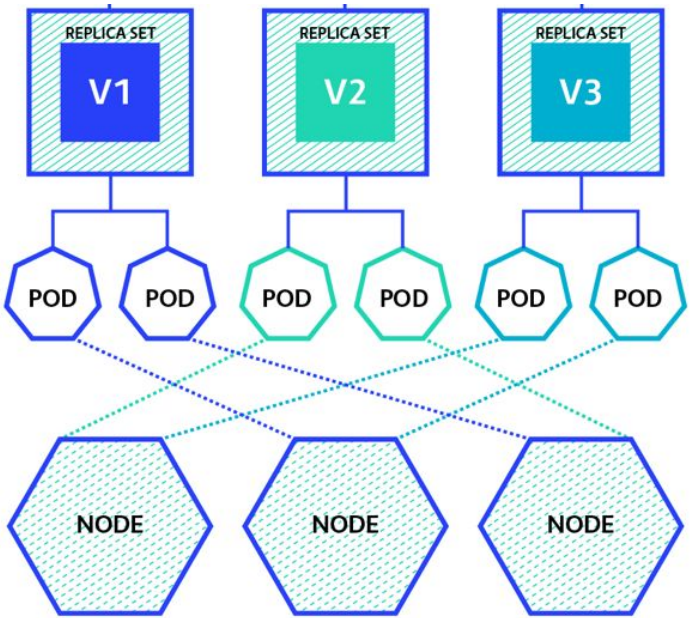
```
# kubectl get po
No resources found in default namespace.
```

Обсуждение Pod’s

- Как вы думаете почему появилось подобное дробление на Pod?
- Какие примеры использования вы можете представить?
- Какие минусы и плюсы на ваш взгляд несет подобная архитектура?

Replica Set

Replica Set



- Масштабирует Pod’ы
- Контролирует количество одновременно запущенных Pod’ов

Создание Kubernetes ReplicaSet

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: my-replicaset
spec:
  replicas: 2
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - image: nginx:1.12
        name: nginx
        ports:
        - containerPort: 80
```

- **spec** - раздел спецификации объекта
  - **Replicas** - количество Pod’ов
  - **Selector**
    - **matchLabels**
      - **app -** за каким префиксом следить
  - **Template -** шаблон аналогично файлу Pod’а
    - **Metadata**
      - **Labels**
        - **app -** префикс имени

Проверим Pods созданные с помощью Replica Set

Статус Pods:

```
# kubectl get pods
NAME                 READY   STATUS    RESTARTS   AGE
my-replicaset-8rnq6  1/1     Running   0          11m
my-replicaset-wnk6r  1/1     Running   0          11m
```

Статус Replica Set:

```
# kubectl get rs
NAME           DESIRED   CURRENT   READY   AGE
my-replicaset  2         2         2       16m
```

Удалим один Pod созданный с помощью RS

Удаляем Pod:

```
# kubectl delete pod my-replicaset-wnk6r
pod "my-replicaset-wnk6r" deleted
```

И сразу проверяем pods:

```
# kubectl get pods
NAME                READY   STATUS    RESTARTS   AGE
my-nginx-pod-1      1/1     Running   2          23h
my-replicaset-8rnq6 1/1     Running   1          22h
my-replicaset-t5bq9 1/1     Running   0          8s
```

Изменим версию image для контейнера в Pod созданный с помощью RS

Меняем версию image:

```
# kubectl set image replicaset my-replicaset nginx=nginx:1.13
```

Вывод:
```
replicaset.apps/my-replicaset image updated
```

И сразу проверяем rs:
```
# kubectl get rs my-replicaset -o yaml | grep -A3 -B3 image
    spec:
      containers:
      - image: nginx:1.13
```

Изменим версию image для контейнера в Pod созданный с помощью RS

Но если проверить Pods то там будет еще старая версия:

```
# kubectl describe my-replicaset-nqs4r
```

Вывод:

```
...
    Image:          nginx:1.12
...
```

С новым image будут созданы только новые контейнеры.
Для форсированного изменения версии image можно удалить поды чтобы RS их перезапустила.

Изменим количество Pods с помощью RS.

Изменим количество подов на 4:
`# kubectl scale replicaset my-replicaset --replicas 4`

Вывод:
`replicaset.apps/my-replicaset scaled`
Проверим: `# kubectl get po`

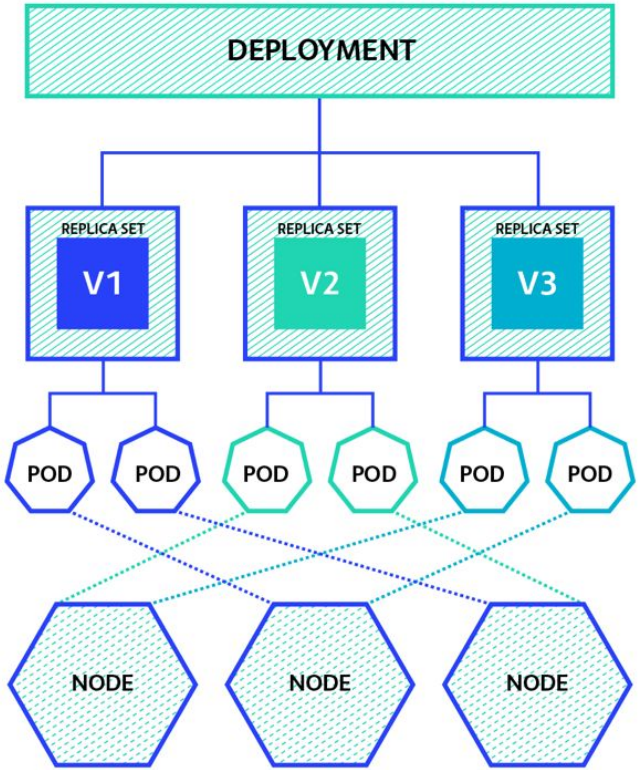
И в обратную сторону:
`# kubectl scale replicaset my-replicaset --replicas 1`
Проверим: `# kubectl get po`

ReplicaSet

- Как вы думаете через что RS взаимодействует с k8s?
- Зачем нужен ReplicaSet?
- На ваш взгляд: где может не хватить функционала RS?

Deployment

Deployment



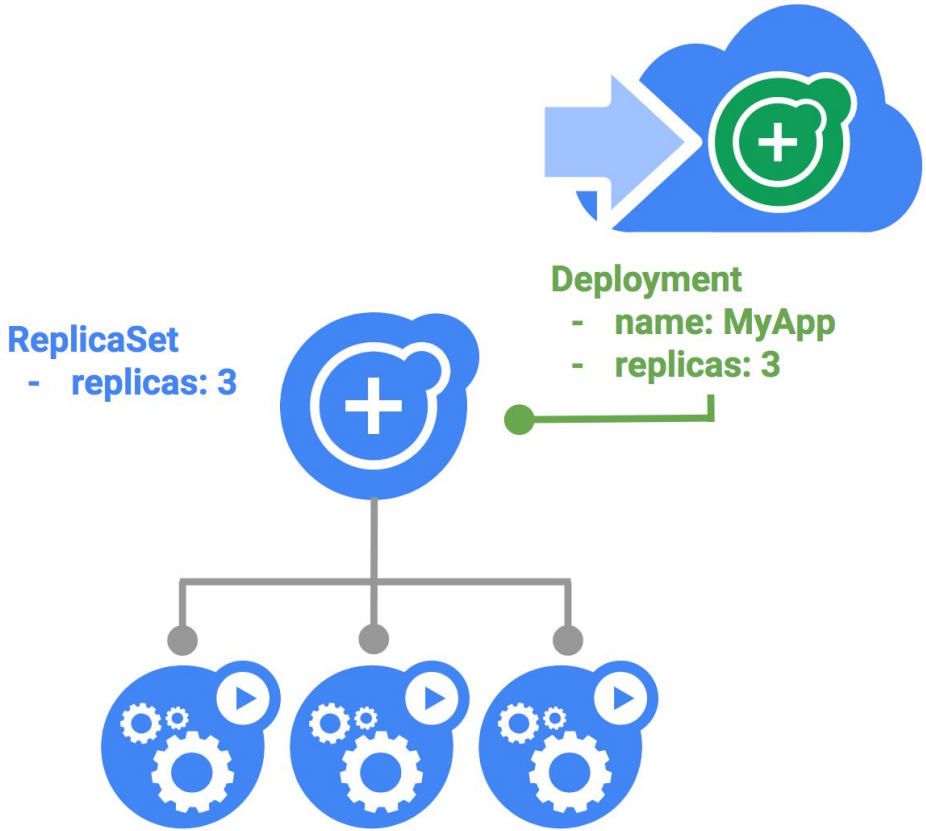
- Контролирует количество Replica Set
- Контролирует версии Replica Set и Pods в них
- В коммерческой эксплуатации используются именно Deployment

Создание Kubernetes deployment

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: my-app
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - image: nginx:1.12
        name: nginx
        ports:
        - containerPort: 80
...
```

- **Strategy -** стратегия обновления
  - **rollingUpdate - раздел настройки опций**
    - **maxSurge -** максимальное превышение количества реплик во время апдейта (можно в %)
    - **maxUnavailable -** максимум реплик одновременно апдейтить (можно в %)
  - **Type -** как обновлять приложение
    RollingUpdate - мягкий способ
    Recreate - жесткий

Kubernetes deployment



- Создает сам Deployment
- Создает ReplicaSet
- Создает Pods
- Следит за ними

Изменение kubernetes deployment

- Можно edit:
  `# kubectl edit deployment my-deployment`
- Можно set:
  `# kubectl set image deployment my-deployment nginx=nginx:1.10`
- Нужно через файл!
  `# kubectl apply -f deployment.yml`

Kubernetes deployment RS после apply

Посмотрим на RS:

```
# kubectl get rs
NAME                          DESIRED   CURRENT   READY   AGE
my-deployment-6c4d5f6b74      6         6         6       2m38s
my-deployment-779cc4b89b      0         0         0       37m
```

Старая RS:

```
# kubectl describe rs my-deployment-779cc4b89b | grep -i image
    Image:        nginx:1.12
```

Новыя RS:

```
# kubectl describe rs my-deployment-6c4d5f6b74  | grep -i image
    Image:        nginx:1.10
```

Kubernetes deployment rollout

- Старые RS остаются для возможности отката (по умолчанию 10 версий).

Откат:

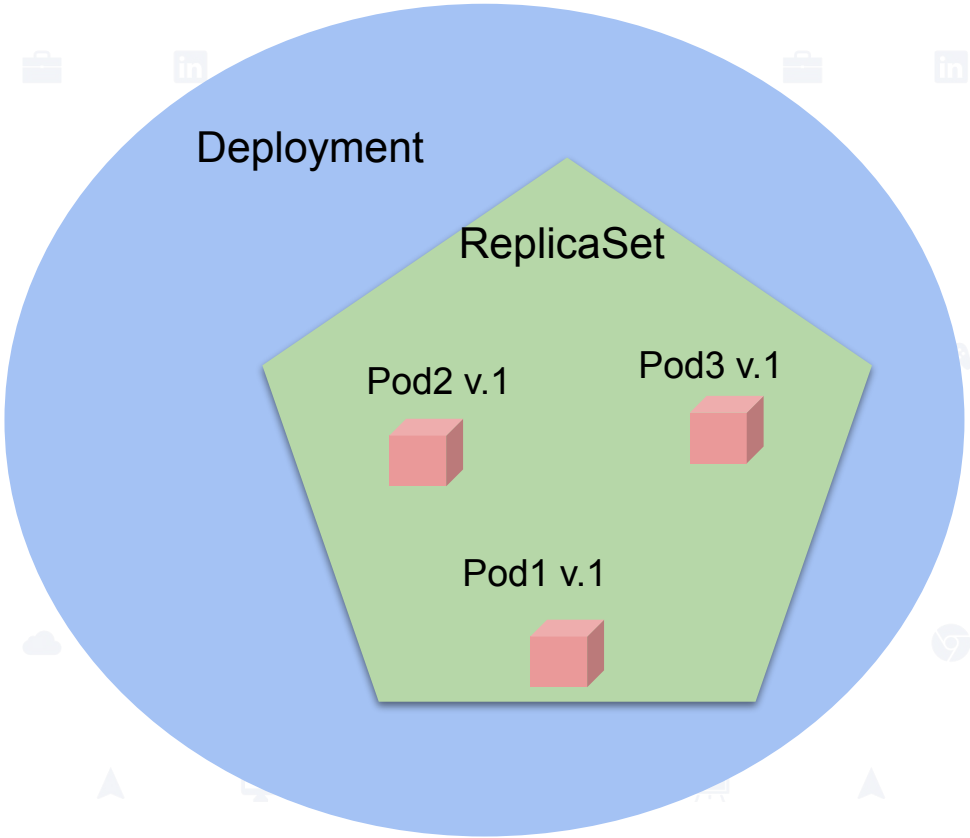
```
# kubectl rollout undo deployment my-deployment --to-revision=0
```

Вывод:

```
deployment.apps/my-deployment rolled back
```

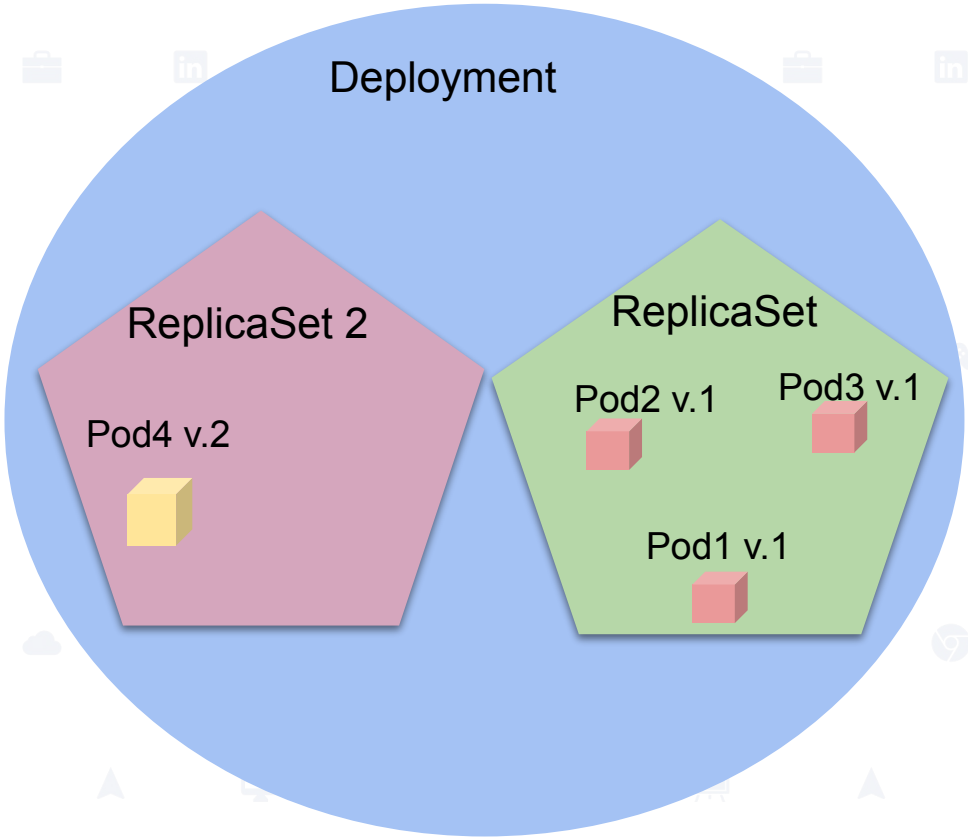
--to-revision=0 - прошлая версия deployment
--to-revision=1 - позапрошлая версия deployment

Deployment создает RS которые создают Pods



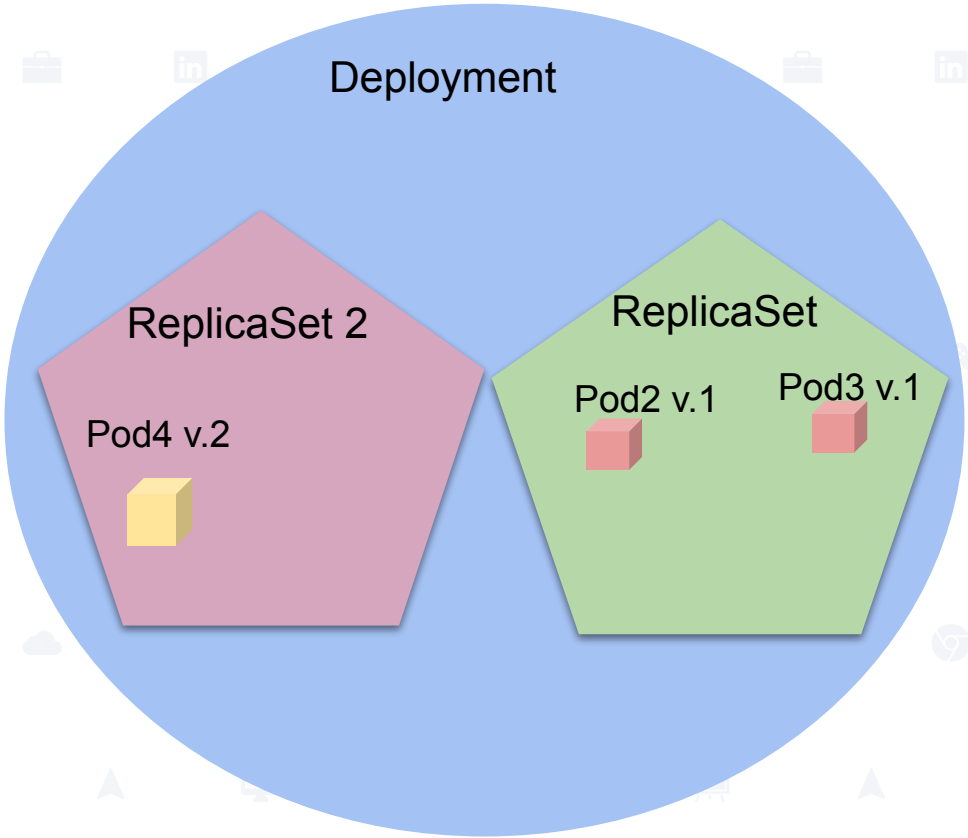
- Deployment с текущей версией подов

Deployment создает другой ReplicaSet



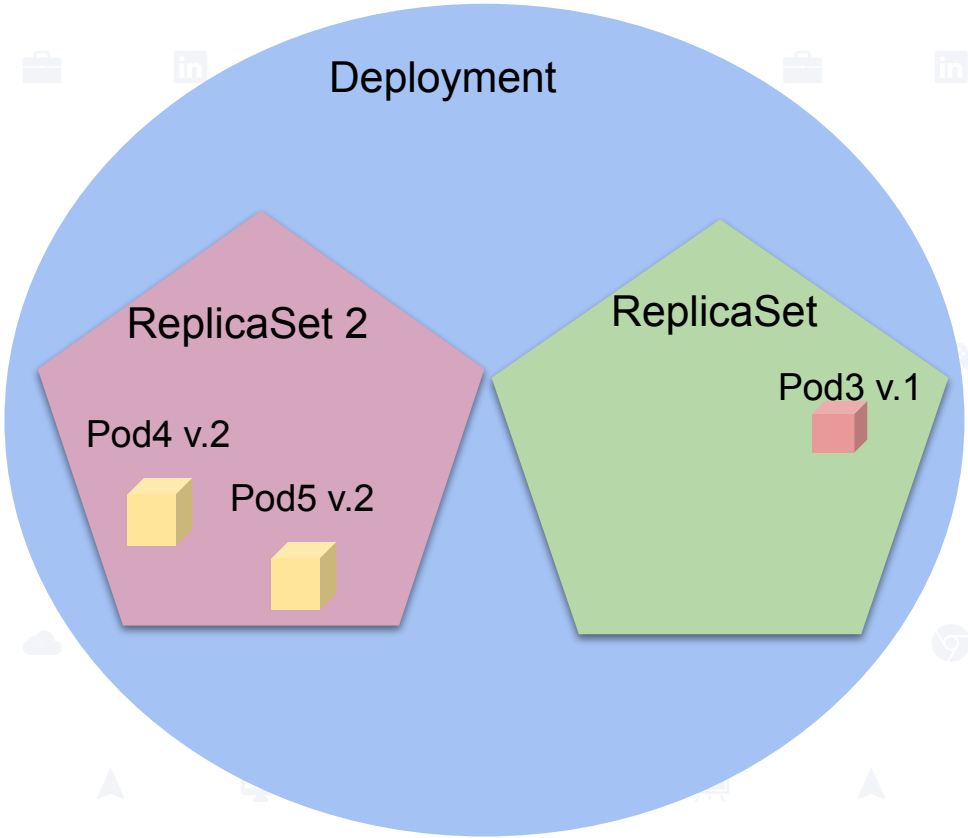
- Deployment создает другой RS который создает Pods других версий, в рамках того же Deployment

Deployment терминирует старые Pods



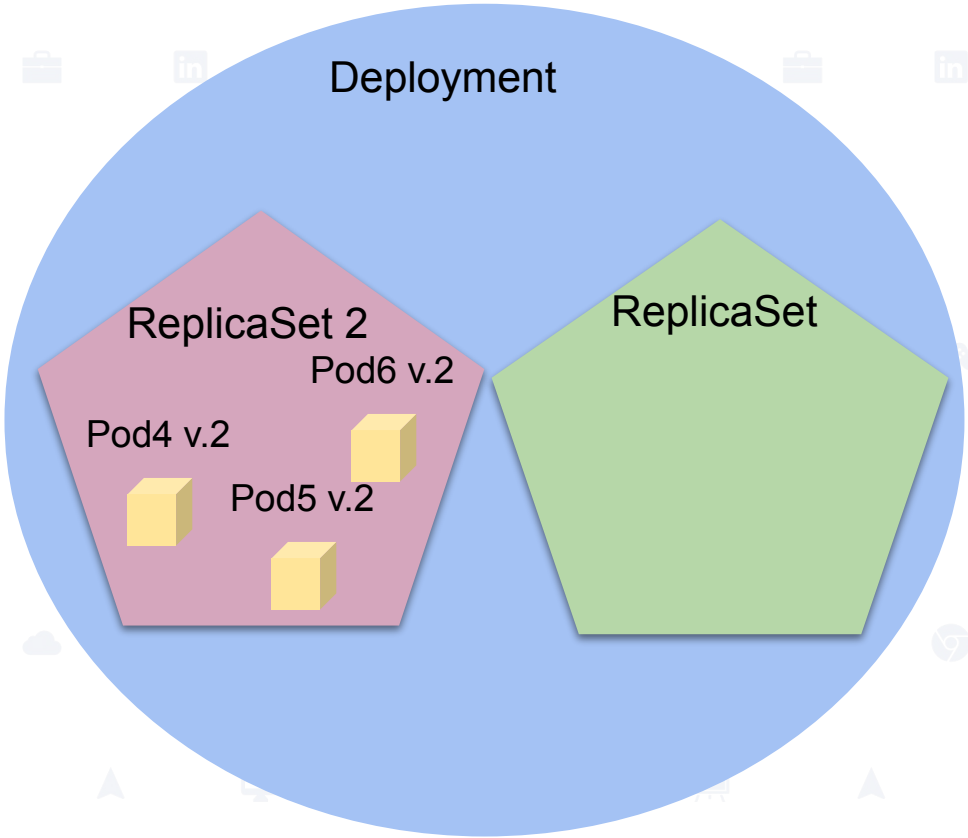
- Deployment начинает терминировать Pods старой версии в первом RS

Deployment терминирует старые Pods



- Deployment продолжает терминировать Pods старой версии в первом RS

Deployment терминирует старые Pods



- Deployment продолжает терминировать Pods старой версии в первом RS, до тех пор пока не будет достигнуто заданное значение в обновленном Deployment в RS2

Probes - Проверки

- Liveness Probe
  - Следит за приложением все время пока оно работает
  - Рестартит приложение в случае фейла проверки
- Readiness Probe
  - Проверяет готово ли приложение принимать пакеты
  - Удаляет приложение из балансировки если приложение не принимает трафик

Создание Kubernetes deployment с Probes

```
apiVersion: apps/v1
kind: Deployment
……..
    spec:
      containers:
      - image: nginx:1.12
        name: nginx
        ports:
        - containerPort: 80
        readinessProbe:
          failureThreshold: 3
          httpGet:
            path: /
            port: 80
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 1
        livenessProbe:
          failureThreshold: 3
          httpGet:
            path: /
            port: 80
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 1
          initialDelaySeconds: 10
```

- **readinessProbe -** тип пробы
  - **failureThreshold -** порог срабатывания действия (убрать из балансировщика)
  - **httpGet -** как проверять (условный curl Get)
    - Можно делать **exec** - выполнение проверки запуском команды внутри контейнера
    - И третий вариант проверка доступности порта **tcpSocket**
  - **periodSeconds -** частота провеки
  - **successThreshold -** сброс счетчика порога срабатывания если N успешных
  - **timeoutSeconds -** таймаут ответа
- **livenessProbe**
  - **initialDelaySeconds -** время через которое начинать проверку после запуска пода

exec Probes

```
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/busybox
    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy
    livenessProbe:
      exec:
        command:
        - cat
        - /tmp/healthy
      initialDelaySeconds: 5
      periodSeconds: 5
```

- **args -** в аргументах команда которая создаст файл
- **exec -** тип пробы
  - **command -** bash команда
    - Прочитать файл /tmp/healthy
  - **initialDelaySeconds -** время через которое начинать проверку после запуска пода
  - **periodSeconds -** частота провеки

tcpSocket Probes

```
spec:
  containers:
  - name: goproxy
    image: k8s.gcr.io/goproxy:0.1
    ports:
    - containerPort: 8080
    readinessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 5
      periodSeconds: 10
    livenessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 15
      periodSeconds: 20
```

- **tcpSocket -** тип пробы
- **Port -** какой порт проверяем
- **initialDelaySeconds -** время через которое начинать проверку после запуска пода
- **periodSeconds -** частота провеки

Resources - ресурсы

- Limits
  - Максимальное количество ресурсов доступных для одного Pod
  - Больше указанного не выдавать
- Requests
  - Резервируемое количество ресурсов для Pod
  - Жесткая резервация, без переподписки с другими Pod

Зачем нужно управление Resources?

- Kubernetes не знает сколько реальных ресурсов есть на сервере/виртуальной машине.

- Для Kubernetes ресурсов на сервере столько сколько мы указали в Requests.

Как описываются Resources?

- В лимитах CPU указывается в милли CPU либо % от 1 ядра, либо в целых ядрах.
- Указывается как: mCPU
- 1 CPU = 1000 mCPU, или 1 CPU = 100%, или 1 CPU = 1
- Pod не может потреблять больше CPU чем указано в лимитах, k8s ограничит использование CPU.
- Если закончится реальная RAM, то ПО будет убито OOM киллером.
- Работает все на основе механизма ядра Linux: cgroups
- Задержка наблюдаемое между запуском пода 0/1 и 1/1 это ожидание подтверждения readinessProbe
- Данные проверки защищают от запуска новую версию ПО (Pod) при обновлении с помощью RS, во избежание получения отказа в предоставлении сервиса.

Разбираем describe deployment

```
# kubectl describe deployment  my-deployment
```

```
Pod Template:
  Labels:  app=my-nginx
  Containers:
   nginx:
    Image:      nginx:1.12
    Port:       80/TCP
    Host Port:  0/TCP
    Limits:
      cpu:     100m
      memory:  100Mi
    Requests:
      cpu:        50m
      memory:     100Mi
    Liveness:     http-get http://:80/ delay=10s timeout=1s period=10s
#success=1 #failure=3
    Readiness:    http-get http://:80/ delay=0s timeout=1s period=10s
#success=1 #failure=3
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
```

- Видим все ранее рассмотренные ресурсы.

Экспериментируем с Requests

- Выдаем больше ядер CPU для Pod, больше чем есть на физическом сервере (виртуальной машине)
- Наблюдаем в результат в kubectl describe deployment my-deployment

```
# kubectl describe deployment my-deployment
Replicas:               2 desired | 2 updated | 3 total | 1 available | 2 unavailable
```

```
# kubectl get pods
NAME                             READY   STATUS    RESTARTS   AGE
my-deployment-54d56f4555-jvkbx   0/1     Pending   0          57s
my-deployment-54d56f4555-zlphj   0/1     Pending   0          57s
my-deployment-79754bb7c-6zfh2    1/1     Running   0          2m3s
```

- Если в Requests указать больше чем в Limits то умный k8s скажем нам об этом.

```
# kubectl apply -f deployments-probes-with-error.yml
```

Вывод:

```
The Deployment "my-deployment" is invalid: spec.template.spec.containers[0].resources.requests: Invalid value: "400m": must be less than or equal to cpu limit
```

Спасибо за внимание!

Илия Карин

iliyakarin.it@gmail.com
in/iliya-karin