上节课我们一起学习了如何在 Prometheus Operator 下面自定义一个监控选项，以及自定义报警规则的使用。那么我们还能够直接使用前面课程中的自动发现功能吗？如果在我们的 Kubernetes 集群中有了很多的 Service/Pod，那么我们都需要一个一个的去建立一个对应的 ServiceMonitor 对象来进行监控吗？这样岂不是又变得麻烦起来了？

自动发现配置

为解决上面的问题，Prometheus Operator 为我们提供了一个额外的抓取配置的来解决这个问题，我们可以通过添加额外的配置来进行服务发现进行自动监控。和前面自定义的方式一样，我们想要在 Prometheus Operator 当中去自动发现并监控具有`prometheus.io/scrape=true`这个 annotations 的 Service，之前我们定义的 Prometheus 的配置如下：

```
- job_name: 'kubernetes-service-endpoints'
  kubernetes_sd_configs:
  - role: endpoints
  relabel_configs:
  - source_labels: [__meta_kubernetes_service_annotation_prometheus_io_scrape]
    action: keep
    regex: true
  - source_labels: [__meta_kubernetes_service_annotation_prometheus_io_scheme]
    action: replace
    target_label: __scheme__
    regex: (https?)
  - source_labels: [__meta_kubernetes_service_annotation_prometheus_io_path]
    action: replace
    target_label: __metrics_path__
    regex: (.+)
  - source_labels: [__address__, __meta_kubernetes_service_annotation_prometheus_io_port]
    action: replace
    target_label: __address__
    regex: ([^:]+)(?::\d+)?;(\d+)
    replacement: $1:$2
  - action: labelmap
    regex: __meta_kubernetes_service_label_(.+)
  - source_labels: [__meta_kubernetes_namespace]
    action: replace
    target_label: kubernetes_namespace
  - source_labels: [__meta_kubernetes_service_name]
```

```
    action: replace
    target_label: kubernetes_name
```

如果你对上面这个配置还不是很熟悉的话，建议去查看下前面关于 Kubernetes常用资源对象监控章节的介绍，要想自动发现集群中的 Service，就需要我们在 Service 的`annotation`区域添加`prometheus.io/scrape=true`的声明，将上面文件直接保存为 prometheus-additional.yaml，然后通过这个文件创建一个对应的 Secret 对象：

```
$ kubectl create secret generic additional-configs --from-file=prometheus-additional.yaml -n monitoring
secret "additional-configs" created
```

> 注意我们所有的操作都在 Prometheus Operator 源码`contrib/kube-prometheus/manifests/`目录下面。

创建完成后，会将上面配置信息进行 base64 编码后作为 prometheus-additional.yaml 这个 key 对应的值存在：

```
$ kubectl get secret additional-configs -n monitoring -o yaml
apiVersion: v1
data:
  prometheus-additional.yaml: LSBqb2JfbmFtZTogJ2t1YmVybmV0ZXMtc2VydmljZS1lbmRwb2ludHMnCiAga3ViZXJuZXRlc19zZF9jb25maWdzOgogIC0gcm9sZTogZW5kcG9pbnRzCiAgcmVsYWJlbF9jb25maWdzOgogIC0gc291cmNlX2xhYmVsczogW19fbWV0YV9rdWJlcm5ldGVzX3NlcnZpY2VfYW5ub3RhdGlvbl9wcm9tZXRoZXVzX2lvX3NjcmFwZV0KICAgIGFjdGlvbjoga2VlcAogICAgcmVnZXg6IHRydWUKICAtIHNvdXJjZV9sYWJlbHM6IFtfX21ldGFfa3ViZXJuZXRlc19zZXJ2aWNlX2Fubm90YXRpb25fcHJvbWV0aGV1c19pb19zY2hlbWVdCiAgICBhY3Rpb246IHJlcGxhY2UKICAgIHRhcmdldF9sYWJlbDogX19zY2hlbWVfXwogICAgcmVnZXg6IChodHRwcz8pCiAgLSBzb3VyY2VfbGFiZWxzOiBbX19tZXRhX2t1YmVybmV0ZXNfc2VydmljZV9hbm5vdGF0aW9uX3Byb21ldGhldXNfaW9fcGF0aF0KICAgIGFjdGlvbjogcmVwbGFjZQogICAgdGFyZ2V0X2xhYmVsOiBfX21ldHJpY3NfcGF0aF9fCiAgICByZWdleDogKC4rKQogIC0gc291cmNlX2xhYmVsczogW19fYWRkcmVzc19fLCBfX21ldGFfa3ViZXJuZXRlc19zZXJ2aWNlX2Fubm90YXRpb25fcHJvbWV0aGV1c19pb19wb3J0XQogICAgYWN0aW9uOiByZXBsYWNlCiAgICB0YXJnZXRfbGFiZWw6IF9fYWRkcmVzc19fCiAgICByZWdleDogKFteOl0rKSg/OjpcZCspPzsoXGQrKQogICAgcmVwbGFjZW1lbnQ6ICQxOiQyCiAgLSBhY3Rpb246IGxhYmVsbWFwCiAgICByZWdleDogX19tZXRhX2t1YmVybmV0ZXNfc2VydmljZV9sYWJlbF8oLispCiAgLSBzb3VyY2VfbGFiZWxzOiBbX19tZXRhX2t1YmVybmV0ZXNfbmFtZXNwYWNlXQogICAgYWN0aW9uOiByZXBsYWNlCiAgICB0YXJnZXRfbGFiZWw6IGt1YmVybmV0ZXNfbmFtZXNwYWNlCiAgLSBzb3VyY2VfbGFiZWxzOiBbX19tZXRhX2t1YmVybmV0ZXNfc2VydmljZV9uYW1lXQogICAgYWN0aW9uOiByZXBsYWNlCiAgICB0YXJnZXRfbGFiZWw6IGt1YmVybmV0ZXNfbmFtZQo=
kind: Secret
metadata:
  creationTimestamp: 2018-12-20T14:50:35Z
  name: additional-configs
  namespace: monitoring
```

```
  resourceVersion: "41814998"
  selfLink: /api/v1/namespaces/monitoring/secrets/additional-configs
  uid: 9bbe22c5-0466-11e9-a777-525400db4df7
type: Opaque
```

然后我们只需要在声明 prometheus 的资源对象文件中添加上这个额外的配置：(prometheus-prometheus.yaml)

```
apiVersion: monitoring.coreos.com/v1
kind: Prometheus
metadata:
  labels:
    prometheus: k8s
  name: k8s
  namespace: monitoring
spec:
  alerting:
    alertmanagers:
    - name: alertmanager-main
      namespace: monitoring
      port: web
  baseImage: quay.io/prometheus/prometheus
  nodeSelector:
    beta.kubernetes.io/os: linux
  replicas: 2
  secrets:
  - etcd-certs
  resources:
    requests:
      memory: 400Mi
  ruleSelector:
    matchLabels:
      prometheus: k8s
      role: alert-rules
  securityContext:
    fsGroup: 2000
    runAsNonRoot: true
    runAsUser: 1000
```

```
  additionalScrapeConfigs:
    name: additional-configs
    key: prometheus-additional.yaml
  serviceAccountName: prometheus-k8s
  serviceMonitorNamespaceSelector: {}
  serviceMonitorSelector: {}
  version: v2.5.0
```

添加完成后，直接更新 prometheus 这个 CRD 资源对象：

```
$ kubectl apply -f prometheus-prometheus.yaml
prometheus.monitoring.coreos.com "k8s" configured
```

隔一小会儿，可以前往 Prometheus 的 Dashboard 中查看配置是否生效：

在 Prometheus Dashboard 的配置页面下面我们可以看到已经有了对应的的配置信息了，但是我们切换到 targets 页面下面却并没有发现对应的监控任务，查看 Prometheus 的 Pod 日志：

```
$ kubectl logs -f prometheus-k8s-0 prometheus -n monitoring
level=error ts=2018-12-20T15:14:06.772903214Z caller=main.go:240
component=k8s_client_runtime
err="github.com/prometheus/prometheus/discovery/kubernetes/kubernetes.go:302: Failed
to list *v1.Pod: pods is forbidden: User \"system:serviceaccount:monitoring:prometheus-
k8s\" cannot list pods at the cluster scope"
level=error ts=2018-12-20T15:14:06.773096875Z caller=main.go:240
component=k8s_client_runtime
err="github.com/prometheus/prometheus/discovery/kubernetes/kubernetes.go:301: Failed
to list *v1.Service: services is forbidden: User
\"system:serviceaccount:monitoring:prometheus-k8s\" cannot list services at the
cluster scope"
level=error ts=2018-12-20T15:14:06.773212629Z caller=main.go:240
component=k8s_client_runtime
err="github.com/prometheus/prometheus/discovery/kubernetes/kubernetes.go:300: Failed
to list *v1.Endpoints: endpoints is forbidden: User
\"system:serviceaccount:monitoring:prometheus-k8s\" cannot list endpoints at the
cluster scope"
......
```

可以看到有很多错误日志出现，都是`xxx is forbidden`，这说明是 RBAC 权限的问题，通过 prometheus 资源对象的配置可以知道 Prometheus 绑定了一个名为 prometheus-k8s 的

ServiceAccount 对象，而这个对象绑定的是一个名为 prometheus-k8s 的 ClusterRole：（prometheus-clusterRole.yaml）

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus-k8s
rules:
- apiGroups:
  - ""
  resources:
  - nodes/metrics
  verbs:
  - get
- nonResourceURLs:
  - /metrics
  verbs:
  - get
```

上面的权限规则中我们可以看到明显没有对 Service 或者 Pod 的 list 权限，所以报错了，要解决这个问题，我们只需要添加上需要的权限即可：

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus-k8s
rules:
- apiGroups:
  - ""
  resources:
  - nodes
  - services
  - endpoints
  - pods
  - nodes/proxy
  verbs:
  - get
```

```
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - configmaps
  - nodes/metrics
  verbs:
  - get
- nonResourceURLs:
  - /metrics
  verbs:
  - get
```

更新上面的 ClusterRole 这个资源对象，然后重建下 Prometheus 的所有 Pod，正常就可以看到 targets 页面下面有 kubernetes-service-endpoints 这个监控任务了：

我们这里自动监控了两个 Service，第一个就是我们之前创建的 Redis 的服务，我们在 Redis Service 中有两个特殊的 annotations：

```
annotations:
  prometheus.io/scrape: "true"
  prometheus.io/port: "9121"
```

所以被自动发现了，当然我们也可以用同样的方式去配置 Pod、Ingress 这些资源对象的自动发现。

数据持久化

上面我们在修改完权限的时候，重启了 Prometheus 的 Pod，如果我们仔细观察的话会发现我们之前采集的数据已经没有了，这是因为我们通过 prometheus 这个 CRD 创建的 Prometheus 并没有做数据的持久化，我们可以直接查看生成的 Prometheus Pod 的挂载情况就清楚了：

```
$ kubectl get pod prometheus-k8s-0 -n monitoring -o yaml
......
    volumeMounts:
    - mountPath: /etc/prometheus/config_out
      name: config-out
```

```
      readOnly: true
    - mountPath: /prometheus
      name: prometheus-k8s-db
.....
  volumes:
.....
  - emptyDir: {}
    name: prometheus-k8s-db
.....
```

我们可以看到 Prometheus 的数据目录 /prometheus 实际上是通过 emptyDir 进行挂载的，我们知道 emptyDir 挂载的数据的生命周期和 Pod 生命周期一致的，所以如果 Pod 挂掉了，数据也就丢失了，这也就是为什么我们重建 Pod 后之前的数据就没有了的原因，对应线上的监控数据肯定需要做数据的持久化的，同样的 prometheus 这个 CRD 资源也为我们提供了数据持久化的配置方法，由于我们的 Prometheus 最终是通过 Statefulset 控制器进行部署的，所以我们这里需要通过 storageclass 来做数据持久化，首先创建一个 StorageClass 对象：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: prometheus-data-db
provisioner: fuseim.pri/ifs
```

这里我们声明一个 StorageClass 对象，其中 provisioner=fuseim.pri/ifs，则是因为我们集群中使用的是 nfs 作为存储后端，而前面我们课程中创建的 nfs-client-provisioner 中指定的 PROVISIONER_NAME 就为 fuseim.pri/ifs，这个名字不能随便更改，将该文件保存为 prometheus-storageclass.yaml:

```
$ kubectl create -f prometheus-storageclass.yaml
storageclass.storage.k8s.io "prometheus-data-db" created
```

然后在 prometheus 的 CRD 资源对象中添加如下配置：

```
storage:
  volumeClaimTemplate:
```

```
    spec:
      storageClassName: prometheus-data-db
      resources:
        requests:
          storage: 10Gi
```

注意这里的 storageClassName 名字为上面我们创建的 StorageClass 对象名称，然后更新 prometheus 这个 CRD 资源。更新完成后会自动生成两个 PVC 和 PV 资源对象：

```
$ kubectl get pvc -n monitoring
NAME                                 STATUS    VOLUME
   CAPACITY   ACCESS MODES   STORAGECLASS         AGE
prometheus-k8s-db-prometheus-k8s-0   Bound     pvc-0cc03d41-047a-11e9-a777-
525400db4df7   10Gi       RWO            prometheus-data-db   8m
prometheus-k8s-db-prometheus-k8s-1   Bound     pvc-1938de6b-047b-11e9-a777-
525400db4df7   10Gi       RWO            prometheus-data-db   1m
$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY
STATUS      CLAIM                                          STORAGECLASS
REASON    AGE
pvc-0cc03d41-047a-11e9-a777-525400db4df7   10Gi       RWO            Delete
Bound       monitoring/prometheus-k8s-db-prometheus-k8s-0   prometheus-data-db
    2m
pvc-1938de6b-047b-11e9-a777-525400db4df7   10Gi       RWO            Delete
Bound       monitoring/prometheus-k8s-db-prometheus-k8s-1   prometheus-data-db
    1m
```

现在我们再去看 Prometheus Pod 的数据目录就可以看到是关联到一个 PVC 对象上了。

```
$ kubectl get pod prometheus-k8s-0 -n monitoring -o yaml
......
    volumeMounts:
    - mountPath: /etc/prometheus/config_out
      name: config-out
      readOnly: true
    - mountPath: /prometheus
      name: prometheus-k8s-db
......
  volumes:
......
```

```
  - name: prometheus-k8s-db
    persistentVolumeClaim:
      claimName: prometheus-k8s-db-prometheus-k8s-0
......
```

现在即使我们的 Pod 挂掉了，数据也不会丢失了，最后，下面是我们 Prometheus Operator 系列课程中最终的创建资源清单文件，更多的信息可以在https://github.com/cnych/kubernetes-learning 下面查看。

```
apiVersion: monitoring.coreos.com/v1
kind: Prometheus
metadata:
  labels:
    prometheus: k8s
  name: k8s
  namespace: monitoring
spec:
  alerting:
    alertmanagers:
    - name: alertmanager-main
      namespace: monitoring
      port: web
  storage:
    volumeClaimTemplate:
      spec:
        storageClassName: prometheus-data-db
        resources:
          requests:
            storage: 10Gi
  baseImage: quay.io/prometheus/prometheus
  nodeSelector:
    beta.kubernetes.io/os: linux
  replicas: 2
  secrets:
  - etcd-certs
  additionalScrapeConfigs:
    name: additional-configs
    key: prometheus-additional.yaml
```

```
  resources:
    requests:
      memory: 400Mi
  ruleSelector:
    matchLabels:
      prometheus: k8s
      role: alert-rules
  securityContext:
    fsGroup: 2000
    runAsNonRoot: true
    runAsUser: 1000
  serviceAccountName: prometheus-k8s
  serviceMonitorNamespaceSelector: {}
  serviceMonitorSelector: {}
  version: v2.5.0
```

到这里 Prometheus Operator 系列教程就告一段落了，大家还有什么问题可以到微信群里面继续交流，接下来会和大家介绍 Kubernetes 日志收集方便的知识点