容器云平台监控告警体系（三）—— 使用Prometheus Operator部署并管理Prometheus Server

1、概述

Prometheus Operator是一种基于Kubernetes的应用程序，用于管理Prometheus实例和相关的监控组件。它是由CoreOS开发的开源工具，旨在简化Prometheus和相关监控组件的部署和配置。

容器云平台通过使用Prometheus Operator简化在Kubernetes下部署和管理Prmetheus的复杂度，其通过prometheuses.monitoring.coreos.com资源声明式创建和管理Prometheus Server实例；其通过servicemonitors.monitoring.coreos.com和podmonitors.monitoring.coreos.com资源声明式的管理监控配置；其通过prometheusrules.monitoring.coreos.com资源进行规则记录，实现对复杂查询的 PromQL 语句的性能优化，提高查询效率。

2、先决条件

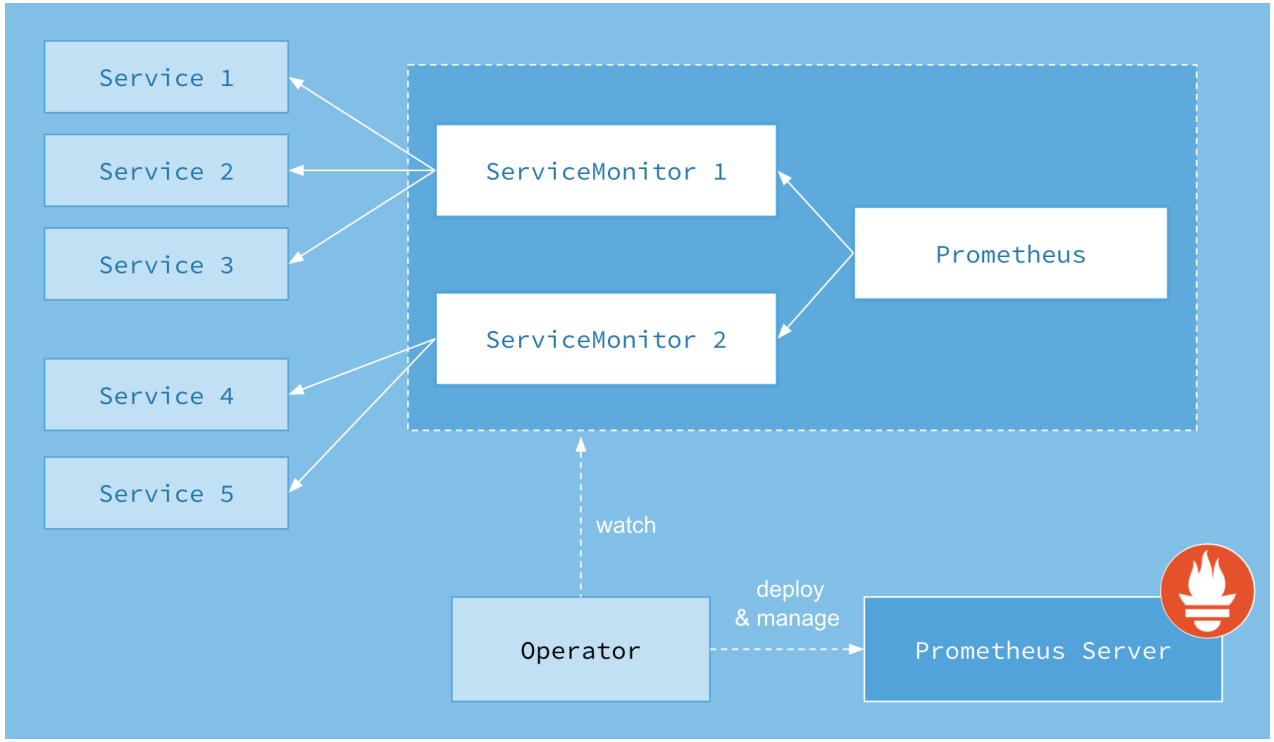
截止2023年3月30日Prometheus Operator最新版本为0.64.0，对于0.39.0及以上版本需要Kubernetes集群的版本 `>=1.16.0,` 如果您刚刚开始使用Prometheus Operator，强烈建议使用最新版本。

如果您有旧版本的Kubernetes和正在运行的Prometheus Operator，建议先升级 Kubernetes，然后再升级 Prometheus Operator。

> **注意：**Prometheus Operator安装和卸载比较简单，直接参考 Prometheus Operator 工程的 README.md 文件即可，推荐使用Helm方式进行安装，本文不再讲解。在容器云平台中是将Prometheus Operator中安装相关的yaml文件下载下来，然后通过Kubectl apply方式进行安装。

3、Prometheus Operator的工作原理

简单来说Prometheus Operator是一组自定义的CRD资源以及Controller的实现，Prometheus Operator负责监听这些自定义资源的变化，并且根据这些资源的定义自动化的完成Prometheus实例和相关的监控组件的自动化管理工作，Prometheus Operator 官方提供的架构图如下：



从下向上看，Operator可以部署并且管理Prometheus Server，并且Operator可以监听Prometheus和ServiceMonitor这两种自定义资源对象的变化。当Operator监听到Prometheus资源对象变化时，会对应部署或者管理Prometheus Server；当Operator监听到ServiceMonitor资源对象变化时，会对应修改Prometheus Server配置文件。上图中Service1 - Service5是Kubernetes中的Service资源，ServiceMonitor资源对象通过labelSelector的方式去匹配一类Service（一般来说，一个ServiceMonitor资源对象对应一个Service资源对象），Prometheus也通过labelSelector去匹配一个或多个ServiceMonitor。

- Operator：Operator是整个系统的主要控制器，会以Deployment的方式运行于Kubernetes集群上，并根据自定义资源（Custom Resources Definition）CRD 来负责部署与管理Prometheus Server，Operator会通过监听这些CRD的变化来做出相对应的处理。
- Prometheus：Operator会观察集群内的Prometheus CRD资源对象，并在指定命名空间下（默认是monitoring命名空间下）创建一个StatefulSet资源对象来管理Prometheus Server，并且挂载了一个名为prometheus-"**name**"（Prometheus自定义资源名称）的Secret为Volume到/etc/prometheus/config目录，Secret的data包含了以下内容：
  - prometheus.yaml.gz：为Prometheus Server主配置文件，Operator监听到Prometheus Server配置变更会更新secret(文件prometheus-yaml.gz，使用gz保证<1M)，config-reloader监控到prometheus-yaml.gz文件有变更，将其解压至prometheus-env.yaml，然后发送reload给prometehus。

- ServiceMonitor ： Operator会通过监听ServiceMonitor资源对象的变化来动态生成Prometheus的配置文件中的Scrape targets（抓取目标），并让这些配置实时生效，Operator通过将生成的监控目标更新到上面的prometheus-k8s这个Secret的Data的prometheus.yaml.gz字段里，然后Prometheus Server这个Pod里的Sidecar容器 `prometheus-config-reloader` 当检测到挂载路径的文件发生改变后自动去执行HTTP Post请求到 `/api/-reload-` 路径去reload配置。该自定义资源（CRD）通过labels选取对应的Service,并让Prometheus Server通过选取的Service拉取对应的监控信息（metrics）。
- Service ： Service其实就是指Kubernetes的Service资源，简单的说就是 Prometheus 监控的对象，比如部署在Kbernetes上的Mysql-Exporter的Service。

想象一下，我们以传统的方式去监控一个Mysql服务，首先需要安装Mysql-Exporter，获取Mysql Metrics，并且暴露一个端口，等待Prometheus Server服务来拉取监控信息，然后去Prometheus Server的prometheus.yaml文件中在scarpe_config中添加Mysql-Exporter的job，配置Mysql-Exporter的地址和端口等信息，再然后，需要重启Prometheus服务，就完成添加一个mysql监控的任务。

现在我们以Prometheus-Operator的方式来部署Prometheus，当我们需要添加一个Mysql监控我们会怎么做，首先第一步和传统方式一样，部署一个Mysql-Exporter来获取Mysql监控项，然后编写一个ServiceMonitor通过labelSelector选择刚才部署的Mysql-Exporter，由于Operator在部署Prometheus的时候默认指定了Prometheus选择label为： `prometheus: kube-prometheus` 的ServiceMonitor，所以只需要在ServiceMonitor上打上 `prometheus: kube-prometheus` 标签就可以被Prometheus选择了，完成以上两步就完成了对Mysql的监控，不需要改Prometheus Server配置文件，也不需要重启Prometheus Server服务，是不是很方便，Operator观察到ServiceMonitor发生变化，会动态生成Prometheus Server配置文件，并保证配置文件实时生效。

> **注意1：** Prometheus Operator官方提供的架构图通过Prometheus和ServiceMonitor这两个CRD资源来部署和管理Prometheus Server，而Prometheus Operator中提供的CRD远大于2个，截止当前 Prometheus Operator 共有8个CRD资源，通过Prometheus Operator CRD不仅可以部署和管理Prometheus Server，还可以部署和管理其他相关的监控组件，比如Alertmanager和ThanosRuler。
>
> **注意2：**可以通过以下命令查看prometheus.yaml.gz中存放的Prometheus Server的配置文件内容。

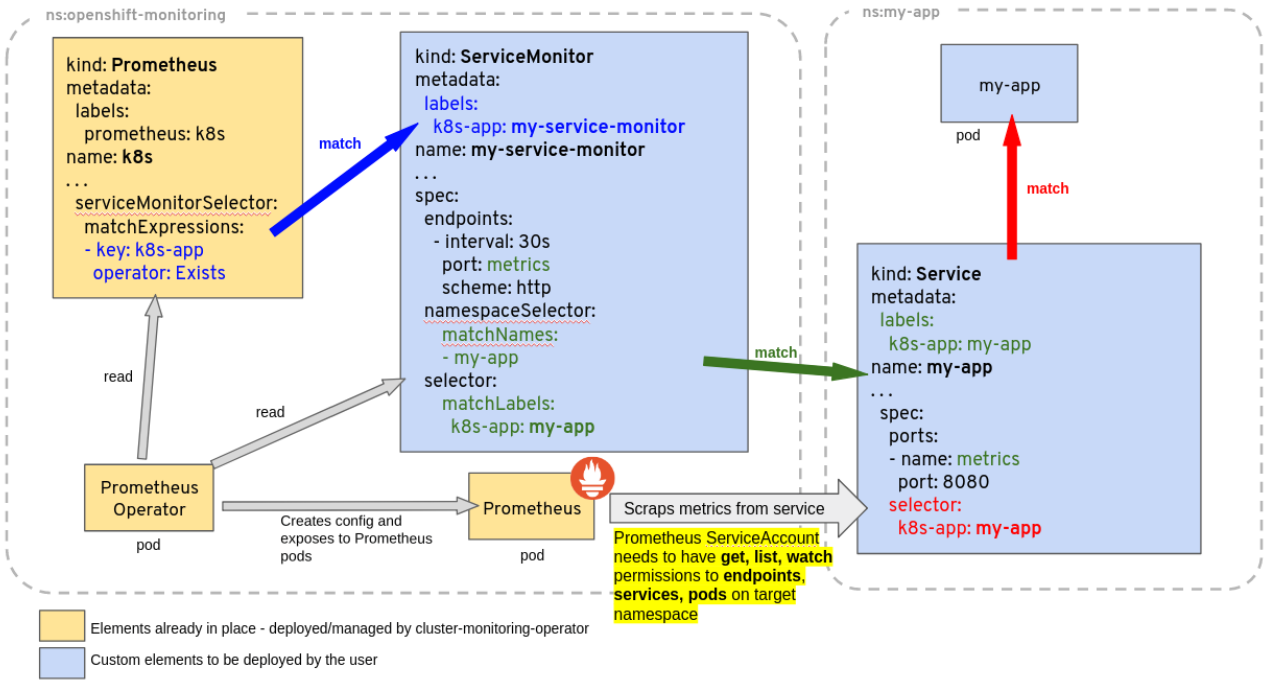
```
kubectl get secrets -n=命名空间 prometheus-k8s -o json| jq -r '.data."prometheus.yaml.gz"' | base64 -d | gzip -d
#先取出 data.“xxx.xxx.gz” 的值，再做 base64 解密和 gzip 还原，得到最终配置文件
```

4、自定义资源

截止当前，Prometheus Operator对以下自定义资源定义 (CRD) 进行操作：

- prometheuses.monitoring.coreos.com
- servicemonitors.monitoring.coreos.com
- podmonitors.monitoring.coreos.com
- prometheusrules.monitoring.coreos.com
- alertmanagers.monitoring.coreos.com
- alertmanagerconfigs.monitoring.coreos.com
- thanosrulers.monitoring.coreos.com
- probes.monitoring.coreos.com

Prometheus Operator 官方提供根据Prometheus Operator CRD自动管理Prometheus Server的工作机制原理图如下：



ServiceMonitor 可以通过 labelSelector 的方式去匹配一类 Service，Prometheus 也可以通过 labelSelector 去匹配多个ServiceMonitor。Prometheus Operator 自动检测 Kubernetes API 服务器对上述任何对象的更改，并确保匹配的部署和配置保持同步。

4.1 prometheuses.monitoring.coreos.com

Prometheus 自定义资源（CRD）声明了在 Kubernetes 集群中运行的 Prometheus 的期望设置。包含了副本数量，持久化存储，serviceMonitorSelector，podMonitorSelector，ruleSelector 以及 Prometheus 实例发送警告到的 Alertmanagers等配置选项。

每一个 Prometheus 资源，Operator 都会在相同 namespace 下部署成一个正确配置的 StatefulSet，Prometheus 的 Pod 都会挂载一个名为 <prometheus-name> 的 Secret，里面包含了 Prometheus 的配置。Operator 根据包含的 ServiceMonitor/PodMonitor等CRD资源对象生成配置，并且更新含有配置的 Secret。无论是对 ServiceMonitors 或者 PodMonitor 的修改，Operator 都会实时更新在Secret里。

一个样例配置如下：

```
apiVersion: monitoring.coreos.com/v1
kind: Prometheus
metadata: # 略。。。。
spec:
  alerting:
    alertmanagers: #Prometheus 对接的 Alertmanager 集群的名字，在 monitor 这个 namespace 中
    - apiVersion: v2
      name: prometheus-kube-prometheus-alertmanager
      namespace: monitor
      pathPrefix: /
      port: web
  arbitraryFSAccessThroughSMs: {}
  externalUrl: http://prometheus-dev.rencaiyoujia.cn/
  image: quay.io/prometheus/prometheus:v2.26.0
  logFormat: logfmt
  logLevel: info
  podMonitorNamespaceSelector: {} # podMonitor选择名称空间，空为所有
  podMonitorSelector:  #podMonitor 选择标签， 必须带有这个标签才能被Prometheus 匹配到。
    matchLabels:
      release: prometheus
  portName: web
  replicas: 2  # 定义该 Proemtheus “集群”有两个副本，说是集群，其实 Prometheus 自身不带集群功能，这里只是起两个完全一样的 Prometheus 来避免单点故障
  resources: {}
```

```
  retention: 10d
  routePrefix: /
  ruleNamespaceSelector: {} # PrometheusRule 选择的名称空间，空为所有
  ruleSelector: #  PrometheusRule 必须带有这两个标签才能被 Prometheus 匹配到。
    matchLabels:
      app: kube-prometheus-stack
      release: prometheus
  rules:
    alert: {}
  securityContext:
    fsGroup: 2000
    runAsGroup: 2000
    runAsNonRoot: true
    runAsUser: 1000
  serviceAccountName: prometheus-kube-prometheus-prometheus
  serviceMonitorNamespaceSelector: {}  # serviceMonitor 选择的名称空间，空为所有
  serviceMonitorSelector: #  serviceMonitor 必须带有这个标签才能被 Prometheus 匹配到。
    matchLabels:
      release: prometheus
  version: v2.26.0
```

4.2 servicemonitors.monitoring.coreos.com

ServiceMonitor 也是一个自定义资源，它描述了一组被 Prometheus 监控的 targets 列表。该资源通过 Labels 来选取对应的 Service Endpoint，让 Prometheus Server 通过选取的 Service 来获取 Metrics 信息。

Operator 能够动态更新 Prometheus 的 Target 列表，ServiceMonitor 就是 Target 的抽象。比如想监控 Kubernetes Scheduler，用户可以创建一个与 Scheduler Service 相映射的 ServiceMonitor 对象。Operator 则会发现这个新的 ServiceMonitor，并将 Scheduler 的 Target 添加到 Prometheus 的监控列表中。

要想使用 Prometheus Operator 监控 Kubernetes 集群中的应用，Endpoints 对象必须存在。Endpoints 对象本质是一个 IP 地址列表。通常，Endpoints 对象由 Service 构建。Service 对象通过对象选择器发现 Pod 并将它们添加到 Endpoints 对象中。

Prometheus Operator 引入 ServiceMonitor 对象，它发现 Endpoints 对象并配置 Prometheus 去监控这些 Pods。

ServiceMonitorSpec 的 endpoints 部分用于配置需要收集 metrics 的 Endpoints 的端口和其他参数。在一些用例中会直接监控不在服务 endpoints 中的 pods 的端口。因此，在 endpoints 部分指定 endpoint 时，请严格使用，不要混淆。

> 注意：endpoints（小写）是 ServiceMonitor CRD 中的一个字段，而 Endpoints（大写）是 Kubernetes 资源类型。

ServiceMonitor 和发现的目标可能来自任何 namespace。这对于跨 namespace 的监控十分重要，比如 meta-monitoring。使用 Prometheus.spec 下 ServiceMonitorNamespaceSelector, 通过各自 Prometheus server 限制 ServiceMonitors 作用 namespece。使用 ServiceMonitor.spec 下的 namespaceSelector 可以现在允许发现 Endpoints 对象的命名空间。要发现所有命名空间下的目标，namespaceSelector 必须为空。

一个样例配置如下：

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor #自定义资源
metadata:
  annotations:
    meta.helm.sh/release-name: prometheus
    meta.helm.sh/release-namespace: monitor
  creationTimestamp: "2021-04-19T08:12:35Z"
  generation: 1
  labels:
    app: kube-prometheus-stack-alertmanager
    app.kubernetes.io/managed-by: Helm
    chart: kube-prometheus-stack-14.9.0
    heritage: Helm
    release: prometheus  #必须带有此标签才能被prometheus 选择到，在prometheus 自定义资源中配置。
  name: prometheus-kube-prometheus-alertmanager
  namespace: monitor
  resourceVersion: "57106409"
  selfLink: /apis/monitoring.coreos.com/v1/namespaces/monitor/servicemonitors/prometheus-kube-prometheus-alertmanager
  uid: 06d6df58-ebc5-4e6d-afd8-d551afe8ad4e
```

```
spec:
  endpoints:
  - path: /metrics
    port: web
  namespaceSelector:
    matchNames:
    - monitor
  selector: # 要监控的 Endpoints  必须带有以下标签。
    matchLabels:
      app: kube-prometheus-stack-alertmanager
      release: prometheus
      self-monitor: "true"
```

4.3 alertmanagers.monitoring.coreos.com

Alertmanager 自定义资源(CRD)声明在 Kubernetes 集群中运行的 Alertmanager 的期望设置。它也提供了配置副本集和持久化存储的选项。Alertmanager自定义资源在Kubernetes中以StatefulSet运行由Operator创建和Operator同一namespace。Alertmanager Pod 都会挂载一个名为 的Secret，该Secret是 Alertmanager 的相关配置，Operator 根据相关配置会实时更新相关Secret。

一个样例配置如下：

```
apiVersion: monitoring.coreos.com/v1
 kind: Alertmanager #自定义资源
 metadata:
   annotations:
     meta.helm.sh/release-name: prometheus
     meta.helm.sh/release-namespace: monitor
   creationTimestamp: "2021-04-19T08:12:34Z"
   generation: 2
   labels:
     app: kube-prometheus-stack-alertmanager
     app.kubernetes.io/managed-by: Helm
     chart: kube-prometheus-stack-14.9.0
```

```
      heritage: Helm
      release: prometheus
    name: prometheus-kube-prometheus-alertmanager
    namespace: monitor
    resourceVersion: "57305495"
    selfLink: /apis/monitoring.coreos.com/v1/namespaces/monitor/alertmanagers/prometheus-kube-prometheus-alertmanager
    uid: aa997233-0341-413f-adde-45746e43ccdc
  spec:
    alertmanagerConfigNamespaceSelector: {}
    alertmanagerConfigSelector: {}
    externalUrl: http://alertmanager-dev.rencaiyoujia.cn/
    image: quay.io/prometheus/alertmanager:v0.21.0
    listenLocal: false
    logFormat: logfmt
    logLevel: info
    paused: false
    portName: web
    replicas: 1
    retention: 120h
    routePrefix: /
    securityContext:
      fsGroup: 2000
      runAsGroup: 2000
      runAsNonRoot: true
      runAsUser: 1000
    serviceAccountName: prometheus-kube-prometheus-alertmanager
    version: v0.21.0
```

4.4 prometheusrules.monitoring.coreos.com

PrometheusRule CRD 能够定义了一组所需的 Prometheus 警报或记录规则。Alerts 和 recording rules 可以保存并应用为 yaml 文件，可以被动态加载而不需要重启。

一个样例配置如下：

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule #自定义资源
metadata:
  labels: ## 必须带有以下标签才能被prometheus 选择到，在prometheus 自定义资源中配置。
    app: kube-prometheus-stack
    release: prometheus
  name: disk-free-rules
  namespace: monitor
spec:
  groups:
  - name: disk
    rules: # 定义了一组报警规则，
    - alert: diskFree
      annotations:
        summary: "{{ $labels.job }}  项目实例 {{ $labels.instance }} 磁盘使用率大于 80%"
        description: "{{ $labels.instance }}  {{ $labels.mountpoint }}  磁盘使用率大于80%  (当前的值: {{ $value }}%),请及时处理"
      expr: |
        (1-(node_filesystem_free_bytes{fstype=~"ext4|xfs",mountpoint!="/boot"} / node_filesystem_size_bytes{fstype=~"ext4|xfs",mountpoint!="
      for: 3m
      labels:
        level: disaster
        severity: warning
```

4.5 其他CRD资源

以上介绍了比较常用的4个Prometheus Operator CRD资源，下面简要说下Prometheus Operator提供的另外4个CRD资源。

- `PodMonitor`：Prometheus Operator 通过 `PodMonitor` 和 `ServiceMonitor` 实现对资源的监控，`PodMonitor` 用于对 Pod 进行监控，推荐首选 `ServiceMonitor`。`PodMonitor` 声明性地指定了应该如何监视一组 pod。Operator 根据 API 服务器中对象的当前状态自动生成 Prometheus 刮擦配置。
- `Probe`：它声明性地指定了应该如何监视 ingress 或静态目标组。Operator 根据定义自动生成 Prometheus 刮擦配置。

- `AlertmanagerConfig`：用于管理 AlertManager 配置文件，主要是告警发给谁；它声明性地指定 Alertmanager 配置的子部分，允许将警报路由到自定义接收器，并设置禁止规则。
- `ThanosRuler`：它定义了 ThanosRuler 期望的部署；如果有多个 Prometheus 实例，则通过 `ThanosRuler` 进行告警规则的统一管理。

> **注意：**第4章节介绍了Prometheus Operator各CRD资源的作用，其中对常用的4个CRD资源进行了示例展示，如果想更加深入的使用Prometheus Operator各CRD资源的话，需要研究Prometheus Operator官方文档及源码。

5、Prometheus Operator 优点

Prometheus Operator 中所有的 API 对象都是 CRD 中定义好的 Schema，API Server会校验。当开发者使用 ConfigMap 保存配置没有任何校验，配置文件写错时，自表现为功能不可用，问题排查复杂。在 Prometheus Operator 中，所有在 Prometheus 对象、ServiceMonitor 对象、PrometheusRule 对象中的配置都是有 Schema 校验的，校验失败 apply 直接出错，这就大大降低了配置异常的风险。

Prometheus Operator 借助 K8S 将 Prometheus 服务平台化。有了 Prometheus、Thanos Ruler、AlertManager 这样的CRD资源对象，非常简单、快速的可以在 K8S 集群中创建和管理 Prometheus 服务、Thanos Ruler 服务和 AlertManager 服务，以应对不同业务部门，不同领域的监控需求。

ServiceMonitor 和 PrometheusRule 解决了 Prometheus 配置难维护问题，开发者不再需要通过 CI 和 k8s ConfigMap 等手段把配置文件更新到 Pod 内再触发 webhook 热更新，只需要修改这两个对象资源就可以了。

6、总结

Prometheus Operator是一组自定义的CRD资源以及Controller的实现，Prometheus Operator负责监听这些自定义资源的变化，并且根据这些资源的定义自动化的完成Prometheus实例和相关的监控组件的自动化管理工作。

容器云平台通过使用Prometheus Operator简化在Kubernetes下部署和管理Prmetheus的复杂度，其通过prometheuses.monitoring.coreos.com资源声明式创建和管理Prometheus Server实例；其通过servicemonitors.monitoring.coreos.com和podmonitors.monitoring.coreos.com资源声明式的管理监控配置；其通过prometheusrules.monitoring.coreos.com资源进行规则记录，实现对复杂查询的 PromQL 语句的性能优化，提高查询效率。

> **注意：**容器云平台使用的是自研的告警系统，并未接入Alertmanager，因此并没有使用Prometheus Operator提供的告警通知相关的CRD资源声明式

> 的管理告警通知配置。

参考：prometheus-book

参考：https://github.com/prometheus-operator/prometheus-operator

参考：kube-prometheus 监控kubernetes集群

参考：Prometheus-operator 介绍和配置解析

Prometheus-operator自定义监控ServiceMonitor

一、ServiceMonitor

1、介绍

用于监控指定的服务状态

感觉ServiceMonitor和PodMonitor差不多。

我这里使用的是Prometheus-operator，所以就直接使用Prometheus-operator的Prometheus

2、创建一个用于监控的测试项目

```
[root@master monitor]# cat ServiceMonitor_test_dep.yaml
kind: Service
apiVersion: v1
metadata:
  name: example-app
  labels:
    app: example-app
spec:
  selector:
    app: example-app
  ports:
  - name: web
    port: 80
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: example-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: example-app
  template:
    metadata:
      labels:
        app: example-app
```

```
    spec:
      containers:
      - name: example-app
        image: nginx:alpine
        ports:
        - name: web
          containerPort: 80
```

3、查看

```
[root@master monitor]# kubectl get ep -l app=example-app
NAME          ENDPOINTS           AGE
example-app   10.244.167.179:80   60m
[root@master monitor]# curl  10.244.167.179:80 -I
HTTP/1.1 200 OK
Server: nginx/1.17.10
Date: Thu, 11 Jun 2020 02:31:14 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Tue, 14 Apr 2020 14:46:22 GMT
Connection: keep-alive
ETag: "5e95ccbe-264"
Accept-Ranges: bytes
```

4、创建ServiceMonitor

```
[root@master monitor]# cat   ServiceMonitor_test.yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: monitor-example-app
  namespace: default
  labels:
    release: mypro  #Prometheus所选择的标签
spec:
  namespaceSelector: #监控的pod所在名称空间
    matchNames:
    - default
  selector:  #选择监控endpoint的标签
    matchLabels:
      app: example-app
  endpoints:
  - port: web #service中对应的端口名称
```

5、浏览器查看prometheus的Targets监控