KUB 06: Основы работы с Kubectl

Описание:

kubectl — это консольная утилита, необходимая для взаимодействия с кластером Kubernetes. Более подробное описание возможностей kubectl вы можете отыскать в документации kubectl.
Kubectl Basics
Основные команды:

- get — получает ресурс из API-сервера, по умолчанию — в сокращенном формате, -o задает формат;
- describe — описывает ресурс в выбранном формате;
- create — создает ресурс из файла или с указанием типа и параметров;
- delete — удаляет ресурс по выбору или его имени;
- edit — открывает редактор и загружает в него файл описания ресурса, при сохранении отправляет измененный ресурс API-серверу;
- explain — показывает документацию по ресурсу;
- patch — вносит изменения в поля ресурса, используя стратегию слияния патча;
- proxy — запускает локальный прокси-сервер до API-сервера Kubernetes;
- replace — заменяет ресурс новой версией;
- logs — выдает логи ресурса;
- expose — создает сервис для pod, rs, rc;
- port-forward — прокидывает локальный порт на порт внутри контейнера pod;
- run — запускает pod из указанного образа (image);
- scale — изменяет spec.replicas;
- rolling-update — выполняет обновление, клиентом будет kubectl на хосте.

Давайте попробуем посмотреть некоторые ресурсы в kubernetes:

```
$ kubectl get namespaces
NAME              STATUS   AGE
default           Active   77m
kube-node-lease   Active   77m
kube-public       Active   77m
kube-system       Active   77m
$ kubectl get nodes
NAME                         STATUS   ROLES    AGE   VERSION
cl128p75cket5bspv91v-oles    Ready    <none>   74m   v1.18.9
cl128p75cket5bspv91v-oxuq    Ready    <none>   74m   v1.18.9
$
```

Можно посмотреть на ноды детальнее:

```
$ kubectl get nodes -o wide
NAME                        STATUS   ROLES    AGE   VERSION
INTERNAL-IP   EXTERNAL-IP      OS-IMAGE
KERNEL-VERSION     CONTAINER-RUNTIME
cl128p75cket5bspv91v-oles   Ready    <none>   74m   v1.18.9
10.1.0.26     84.201.137.28    Ubuntu 18.04.4 LTS
5.4.0-52-generic   docker://19.3.13
cl128p75cket5bspv91v-oxuq   Ready    <none>   74m   v1.18.9
10.0.0.23     130.193.50.214   Ubuntu 18.04.4 LTS
5.4.0-52-generic   docker://19.3.13
```

Или вообще вывести всю информацию о ноде:

```
$ kubectl get node cl128p75cket5bspv91v-oles -o  yaml
apiVersion: v1
kind: Node
metadata:
  annotations:
    csi.volume.kubernetes.io/nodeid:
'{"disk-csi-driver.mks.ycloud.io":"epdo8qjcqibah889f8un","io.ycl
oud.mks.disk-csi-driver":"epdo8qjcqibah889f8un"}'
    node.alpha.kubernetes.io/ttl: "0"
    volumes.kubernetes.io/controller-managed-attach-detach:
"true"
  creationTimestamp: "2020-12-02T12:23:11Z"
  labels:
    beta.kubernetes.io/arch: amd64
    beta.kubernetes.io/instance-type: standard-v2
    beta.kubernetes.io/os: linux
    failure-domain.beta.kubernetes.io/zone: ru-central1-b
    kubernetes.io/arch: amd64
    kubernetes.io/hostname: cl128p75cket5bspv91v-oles
    kubernetes.io/os: linux
    node.kubernetes.io/kube-proxy-ds-ready: "true"
    node.kubernetes.io/masq-agent-ds-ready: "true"
    node.kubernetes.io/node-problem-detector-ds-ready: "true"
    status: done
    yandex.cloud/node-group-id: cath1d7ouru0i73ci893
    yandex.cloud/pci-topology: k8s
    yandex.cloud/preemptible: "false"
  managedFields:
  - apiVersion: v1
```

```
  ... skipped ...
```

Templating

Также kubectl поддерживает формат шаблонов golang templates. Подробнее о go templating можно изучить здесь

Давайте попробуем вывести все labels у каждой ноды в кластере:

```
$ kubectl get nodes -o go-template --template='{{ range .items }}{{ printf "%s\n" .metadata.name }}{{ range $key, $value := .metadata.labels }}{{ printf "  %s = %s\n" $key $value }}{{ end }}{{ end }}'

cl128p75cket5bspv91v-oles
  beta.kubernetes.io/arch = amd64
  beta.kubernetes.io/instance-type = standard-v2
  beta.kubernetes.io/os = linux
  failure-domain.beta.kubernetes.io/zone = ru-central1-b
  kubernetes.io/arch = amd64
  kubernetes.io/hostname = cl128p75cket5bspv91v-oles
  kubernetes.io/os = linux
  node.kubernetes.io/kube-proxy-ds-ready = true
  node.kubernetes.io/masq-agent-ds-ready = true
  node.kubernetes.io/node-problem-detector-ds-ready = true
  status = done
  yandex.cloud/node-group-id = cath1d7ouru0i73ci893
  yandex.cloud/pci-topology = k8s
  yandex.cloud/preemptible = false
cl128p75cket5bspv91v-oxuq
  beta.kubernetes.io/arch = amd64
  beta.kubernetes.io/instance-type = standard-v2
  beta.kubernetes.io/os = linux
  failure-domain.beta.kubernetes.io/zone = ru-central1-a
  kubernetes.io/arch = amd64
  kubernetes.io/hostname = cl128p75cket5bspv91v-oxuq
  kubernetes.io/os = linux
  node.kubernetes.io/kube-proxy-ds-ready = true
  node.kubernetes.io/masq-agent-ds-ready = true
  node.kubernetes.io/node-problem-detector-ds-ready = true
  status = done
  yandex.cloud/node-group-id = cath1d7ouru0i73ci893
  yandex.cloud/pci-topology = k8s
```

```
  yandex.cloud/preemptible = false
```

Еще несколько интересных операций:
Посмотреть образы контейнеров, запущенных в кластере.

```
kubectl get pods -n test -o go-template --template='{{range
.items}}{{range .spec.containers}}{{printf "%s\n"
.image}}{{end}}{{end}}'
```

Вывести для pod аннотации app.kubernetes.io/instance, у которых количество рестартов первого контейнера больше 100.

```
kubectl get pods --all-namespaces -o go-template
--template='{{range .items}}{{if gt (index
.status.containerStatuses 0).restartCount 100.0}}{{ printf
"%s\n" (index .metadata.labels
"app.kubernetes.io/instance")}}{{end}}{{end}}'
```

Kubectl Create & Update
Давайте сейчас попробуем создать простой под, который будет содержать один контейнер nginx:

```
$ kubectl run nginx --image nginx
pod/nginx created
$ kubectl get pods
NAME    READY   STATUS              RESTARTS   AGE
nginx   0/1     ContainerCreating   0          5s
$
```

Как вы можете увидеть — у нас создался под с именем nginx, но он пока еще не готов — его статус ContainerCreating. Через некоторое время он запустится, а пока мы можем посмотреть детальную информацию о нем:

```
$ kubectl describe pod nginx
Name:         nginx
Namespace:    default
Priority:     0
Node:         cl128p75cket5bspv91v-oles/10.1.0.26
Start Time:   Wed, 02 Dec 2020 16:47:01 +0300
Labels:       run=nginx
Annotations:  <none>
Status:       Running
IP:           10.112.128.5
IPs:
  IP:  10.112.128.5
Containers:
```

```
  nginx:
    Container ID:
docker://9be16dab7af091494a661204c150369e89d2c6773b48fb0a3baa8a3
6abd4184f
    Image:          nginx
    Image ID:
docker-pullable://nginx@sha256:6b1daa9462046581ac15be20277a7c754
76283f969cb3a61c8725ec38d3b01c3
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Wed, 02 Dec 2020 16:47:16 +0300
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from
default-token-qh5rv (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-qh5rv:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-qh5rv
    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  91s   default-scheduler
Successfully assigned default/nginx to cl128p75cket5bspv91v-oles
```

```
  Normal  Pulling    90s   kubelet, cl128p75cket5bspv91v-oles
Pulling image "nginx"
  Normal  Pulled     78s   kubelet, cl128p75cket5bspv91v-oles
Successfully pulled image "nginx"
  Normal  Created    76s   kubelet, cl128p75cket5bspv91v-oles
Created container nginx
  Normal  Started    76s   kubelet, cl128p75cket5bspv91v-oles
Started container nginx
```

Здесь мы можем посмотреть статус контейнера, его IP, на какой ноде он запустился, а также самое главное — список событий, которые относятся к этому поду. Например, какая нода была выбрана для его запуска, что начался процесс скачивания образа и так далее. При таком использовании команды run, kubernetes запустит внутри контейнера команду, которая была определена в dockerfile - cmd или entrypoint. Вы можете изменить ее при создании пода примерно таким образом:

```
$ kubectl run ubuntu --image ubuntu -- /bin/bash -c 'while true;
do sleep 300; done'
pod/ubuntu created
```

Таким образом у нас запустится образ ubuntu с командой /bin/bash -c 'while true; do sleep 300; done', которая по сути в бесконечном цикле будет засыпать на 300 секунд. И контейнер будет жить бесконечно.
А теперь давайте попробуем добавить лейбл нашему поду:

```
$ kubectl label pod nginx test=true
pod/nginx labeled
```

И уверимся, что лейбл создался:

```
$ kubectl get pod nginx -o go-template --template='{{ range
$key, $value := .metadata.labels }}{{ printf " %s = %s\n" $key
$value }}{{ end }}'
 run = nginx
 test = true
```

На самом деле, это правильная практика — использовать go-template для фильтрации вывода. Стоит привыкать сразу ее использовать, поскольку когда у вас будут запущены сотни подов, ориентироваться в них станет достаточно нелегко.
Kubectl Exec & Logs
Давайте теперь попробуем запустить внутри нашего пода команду id:

```
$ kubectl exec nginx -- id
uid=0(root) gid=0(root) groups=0(root)
```

Чтобы зайти внутрь пода, надо добавить ключик -it — interactive tty:

```
$ kubectl exec -it nginx -- sh
# id
uid=0(root) gid=0(root) groups=0(root)
# pwd
/
# ps
sh: 3: ps: not found
#
```

Можем вывести логи нашего пода:

```
$ kubectl logs nginx
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will
attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in
/docker-entrypoint.d/
/docker-entrypoint.sh: Launching
/docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: Getting the checksum of
/etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: Enabled listen on IPv6 in
/etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching
/docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Configuration complete; ready for start
up
```

В команде logs вы также можете использовать ключи --tail=100 и -f для указания количества последних строк, которые надо вывести, и ожидания новых строк. Прямо как с tail.

Давайте попробуем теперь удалить pod:

```
$ kubectl delete pod nginx
pod "nginx" deleted
```

Собственно, это основные команды для работы с kubectl — мы только что рассмотрели, как вы можете просматривать ресурсы, удалять их, просматривать логи и так далее. В будущих заданиях мы более детально рассмотрим все ресурсы, а также дополним наш аресенал дополнительными командами для работы с ними.

Kubectl Nodes Management

А сейчас хочется еще поговорить про maintenance нод. Для этого существуют команды cordon, drain и uncordon.

Давайте пометим наш узел как неназначаемый, чтобы на нем не могли запускаться поды:

```
$ kubectl cordon cl128p75cket5bspv91v-oles
```

```
node/cl128p75cket5bspv91v-oles cordoned
$ kubectl get nodes
NAME                        STATUS                     ROLES
AGE   VERSION
cl128p75cket5bspv91v-oles   Ready,SchedulingDisabled   <none>
96m   v1.18.9
cl128p75cket5bspv91v-oxuq   Ready                      <none>
96m   v1.18.9
$
```

Как видите — мы пометили узел SchedulingDisabled, но это опция будет доступна только для новых подов. Чтобы переместить текущие поды с этой ноды, необходимо сделать drain:

```
$ kubectl drain cl128p75cket5bspv91v-oles --ignore-daemonsets
node/cl128p75cket5bspv91v-oles already cordoned
WARNING: ignoring DaemonSet-managed Pods:
kube-system/ip-masq-agent-swxdw, kube-system/kube-proxy-rhtm2,
kube-system/npd-v0.8.0-hff54,
kube-system/yc-disk-csi-node-v2-76bkb
evicting pod kube-system/coredns-7c646474c9-kdvh6
evicting pod kube-system/kube-dns-autoscaler-7c867cbf8d-smz8s
pod/kube-dns-autoscaler-7c867cbf8d-smz8s evicted
pod/coredns-7c646474c9-kdvh6 evicted
node/cl128p75cket5bspv91v-oles evicted
vozerov@mba:~/work/rebrain/yc-kub $ kubectl get nodes
NAME                        STATUS                     ROLES
AGE    VERSION
cl128p75cket5bspv91v-oles   Ready,SchedulingDisabled   <none>
100m   v1.18.9
cl128p75cket5bspv91v-oxuq   Ready                      <none>
99m    v1.18.9
```

В данном случае мы заставили Kubernetes перетащить с ноды все системные поды. Прошу обратить внимание, что наш nginx не мог быть перенесен из-за того, что мы запустили его без вышестоящего ресурса — replica set, deployment, stateful set или daemonset, поэтому Kubernetes не знает, как с ним стоит работать — нужно ли переподключать ему диск или можно просто убить и запустить новый. Про эти ресурсы поговорим в дальнейшем.

Чтобы вернуть ноду в рабочее состояние, можем использовать команду uncordon:

```
$ kubectl uncordon cl128p75cket5bspv91v-oles
node/cl128p75cket5bspv91v-oles uncordoned
$ kubectl get nodes
```

```
NAME                         STATUS   ROLES    AGE    VERSION
cl128p75cket5bspv91v-oles    Ready    <none>   101m   v1.18.9
cl128p75cket5bspv91v-oxuq    Ready    <none>   101m   v1.18.9
```

И теперь на нее смогут попадать новые поды!

Kubectl Metrics

Внутри кластера Kubernetes запускается metrics-server, который собирает базовые метрики — cpu & memory, их можно посмотреть через kubectl:

```
$ kubectl get podmetrics
NAME                       AGE
nginx-6dffdfcc57-cbndb     0s
whoami                     0s
alpine-5476d9677d-4mcv4    0s
alpine-5476d9677d-xbtrd    0s

$ kubectl get nodemetrics
NAME                         AGE
cl1ihia34v9pbjll4oic-yhes    0s
cl128p75cket5bspv91v-oles    0s
cl128p75cket5bspv91v-oxuq    0s

$ kubectl get podmetrics whoami -o yaml
apiVersion: metrics.k8s.io/v1beta1
containers:
- name: whoami
  usage:
    cpu: "0"
    memory: 7184Ki
kind: PodMetrics
metadata:
  creationTimestamp: "2020-12-04T20:32:37Z"
  name: whoami
  namespace: default
  selfLink:
/apis/metrics.k8s.io/v1beta1/namespaces/default/pods/whoami
timestamp: "2020-12-04T20:32:02Z"
window: 30s
$
```

Благодаря этим метрикам, мы можем использовать команды top в kubectl:

```
$ kubectl top pods --all-namespaces
```

```
NAMESPACE        NAME
CPU(cores)   MEMORY(bytes)
kube-system      coredns-7c646474c9-khb9r
3m           9Mi
kube-system      coredns-7c646474c9-s667q
4m           7Mi
kube-system      ip-masq-agent-rwjbp
1m           8Mi
kube-system      ip-masq-agent-sdhvn
1m           7Mi
kube-system      ip-masq-agent-swxdw
1m           6Mi
kube-system      kube-dns-autoscaler-7c867cbf8d-kfqkm
1m           12Mi
kube-system      kube-proxy-2bxxt
1m           26Mi
... skipped ...

$ kubectl top nodes
NAME                        CPU(cores)   CPU%   MEMORY(bytes)
MEMORY%
cl128p75cket5bspv91v-oles   53m          2%     839Mi
65%
cl128p75cket5bspv91v-oxuq   82m          4%     926Mi
72%
cl1ihia34v9pbjll4oic-yhes   162m         8%     1231Mi
45%
```

Это бывает очень удобно, чтобы вычислить самый тяжелый под или самую перегруженную ноду.

Полезные ссылки:

- Информация по администрированию кластера (документация)
- Обзор kubectl
- Kubernetes - полезные команды
- Kubectl Reference Docs

Задание:

1. Запустите под alpine с командой ping 8.8.8.8 (иначе он будет постоянно останавливаться) в namespace default.
2. Добавьте этому поду label internal=true.
3. Создайте файл /test.tmp внутри контейнера alpine.
4. Освободите любую ноду из кластера от подов — чтобы на ней нельзя было запустить поды и текущие поды также перенеслись на другую ноду.
5. Отправьте задание на проверку.