## Настройка Servicemonitor и Podmonitor

## Цели практики

Целью данной практической работы является знакомство со структурой custom resource Servicemonitor и Podmonitor.

### Servicemonitor

Servicemonitor - это абстракция, с помощью которой реализован механизм Service discovery в Prometheus Operator. В данной абстракции описывается, из каких Service необходимо получить список Endpoint, и задаются правила, по которым необходимо производить scraping. В общем случае манифест Servicemonitor выглядит примерно так:

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: prom-kube-prometheus-nodexporter
spec:
  endpoints:
  - interval: 1m
    path: /metrics
    port: metrics-port
  namespaceSelector:
   matchNames:
     - prometheus-operator
  selector:
    matchLabels:
      app: example

Назначение полей:

spec.endpoints - содержит настройки, по которым должен производиться scraping, такие как: частота опроса, endpoint, может быть указан аутентификация, настройки tls и так далее.

spec.endpoints.port - задается порт, по которому будет производиться scraping. Важной особенностью является то, что Servicemonitor работает только с именованными портами, то есть указать port: 443 нельзя.
```

```
spec.namespaceSelector - задает, в каком namespace будет производиться scraping. Возможные значения: matchNames и any: true/false. В случае использования any: true будет производиться по всем namespace.

spec.selector - содержит настройку, какие labels должна содержать служба, для которой будет получаться список endpoints. Может содержать следующие параметры:

matchLabels - список labels, которые должны быть у службы.

matchExpressions - регулярное выражение для discovery Service на основании наличия или отсутствие labels. Формат регулярных выражений:

- key: serviceapp
 operator: Exists
```

Полный список полей можно посмотреть здесь:

https://docs.openshift.com/container-platform/4.4/rest_api/monitoring_apis/servicemonitor-monitoring-coreos-com-v1.html

**PodMonitor**

PodMonitor - это абстракция, с помощью которой реализован механизм Service discovery в Prometheus Operator. В данной абстракции описываются, какие Pod необходимо добавить в Prometheus  и правила, по которым необходимо производить scraping. В общем случае, манифест PodMonitor выглядит примерно так:

```
apiVersion: monitoring.coreos.com/v1

kind: PodMonitor

metadata:

  name: example-app

spec:

  namespaceSelector:
```

```
    matchNames:
      - prometheus-operator
  selector:
    matchLabels:
      app: example
  podMetricsEndpoints:
  - port: web
    honorLabels: true
```

`spec.namespaceSelector` - задает, в каком namespace будет производиться scraping. Возможные значения: `matchNames` и `any: true/false`. В случае использования `any: true` будет производиться по всем namespace.

`spec.selector` - содержит настройку, какие метки должна содержать служба, для которой будет получаться список endpoints. Может содержать следующие параметры: `matchNames` и `any: true/false`. В случае использования `any: true` будет производиться по всем namespace.

`podMetricsEndpoints` - содержит описание, какие порты должны использоваться для scraping, а также другие настройки, относящиеся к scraping.

Полный список полей можно посмотреть здесь: https://docs.openshift.com/container-platform/4.4/rest_api/monitoring_apis/podmonitor-monitoring-coreos-com-v1.html

Создайте Servicemonitor для Nginx ingress с именем: `prom-kube-prometheus-nginx`.

Также сервис должен содержать следующие labels:

```
app: nginx-ingress

component: controller

release: nginx-ingress
```

Порт для scraping называется: `metrics`. Интервал для scraping: `1m`, а поиск службы должен производиться по всем namespace.

### Настройка Prometheus rules

Целью данной практической работы является знакомство со структурой custom resource: PrometheusRule.

### PrometheusRule

Custom resource PrometheusRule достаточно простой, он содержит стандартные поля Kubernetes абстракции, а в поле spec описываются Rules с таким же синтаксисом, как и у Prometheus. В общем случае, PrometheusRule выглядит примерно так:

```
apiVersion: monitoring.coreos.com/v1

kind: PrometheusRule

metadata:

  name: prometheus-example-rules

spec:

  groups:

  - name: node.rules
```

```
    rules:
    - expr: sum(min(kube_pod_info{node!=""}) by (cluster, node))
      record: ':kube_pod_info_node_count:'
    - expr: |-
        topk by(namespace, pod) (1,
          max by (node, namespace, pod) (
            label_replace(kube_pod_info{job="kube-state-metrics",node!=""},
"pod", "$1", "pod", "(.*)")
        ))
      record: 'node_namespace_pod:kube_pod_info'
```

Создайте PrometheusRule с именем: `prom-ingress.rules`,

который вычисляет выражение: `rate(nginx_ingress_controller_requests[5m])`

и сохраняет его с именем: `nginx_ingress_controller_requests_per_second`.

Имя группы: `Ingress`