OPERADOR DE PROMETHEUS PARA KUBERNETES

PRIMEROS PASOS

El operador de Prometheus es OperatorHub's Prometheus y su despliegue es sencillo usando el fichero bundle.yaml.

```
$ kubectl create -f
https://raw.githubusercontent.com/prometheus-operator/prometheus-opera
tor/master/bundle.yaml

customresourcedefinition.apiextensions.k8s.io/alertmanagers.monitoring
.coreos.com created
customresourcedefinition.apiextensions.k8s.io/podmonitors.monitoring.c
oreos.com created
customresourcedefinition.apiextensions.k8s.io/probes.monitoring.coreos
.com created
customresourcedefinition.apiextensions.k8s.io/prometheuses.monitoring.
coreos.com created
customresourcedefinition.apiextensions.k8s.io/prometheusrules.monitori
ng.coreos.com created
customresourcedefinition.apiextensions.k8s.io/servicemonitors.monitori
ng.coreos.com created
customresourcedefinition.apiextensions.k8s.io/thanosrulers.monitoring.
coreos.com created
clusterrolebinding.rbac.authorization.k8s.io/prometheus-operator
created
clusterrole.rbac.authorization.k8s.io/prometheus-operator created
deployment.apps/prometheus-operator created
serviceaccount/prometheus-operator created
service/prometheus-operator created
```

A la hora de utilizar el despliegue de Prometheus, será necesario crear una *service account* con los permisos adecuados para poder obtener las métricas de nuestros despliegues:

- La cuenta de servicio "prometheus":

```
$ cat<<EOF|kubectl create -f -
apiVersion: v1
kind: ServiceAccount
metadata:
  name: prometheus
EOF
```

```
clusterrole.rbac.authorization.k8s.io/prometheus created
```

- El rol de cluster “prometheus”:

```
$ cat<<EOF|kubectl create -f -
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
  name: prometheus
rules:
- apiGroups: [""]
  resources:
  - nodes
  - nodes/metrics
  - services
  - endpoints
  - pods
  verbs: ["get", "list", "watch"]
- apiGroups: [""]
  resources:
  - configmaps
  verbs: ["get"]
- nonResourceURLs: ["/metrics"]
  verbs: ["get"]
EOF
```

```
clusterrole.rbac.authorization.k8s.io/prometheus created
```

- Y por último, la asociación de los permisos definidos a la cuenta de servicio:

```
$ cat<<EOF|kubectl create -f -
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: prometheus
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus
subjects:
```

```
- kind: ServiceAccount
  name: prometheus
  namespace: default
EOF
```

```
clusterrolebinding.rbac.authorization.k8s.io/prometheus
```

NOTA: En este ejemplo se está utilizando el namespace "*default*" y la creación dinámica de ficheros de recursos. En un entorno productivo es necesario tener en cuenta los *namespaces* sobre los que debe aplicarse el rol definido y siempre usar ficheros de definición de recursos para poder almacenarlos y gestionarlos de forma adecuada.

Con estos pasos se ha preparado el entorno para poder desplegar diferentes instancias de Prometheus usando el recurso de tipo "*Prometheus*".

No obstante, antes de desplegar un entorno de monitorización, es necesario desplegar un sencillo servidor web. Utilizaremos NGINX con una configuración especial que incluya un *exporter* de métricas para Prometheus. Para ello, creamos un *ConfigMap* que habilite el site de estadísticas de NGINX, en este caso NGINX Community:

```
$ cat<<EOF|kubectl create -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-metrics
data:
  metrics.conf: |
    server {
      listen       8080;
      server_name  localhost;
      location /metrics {
        stub_status on;
      }
    }
EOF
```

```
configmap/nginx-metrics created
```

- En este caso publicamos las métricas en el puerto 8080 en el path /metrics.

NOTA: Recordad que se debe cerrar el acceso al site de estadísticas desde el exterior.

A continuación se prepara el despliegue de "*webserver*" incluyendo el exporter de NGINX ejecutándose como *Adapter*:

```
$ cat<<EOF|kubectl create -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    env: lab
    app: webserver
  name: webserver
spec:
  selector:
    matchLabels:
      app: webserver
  replicas: 2
  template:
    metadata:
      labels:
        env: lab
        app: webserver
    spec:
      containers:
      - name: nginx
        image: nginx:alpine
        ports:
        - containerPort: 80
        volumeMounts:
        - mountPath: /etc/nginx/conf.d
          name: metrics
      - name: nginx-exporter
        image: nginx/nginx-prometheus-exporter:0.8.0
        ports:
        - containerPort: 9113
        args: ["-nginx.scrape-uri","http://localhost:8080/metrics"]
      volumes:
      - name: metrics
        configMap:
          name: nginx-metrics
EOF
```

```
deployment.apps/webserver created
```

Utilizamos el *exporter* oficial de NGINX: Nginx-prometheus-exporter, publicado en el puerto 9113, el habitual en estas herramientas. Convertirá las métricas de NGINX en el estándar de Prometheus.

Por último, vamos a desplegar un servicio que publique nuestro "*webserver*". Hemos definido el puerto del *exporter* como "exporter" para poder referenciarlo posteriormente en la monitorización.

```
$ cat<<EOF|kubectl create -f -
apiVersion: v1
kind: Service
metadata:
  labels:
    env: lab
    app: webserver
  name: webserver
spec:
  selector:
    app: webserver
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
      name: web
    - protocol: TCP
      port: 9113
      targetPort: 9113
      name: exporter
EOF
```

```
service/webserver created
```

Verificamos el entorno accediendo a las métricas directamente usando la IP de los Pod de nuestro "*webserver*":

```
$ kubectl get pods -o wide
```

```
NAME
READY   STATUS    RESTARTS   AGE     IP              NODE
NOMINATED NODE   READINESS GATES
example-app-bb759dfcc-9qwtq                                       1/1
Running   0          6h11m   10.10.11.149   worker1   <none>
<none>
example-app-bb759dfcc-vbfwd                                       1/1
Running   0          6h11m   10.10.9.203    worker2   <none>
<none>
example-app-bb759dfcc-vxsfg                                       1/1
Running   0          6h11m   10.10.11.148   worker1   <none>
<none>
nginx-controller-nginx-ingress-controller-b645f985d-l5qnn         1/1
Running   2          4d      10.10.11.145   worker1   <none>
<none>
nginx-controller-nginx-ingress-controller-default-backend-46qc4   1/1
Running   1          4d      10.10.11.144   worker1   <none>
<none>
prometheus-operator-6cf57d84f-s4xhb                               1/1
Running   0          40m     10.10.11.158   worker1   <none>
<none>
webserver-66d844fc67-l6qgh                                        2/2
Running   0          114m    10.10.9.214    worker2   <none>
<none>
webserver-66d844fc67-sxh8c                                        2/2
Running   0          114m    10.10.11.157   worker1   <none>
<none>
wordpress-bfd5f4c68-z7cnt                                         1/1
Running   8          5d      10.10.9.202    worker2   <none>
<none>
wordpress-mariadb-0                                               1/1
Running   1          4d      10.10.11.147   worker1   <none>
<none>
```

```
$ curl 10.10.9.214:9113/metrics
# HELP nginx_connections_accepted Accepted client connections
# TYPE nginx_connections_accepted counter
nginx_connections_accepted 4
# HELP nginx_connections_active Active client connections
# TYPE nginx_connections_active gauge
```

```
nginx_connections_active 1
# HELP nginx_connections_handled Handled client connections
# TYPE nginx_connections_handled counter
nginx_connections_handled 4
# HELP nginx_connections_reading Connections where NGINX is reading
the request header
# TYPE nginx_connections_reading gauge
nginx_connections_reading 0
# HELP nginx_connections_waiting Idle client connections
# TYPE nginx_connections_waiting gauge
nginx_connections_waiting 0
# HELP nginx_connections_writing Connections where NGINX is writing
the response back to the client
# TYPE nginx_connections_writing gauge
nginx_connections_writing 1
# HELP nginx_http_requests_total Total http requests
# TYPE nginx_http_requests_total counter
nginx_http_requests_total 123
# HELP nginx_up Status of the last metric scrape
# TYPE nginx_up gauge
nginx_up 1
# HELP nginxexporter_build_info Exporter build information
# TYPE nginxexporter_build_info gauge
nginxexporter_build_info{gitCommit="",version=""} 1
```

Como se puede apreciar, está accesible internamente, sin embargo, al publicar el puerto con el servicio, también aparece publicado en modo NodePort.

NOTA: Desaconsejamos esta acción. Debemos publicar solo internamente o bien usar una *NetworkPolicy* que permita únicamente el uso de las métricas desde los despliegues de Prometheus.

```
$ kubectl get svc
NAME                                                    TYPE
CLUSTER-IP       EXTERNAL-IP   PORT(S)                  AGE
kubernetes                                              ClusterIP
10.96.0.1        <none>        443/TCP                  5d5h
prometheus-operator                                     ClusterIP
None             <none>        8080/TCP                 58m
webserver                                               ClusterIP
10.102.48.37     <none>        80/TCP,9113/TCP          10m
```

Desplegamos ahora una instancia de Prometheus que nos permita monitorizar nuestro sencillo "webserver". Para ello, crearemos un recurso de tipo Prometheus que usará el operador para crear todo lo necesario para ejecutar Prometheus.

```
$ cat<<EOF|kubectl create -f -
apiVersion: monitoring.coreos.com/v1
kind: Prometheus
metadata:
  name: prometheus
spec:
  serviceAccountName: prometheus
  serviceMonitorSelector:
    matchLabels:
      app: webserver
  resources:
    requests:
      memory: 400Mi
  enableAdminAPI: false
EOF

prometheus.monitoring.coreos.com/prometheus created
```

Verificamos que nuestro despliegue de Prometheus está en ejecución, revisando las instancias de recursos de tipo Prometheus:

```
$ kubectl get prometheus
NAME         VERSION   REPLICAS   AGE
prometheus                        14s
```

Tal y como observamos en la definición de nuestro recurso, definimos la *service account* que se usará como **serviceAccountName**. Además, indicamos qué servicios vamos a asociar en este caso a esta instancia de Prometheus, utilizando **serviceMonitorSelector**.

Esta definición seleccionará únicamente las definiciones de tipo ServiceMonitor, definido por el operador de Prometehus y etiquetadas como app=webserver.

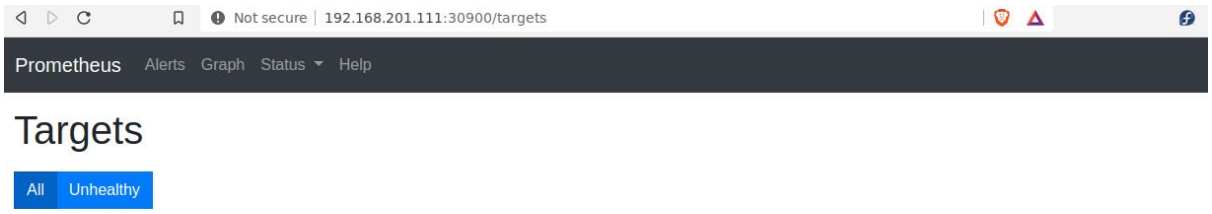
Sin embargo, antes de crear el recurso de tipo ServiceMonitor, vamos a acceder a Prometheus. Para ello, simplemente publicamos el despliegue de Prometheus. Nos interesa que la publicación se realice en un puerto fijo, ya que se trata de una aplicación de

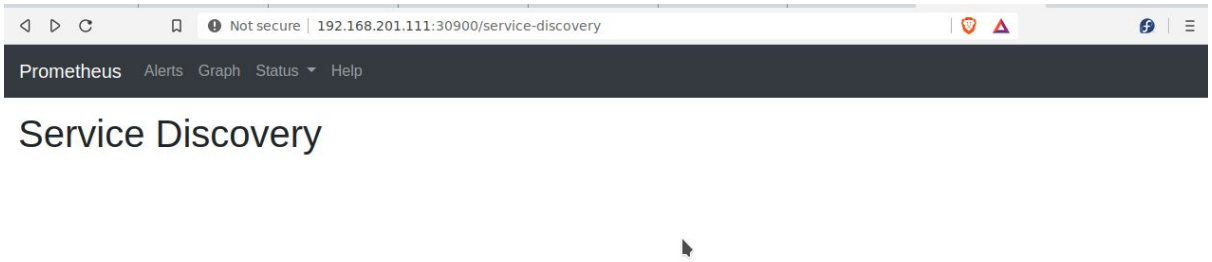
monitorización y es habitualmente deseable; aunque siempre podemos usar un Ingress Controller para publicarla. Por sencillez en el ejemplo, usaremos NodePort:

```
$ cat<<EOF|kubectl create -f -
apiVersion: v1
kind: Service
metadata:
  name: prometheus
spec:
  type: NodePort
  ports:
  - name: web
    nodePort: 30900
    port: 9090
    protocol: TCP
    targetPort: web
  selector:
    prometheus: prometheus
EOF

service/prometheus created
```

Accedemos a Prometheus en el puerto 30900:





Observamos que la configuración de Targets está vacía, al igual que los sistemas descubiertos y no gestionados (“dropped”). Esto se debe a que no se ha definido aún ninguna monitorización ni filtros.

Definiremos ahora una monitorización de servicio, ServiceMonitor.

```
$ cat<<EOF|kubectl create -f -
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: webserver
  labels:
    app: webserver
    env: labs
spec:
  selector:
    matchLabels:
      app: webserver
```

```
  endpoints:
  - port: exporter

EOF
```

```
servicemonitor.monitoring.coreos.com/webserver created
```

Como cualquier objeto de Kubernetes que debe vincularse con otros, usa selectores para poder asociar recursos. En este caso, nuestro “webserver” se ejecuta con las etiquetas app=webserver, entre otras. Incluimos además el puerto que debe tomarse como origen de datos. En nuestro ejemplo usamos el puerto “exporter” tal y como definimos en nuestro despliegue, asociado al contenedor de exporter de NGINX.

Vemos nuestra monitorización dinámica creada utilizando el nuevo recurso de tipo ServiceMonitor:

```
$ kubectl get servicemonitor
NAME         AGE
webserver    5m46s
```

Ahora observamos cómo el operador de Prometheus ha generado las configuraciones necesarias, usando los filtros del recurso para añadir de forma dinámica los endpoints de monitorización en el puerto 9113 (“exporter”).