KUB 33: Custom Metrics для HPA

Описание:

Мы раньше обсуждали Horizontal Pod Autoscaler и сказали тогда, что в качестве метрик допустимо использовать свои кастомные метрики. Но для этого нам нужно было установить Prometheus Operator, чтобы мы могли их откуда-то брать. Поэтому теперь можем вернуться к теме Horizontal Pod Autoscaler и увеличить знания по метрикам.
Итак, у нас есть API metrics.k8s.io, custom.metrics.k8s.io. Если metrics.k8s.io наполняет metrics-server, то custom.metrics.k8s.io должен наполнять кто-то другой. По сути, нам нужен адаптер, который возьмет метрики из prometheus и перенесет их в объект Kubernetes. Для prometheus существует такой адаптер. Давайте попробуем его запустить и вытащить новую метрику в Kubernetes:

```
$ helm repo add prometheus-community
https://prometheus-community.github.io/helm-charts
$ helm repo update
$ helm pull prometheus-community/prometheus-adapter
$ tar zxf prometheus-adapter-2.8.0.tgz
```

Нам необходимо поменять values.yaml и сделать несколько изменений:

1. Указать адрес нашего prometheus-хоста. Мы запускали его в namespace monitoring, поэтому его адрес — prometheus-stack-kube-prom-prometheus.monitoring.svc.
2. Также обратите внимание на переменную RelistInterval — она должна быть равна или больше значения ScrapeInterval в Prometheus, иначе метрики из Kubernetes иногда могут пропадать.
3. Также очень важно поменять dnsPolicy на ClusterFirst, поскольку Default будет копировать конфигурацию dns с хост-машины, которая ничего не знает про сервисы в Kubernetes.

Давайте установим чарт:

```
$ helm -n monitoring upgrade --install prometheus-adapter -f
values.yaml ./
```

Удостоверимся, что он запущен и работает нормально:

```
$ kubectl -n  monitoring get deploy prometheus-adapter
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
prometheus-adapter   1/1     1            1           53s
$ kubectl get --raw /apis/custom.metrics.k8s.io/v1beta1
{"kind":"APIResourceList","apiVersion":"v1","groupVersion":"cust
om.metrics.k8s.io/v1beta1","resources":[{"name":"jobs.batch/prom
etheus_tsdb_tombstone_cleanup_seconds_bucket","singularName":"",
"namespaced":true,"kind":"MetricValueList","verbs":["get"]},{"na
```

```
me":"pods/node_pressure_cpu_waiting","singularName":"","namespac
ed":true,"kind":"MetricValueList","verbs":["get"]},{"name":"node
s/go_goroutines","singularName":"","namespaced":false,"kind":"Me
tricValueList","verbs":["get"]},{"name":"services/prometheus_tsd
b_head_gc_duration_seconds_sum","singularName":"","namespaced":t
rue,"kind":"MetricValueList","verbs":["get"]},{"name":"services/
net_conntrack_listener_conn_closed","singularName":"","namespace
d":true,"kind":"MetricValueList","verbs":["get"]},{"name":"names
paces/node_network_transmit_fifo","singularName":"","namespaced"
:false,"kind":"MetricValueList","verbs":["get"]},{"name":"namesp
aces/kube_deployment_status_replicas_available","singularName":"
","namespaced":fals

... skipped ...
```


Такой вывод будет означать, что prometheus-adapter научился собирать метрики с Prometheus. Теперь мы можем использовать метрику с whoami:

```
$ kubectl get --raw
/apis/custom.metrics.k8s.io/v1beta1/namespaces/default/pods/*/ht
tp_requests | jq .
{
  "kind": "MetricValueList",
  "apiVersion": "custom.metrics.k8s.io/v1beta1",
  "metadata": {
    "selfLink":
"/apis/custom.metrics.k8s.io/v1beta1/namespaces/default/pods/%2A
/http_requests"
  },
  "items": [
    {
      "describedObject": {
        "kind": "Pod",
        "namespace": "default",
        "name": "whoami",
        "apiVersion": "/v1"
      },
      "metricName": "http_requests",
      "timestamp": "2020-12-04T21:37:17Z",
      "value": "24007m",
      "selector": null
```

```
    }
  ]
}
```

Не удивляйтесь такому значению — я из соседнего пода запустил команду: for i in {1..10000}; do curl http://whoami-svc.default.svc/; done, которая постоянно отправляет на него запросы. Собственно, теперь осталось только написать наш HPA.
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: Pods
    pods:
      metricName: http_requests
      targetAverageValue: 4000m
```

В данном случае суффикс m означает милли. То есть 4000 millirequests per second. Или 4 запроса в секунду.

Задание:

1. Установите metrics server в kubernetes кластер.
2. Установите prometheus stack в namespace monitoring.
3. Создайте дейплоймент whoami-dp в namespace default, который внутри будет запускать контейнер whoami.
4. Создайте podmonitor whoami-monitor в namespace default, который будет собирать метрики с подов whoami.
5. Установите prometheus-adapter в namespace monitoring.

6. Создайте HorizontalPodAutoScaler hpa-whoami в namespace default, который будет увеличивать количество реплик данного деплоймента при достижении http_requests более 1ого запроса в секунду..
7. Отправьте задание на проверку.