# KUB 27: Горизонтальное масштабирование

## Описание:

Очень частая задача, которая возникает в любой инфраструктуре, — это автоматическое масштабирование при увеличении нагрузки. В Kubernetes для этих целей есть несколько ресурсов, но вначале нам необходимо будет разобраться с существующими вариантами масштабирования:

1. Горизонтальное масштабирование — Kubernetes будет прибавлять дополнительные поды в случае, если текущая метрика не соответствует заявленной.
2. Вертикальное масштабирование — Kubernetes будет прибавлять ресурсы подам в случае, если текущая метрика не соответствует заявленной.
3. Масштабирование кластера — добавление дополнительных узлов в кластер в случае, если pod не может быть запущен из-за Insufficient Memory / CPU.

Масштабирование кластера

Автоматическое масштабирование кластера целиком управляется облачным провайдером. Поэтому, если вы будете использовать кластер Kubernetes в облаке, вам всего лишь достаточно сделать конкретную группу узлов автоматически масштабируемой и все остальное возьмет на себя cloud controller manager.

Если представить себе ситуацию, что вы хотите запустить pod, но при этом запустить его негде из-за заданных resourses (их больше, чем есть свободных), то дальше cloud controller автоматически добавит новую ноду в кластер. Она стартанет, подключится к кластеру и новый под запустится на ней. Давайте посмотрим вывод лога для примера выше:

```
 Type     Reason            Age                From
Message
  ----     ------            ----               ----
-------
  Warning  FailedScheduling  28s (x2 over 28s)
default-scheduler   0/3 nodes are available: 3 Insufficient
memory.
  Normal   TriggeredScaleUp  19s
cluster-autoscaler  pod triggered scale-up:
[{cat2bcc8jfcqipcpfem5 1->2 (max: 10)}]
```

Оно означает, что pod не может быть запущен из-за недостаточного количества памяти и cluster-autoscaler запустил увеличение группы узлов с одной машины до двух.

Горизонтальное масштабирование

Для горизонтального масштабирования вам придется создать ресурс, который будет увеличивать количество реплик в деплойменте. Давайте посмотрим на такой пример:

```
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: test-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: nginx-dp
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageUtilization: 80
```

В данном случае мы создаем HorizontalPodAutoscaler, который следит за деплойментом nginx-dp и увеличивает количество подов, если утилизация CPU пода достигает 80%. HorizontalPodAutoscaler берет метрики из нескольких API — metrics.k8s.io, custom.metrics.k8s.io, external.metrics.k8s.io. Metrics.k8s.io управляет metrics server, который запущен в кластере. Он собирает базовые метрики — cpu & memory usage. Обычно в облачных окружениях он устанавливается по умолчанию. Но в случае self hosted варианта (если вы не включили его в kubespray) он установлен не будет. Для его установки можно воспользоваться простым манифестом:

```
$ kubectl apply -f
https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
```

Иногда бывает так, что metrics-server не взлетает. Такое возможно из-за того, что используются неподписанные сертификаты. В этом случае нужно скачать локально файл components.yaml и выставить дополнительный параметр --kubelet-insecure-tls в args, например вот так:

```
$ wget
https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml

# открываем на редактирование и вставляем нужный параметр в args:
$ vi components.yaml
```

```
...
args:
  --kubelet-insecure-tls
  --kubelet-preferred-address-types=InternalIP
  --cert-dir=/tmp
  --secure-port=4443
...
```

После установки вы можете проверить что kubernetes api расширился и у него появилась группа metrics.k8s.io (kubectl api-resources). Мы уже смотрели на метрики, которые он выдает, в задании про Kubernetes API, но давайте взглянем на них еще раз:

```
$ kubectl get podmetrics nginx-6dffdfcc57-cbndb -o yaml
apiVersion: metrics.k8s.io/v1beta1
containers:
- name: nginx
  usage:
    cpu: "0"
    memory: 2744Ki
kind: PodMetrics
metadata:
  creationTimestamp: "2020-12-04T20:49:58Z"
  name: nginx-6dffdfcc57-cbndb
  namespace: default
  selfLink:
/apis/metrics.k8s.io/v1beta1/namespaces/default/pods/nginx-6dffd
fcc57-cbndb
timestamp: "2020-12-04T20:49:39Z"
window: 30s
```

Как вы видите — у нас есть cpu, который мы использовали при настройке HPA. Чуть позже мы расширим количество метрик, которые сможет использовать HorizontalPodAutoscaler. Обратите внимание: без указанных requests и limits в манифесте deployment'a, HPA работать не будет.

# Полезные ссылки:

- HPA

# Задание:

1. Установите metrics server в Kubernetes кластер.

2. Создайте дейплоймент alpine-dp в namespace default, который внутри будет запускать команду dd для копирования данных из /dev/zero в /dev/null.
3. Создайте HorizontalPodAutoScaler hpa-alpine в namespace default, который будет увеличивать количество реплик этого деплоймента при достижении ими cpu usage 20% (обратите внимание, иногда для полноценного сбора статистики нужно подождать до 20 минут).
4. Отправьте задание на проверку.