Prometheus监控Kubernetes系列2——监控部署

原创： 罗佳豪 ServiceMesher 昨天

作者：罗佳豪 审校：孙海洲、李征、吴钧泽

这是该系列连载的第二篇，该系列文章：

Prometheus监控Kubernetes系列1——监控框架

由于容器化和微服务的大力发展，Kubernetes基本已经统一了容器管理方案，当我们使用Kubernetes来进行容器化管理的时候，全面监控Kubernetes也就成了我们第一个需要探索的问题。我们需要监控kubernetes的ingress、service、deployment、pod......等等服务，以达到随时掌握Kubernetes集群的内部状况。此文章是Prometheus监控系列的第二篇，基于上一篇讲解了怎么对Kubernetes集群实施Prometheus监控。

K8s编排文件可参考 https://github.com/xianyuLuo/prometheus-monitor-kubernetes

Prometheus部署

在k8s上部署Prometheus十分简单，下面给的例子中将Prometheus部署到prometheus命名空间。

部署—数据采集

将kube-state-metrics和prometheus分开部署，先部署prometheus。

Prometheus

prometheus-rbac.yaml

```yaml
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
  name: prometheus
rules:
- apiGroups: [""]
  resources:
  - nodes
  - nodes/proxy
  - services
  - endpoints
  - pods
  verbs: ["get", "list", "watch"]
- apiGroups:
  - extensions
  resources:
  - ingresses
  verbs: ["get", "list", "watch"]
- nonResourceURLs: ["/metrics"]
  verbs: ["get"]
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: prometheus
  namespace: prometheus
---
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: prometheus
```

prometheus.rbac.yml定义了Prometheus容器访问k8s apiserver所需的ServiceAccount、ClusterRole以及ClusterRoleBinding。

---

prometheus-config-configmap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: prometheus
data:
  prometheus.yml: |
    global:
      scrape_interval:     15s
      evaluation_interval: 15s
    scrape_configs:

    - job_name: 'kubernetes-apiservers'
      kubernetes_sd_configs:
      - role: endpoints
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - source_labels: [__meta_kubernetes_namespace, __meta_kubernetes_service_name, __meta_kubernetes_endpoint_port_name]
        action: keep
        regex: default;kubernetes;https

    - job_name: 'kubernetes-nodes'
```

```
      kubernetes_sd_configs:
      - role: node
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - action: labelmap
        regex: __meta_kubernetes_node_label_(.+)
      - target_label: __address__
        replacement: kubernetes.default.svc:443
      - source_labels: [__meta_kubernetes_node_name]
        regex: (.+)
        target_label: __metrics_path__
        replacement: /api/v1/nodes/${1}/proxy/metrics

    - job_name: 'kubernetes-cadvisor'
      kubernetes_sd_configs:
      - role: node
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - action: labelmap
        regex: __meta_kubernetes_node_label_(.+)
      - target_label: __address__
        replacement: kubernetes.default.svc:443
      - source_labels: [__meta_kubernetes_node_name]
        regex: (.+)
        target_label: __metrics_path__
        replacement: /api/v1/nodes/${1}/proxy/metrics/cadvisor

    - job_name: 'kubernetes-service-endpoints'
      kubernetes_sd_configs:
      - role: endpoints
      relabel_configs:
```

```yaml
      - 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

    - job_name: 'kubernetes-services'
      kubernetes_sd_configs:
      - role: service
      metrics_path: /probe
      params:
        module: [http_2xx]
      relabel_configs:
```

```
      - source_labels: [__meta_kubernetes_service_annotation_prometheus_io_probe]
        action: keep
        regex: true
      - source_labels: [__address__]
        target_label: __param_target
      - target_label: __address__
        replacement: blackbox-exporter.example.com:9115
      - source_labels: [__param_target]
        target_label: instance
      - action: labelmap
        regex: __meta_kubernetes_service_label_(.+)
      - source_labels: [__meta_kubernetes_namespace]
        target_label: kubernetes_namespace
      - source_labels: [__meta_kubernetes_service_name]
        target_label: kubernetes_name

    - job_name: 'kubernetes-ingresses'
      kubernetes_sd_configs:
      - role: ingress
      relabel_configs:
      - source_labels: [__meta_kubernetes_ingress_annotation_prometheus_io_probe]
        action: keep
        regex: true
      - source_labels: [__meta_kubernetes_ingress_scheme,__address__,__meta_kubernetes_ingress_path]
        regex: (.+);(.+);(.+)
        replacement: ${1}://${2}${3}
        target_label: __param_target
      - target_label: __address__
        replacement: blackbox-exporter.example.com:9115
      - source_labels: [__param_target]
        target_label: instance
      - action: labelmap
        regex: __meta_kubernetes_ingress_label_(.+)
      - source_labels: [__meta_kubernetes_namespace]
```

```
        target_label: kubernetes_namespace
      - source_labels: [__meta_kubernetes_ingress_name]
        target_label: kubernetes_name

    - job_name: 'kubernetes-pods'
      kubernetes_sd_configs:
      - role: pod
      relabel_configs:
      - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_scrape]
        action: keep
        regex: true
      - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_path]
        action: replace
        target_label: __metrics_path__
        regex: (.+)
      - source_labels: [__address__, __meta_kubernetes_pod_annotation_prometheus_io_port]
        action: replace
        regex: ([^:]+)(?::\d+)?;(\d+)
        replacement: $1:$2
        target_label: __address__
      - action: labelmap
        regex: __meta_kubernetes_pod_label_(.+)
      - source_labels: [__meta_kubernetes_namespace]
        action: replace
        target_label: kubernetes_namespace
      - source_labels: [__meta_kubernetes_pod_name]
        action: replace
        target_label: kubernetes_pod_name
```

prometheus-config-configmap.yaml定义了prometheus的配置文件，以configmap的形式使用。

---

prometheus-dep.yaml

```
apiVersion: apps/v1beta2
kind: Deployment
```

```yaml
metadata:
  name: prometheus-dep
  namespace: prometheus
spec:
  replicas: 1
  selector:
    matchLabels:
      app: prometheus-dep
  template:
    metadata:
      labels:
        app: prometheus-dep
    spec:
      containers:
      - image: prom/prometheus:v2.3.2
        name: prometheus
        command:
        - "/bin/prometheus"
        args:
        - "--config.file=/etc/prometheus/prometheus.yml"
        - "--storage.tsdb.path=/prometheus"
        - "--storage.tsdb.retention=1d"
        ports:
        - containerPort: 9090
          protocol: TCP
        volumeMounts:
        - mountPath: "/prometheus"
          name: data
        - mountPath: "/etc/prometheus"
          name: config-volume
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
          limits:
            cpu: 500m
            memory: 2500Mi
      serviceAccountName: prometheus
      imagePullSecrets:
        - name: regsecret
      volumes:
      - name: data
        emptyDir: {}
      - name: config-volume
        configMap:
          name: prometheus-config
```

prometheus-dep.yaml定义了prometheus的部署，这里使用--storage.tsdb.retention参数，监控数据只保留1天，因为最终监控数据会统一汇总。

limits资源限制根据集群大小进行适当调整。

---

prometheus-svc.yaml

```
kind: Service
apiVersion: v1
metadata:
  name: prometheus-svc
  namespace: prometheus
spec:
  type: NodePort
  ports:
  - port: 9090
    targetPort: 9090
    nodePort: 30090
  selector:
    app: prometheus-dep
```

prometheus-svc.yaml定义Prometheus的Service，需要将Prometheus以NodePort、LoadBalancer或Ingress暴露到集群外部，这样外部的Prometheus才能访问它。这里采用的NodePort，所以只需要访问集群中有外网地址的任意一台服务器的30090端口就可以使用prometheus。

kube-state-metrics

prometheus部署成功后，接着再部署kube-state-metrics作为prometheus的一个exporter来使用，提供deployment、daemonset、cronjob等服务的监控数据。

kube-state-metrics-rbac.yaml

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: kube-state-metrics
  namespace: prometheus
---

apiVersion: rbac.authorization.k8s.io/v1
# kubernetes versions before 1.8.0 should use rbac.authorization.k8s.io/v1beta1
kind: Role
metadata:
  namespace: prometheus
  name: kube-state-metrics-resizer
rules:
- apiGroups: [""]
  resources:
  - pods
  verbs: ["get"]
- apiGroups: ["extensions"]
  resources:
  - deployments
  resourceNames: ["kube-state-metrics"]
  verbs: ["get", "update"]
---

apiVersion: rbac.authorization.k8s.io/v1
# kubernetes versions before 1.8.0 should use rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
  name: kube-state-metrics
rules:
- apiGroups: [""]
  resources:
  - configmaps
  - secrets
  - nodes
  - pods
  - services
  - resourcequotas
  - replicationcontrollers
  - limitranges
  - persistentvolumeclaims
  - persistentvolumes
  - namespaces
  - endpoints
  verbs: ["list", "watch"]
- apiGroups: ["extensions"]
  resources:
  - daemonsets
  - deployments
  - replicasets
  verbs: ["list", "watch"]
- apiGroups: ["apps"]
  resources:
  - statefulsets
  verbs: ["list", "watch"]
- apiGroups: ["batch"]
  resources:
  - cronjobs
  - jobs
  verbs: ["list", "watch"]
- apiGroups: ["autoscaling"]
  resources:
  - horizontalpodautoscalers
  verbs: ["list", "watch"]
---

apiVersion: rbac.authorization.k8s.io/v1
# kubernetes versions before 1.8.0 should use 
rbac.authorization.k8s.io/v1beta1
kind: RoleBinding
metadata:
  name: kube-state-metrics
  namespace: prometheus
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: kube-state-metrics-resizer
subjects:
- kind: ServiceAccount
  name: kube-state-metrics
  namespace: prometheus
---

apiVersion: rbac.authorization.k8s.io/v1
# kubernetes versions before 1.8.0 should use rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: kube-state-metrics
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: kube-state-metrics
subjects:
- kind: ServiceAccount
  name: kube-state-metrics
  namespace: prometheus
```

kube-state-metrics-rbac.yaml定义了kube-state-metrics访问k8s apiserver所需的ServiceAccount和ClusterRole及ClusterRoleBinding。

---

kube-state-metrics-dep.yaml

```
apiVersion: apps/v1beta2
# Kubernetes versions after 1.9.0 should use apps/v1
# Kubernetes versions before 1.8.0 should use apps/v1beta1 or extensions/v1beta1
# addon-resizer描述: https://github.com/kubernetes/autoscaler/tree/master/addon-resizer
kind: Deployment
```

```yaml
metadata:
  name: kube-state-metrics
  namespace: prometheus
spec:
  selector:
    matchLabels:
      k8s-app: kube-state-metrics
  replicas: 1
  template:
    metadata:
      labels:
        k8s-app: kube-state-metrics
    spec:
      serviceAccountName: kube-state-metrics
      containers:
      - name: kube-state-metrics
        image: xianyuluo/kube-state-metrics:v1.3.1
        ports:
        - name: http-metrics
          containerPort: 8080
        - name: telemetry
          containerPort: 8081
        readinessProbe:
          httpGet:
            path: /healthz
            port: 8080
          initialDelaySeconds: 5
          timeoutSeconds: 5
      - name: addon-resizer
        image: xianyuluo/addon-resizer:1.7
        resources:
          limits:
            cpu: 100m
            memory: 30Mi
          requests:
            cpu: 100m
            memory: 30Mi
        env:
          - name: MY_POD_NAME
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
          - name: MY_POD_NAMESPACE
            valueFrom:
              fieldRef:
                fieldPath: metadata.namespace
        command:
          - /pod_nanny
          - --container=kube-state-metrics
          - --cpu=100m
          - --extra-cpu=1m
          - --memory=100Mi
          - --extra-memory=2Mi
          - --threshold=5
          - --deployment=kube-state-metrics
```

kube-state-metrics-dep.yaml定义了kube-state-metrics的部署。

---

kube-state-metrics-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: kube-state-metrics
  namespace: prometheus
  labels:
    k8s-app: kube-state-metrics
  annotations:
    prometheus.io/scrape: 'true'
spec:
  ports:
  - name: http-metrics
    port: 8080
    targetPort: http-metrics
    protocol: TCP
  - name: telemetry
    port: 8081
    targetPort: telemetry
    protocol: TCP
  selector:
    k8s-app: kube-state-metrics
```

kube-state-metrics-svc.yaml定义了kube-state-metrics的暴露方式，这里只需要使用默认的ClusterIP就可以了，因为它只提供给集群内部的promethes访问。

k8s集群中的prometheus监控到这儿就已经全部OK了，接下来还需要做的是汇总数据、展示数据及告警规则配置。

部署—数据汇总

prometheus-server

prometheus-server和前面prometheus的步骤基本相同，需要针对configmap、数据存储时间（一般为30d）、svc类型做些许改变，同时增加 rule.yaml。prometheus-server不需要kube-state-metrics。prometheus-server可以部署在任意k8s集群，或者部署在K8s集群外部都可以。

prometheus-rbac.yaml (内容和上面的一致，namespace为prometheus-server)

```
......
```

---

prometheus-server-config-configmap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-server-config
  namespace: prometheus-server
data:
  prometheus.yml: |
    global:
      scrape_interval:     30s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
      evaluation_interval: 30s # Evaluate rules every 15 seconds. The default is every 1 minute.
      scrape_timeout: 30s

    # Alertmanager configuration
    alerting:
      alertmanagers:
      - static_configs:
```

```
    - targets: ["x.x.com:80"]

rule_files:
 - "/etc/prometheus/rule.yml"

scrape_configs:
 - job_name: 'federate-k8scluster-1'
   scrape_interval: 30s
   honor_labels: true
   metrics_path: '/federate'
   params:
    'match[]':
     - '{job=~"kubernetes-.*"}'
   static_configs:
    - targets: ['x.x.x.x:30090']
     labels:
      k8scluster: xxxx-k8s
 - job_name: 'federate-k8scluster-2'
   scrape_interval: 30s
   honor_labels: true
   metrics_path: '/federate'
   params:
    'match[]':
     - '{job=~"kubernetes-.*"}'
   static_configs:
    - targets: ['x.x.x.x:30090']
     labels:
      k8scluster: yyyy-k8s
```

global：全局配置。设置了收集数据频率、超时等

alerting：告警配置。指定了prometheus将满足告警规则的信息发送到哪儿？告警规则在rule_files定义

rule_files：定义的告警规则文件

scrape_configs：监控数据刮取配置。定义了2个job，分别是federate-k8scluster-1、federate-k8scluster-2。其中federate-k8scluster-1配置了去x.x.x.x30090采集数据，并且要匹配job名为"kubernetes-"开头。注意下面的

labels，这个是自己定义的，它的作用在于给每一条刮取过来的监控数据都加上一个 **k8scluster: xxxx-k8s** 的Key-Value，xxxx一般指定为项目代码。这样我们可以在多个集群数据中区分该条数据是属于哪一个k8s集群，这对于后面的展示和告警都非常有利。

---

prometheus-server-rule-configmap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-server-rule-config
  namespace: prometheus-server
data:
  rule.yml: |
    groups:
    - name: kubernetes
      rules:
      - alert: PodDown
        expr: kube_pod_status_phase{phase="Unknown"} == 1 or kube_pod_status_phase{phase="Failed"} == 1
        for: 1m
        labels:
          severity: error
          service: prometheus_bot
          receiver_group: "{{ $labels.k8scluster}}_{{ $labels.namespace }}"
        annotations:
          summary: Pod Down
          k8scluster: "{{ $labels.k8scluster}}"
          namespace: "{{ $labels.namespace }}"
          pod: "{{ $labels.pod }}"
          container: "{{ $labels.container }}"

      - alert: PodRestart
        expr: changes(kube_pod_container_status_restarts_total{pod !~ "analyzer.*"}[10m]) > 0
        for: 1m
        labels:
          severity: error
          service: prometheus_bot
          receiver_group: "{{ $labels.k8scluster}}_{{ $labels.namespace }}"
        annotations:
          summary: Pod Restart
          k8scluster: "{{ $labels.k8scluster}}"
          namespace: "{{ $labels.namespace }}"
          pod: "{{ $labels.pod }}"
          container: "{{ $labels.container }}"

      - alert: NodeUnschedulable
        expr: kube_node_spec_unschedulable == 1
        for: 5m
        labels:
          severity: error
          service: prometheus_bot
          receiver_group: "{{ $labels.k8scluster}}_{{ $labels.namespace }}"
        annotations:
          summary: Node Unschedulable
          k8scluster: "{{ $labels.k8scluster}}"
          node: "{{ $labels.node }}"

      - alert: NodeStatusError
        expr: kube_node_status_condition{condition="Ready", status!="true"} == 1
        for: 5m
        labels:
          severity: error
          service: prometheus_bot
          receiver_group: "{{ $labels.k8scluster}}_{{ $labels.namespace }}"
        annotations:
          summary: Node Status Error
          k8scluster: "{{ $labels.k8scluster}}"
          node: "{{ $labels.node }}"

      - alert: DaemonsetUnavailable
        expr: kube_daemonset_status_number_unavailable > 0
        for: 5m
        labels:
          severity: error
          service: prometheus_bot
          receiver_group: "{{ $labels.k8scluster}}_{{ $labels.namespace }}"
        annotations:
          summary: Daemonset Unavailable
          k8scluster: "{{ $labels.k8scluster}}"
          namespace: "{{ $labels.namespace }}"
          daemonset: "{{ $labels.daemonset }}"

      - alert: JobFailed
        expr: kube_job_status_failed == 1
        for: 5m
        labels:
          severity: error
          service: prometheus_bot
          receiver_group: "{{ $labels.k8scluster}}_{{ $labels.namespace }}"
        annotations:
          summary: Job Failed
          k8scluster: "{{ $labels.k8scluster}}"
          namespace: "{{ $labels.namespace }}"
          job: "{{ $labels.exported_job }}"
```

rule.yaml定义了告警规则。此文件中定义了 PodDown、PodRestart、NodeUnschedulable、NodeStatusError、DaemonsetUnavailable、JobFailed 共6条规则。

alert：名称

expr：表达式。prometheus的SQL语句

for：时间范围

annotations：告警消息，其中 {{*}} 为Prometheus内部变量

---

prometheus-server-dep.yaml (参考上面的prometheus-dep.yaml做些许调整)

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: prometheus-server-dep
  namespace: prometheus-server
```

```
spec:
  replicas: 1
  selector:
    matchLabels:
      app: prometheus-server-dep
      ......
        args:
        - "--config.file=/etc/prometheus/prometheus.yml"
        - "--storage.tsdb.path=/prometheus"
        - "--web.console.libraries=/usr/share/prometheus/console_libraries"
        - "--web.console.templates=/usr/share/prometheus"
        - "--storage.tsdb.retention=30d"
        - "--web.enable-lifecycle"
        ......
        volumeMounts:
        - mountPath: "/prometheus"
          name: data
        - mountPath: "/etc/prometheus/prometheus.yml"
          name: server-config-volume
          subPath: prometheus.yml
        - mountPath: "/etc/prometheus/rule.yml"
          name: rule-config-volume
          subPath: rule.yml
        ......
      volumes:
      - name: data
        emptyDir: {}
      - name: server-config-volume
        configMap:
          name: prometheus-server-config
      - name: rule-config-volume
        configMap:
          name: prometheus-server-rule-config
```

volumes.data这里使用的是emptyDir，这样其实不妥，应该单独挂载一块盘来存储汇总数据。可使用pv实现。

---

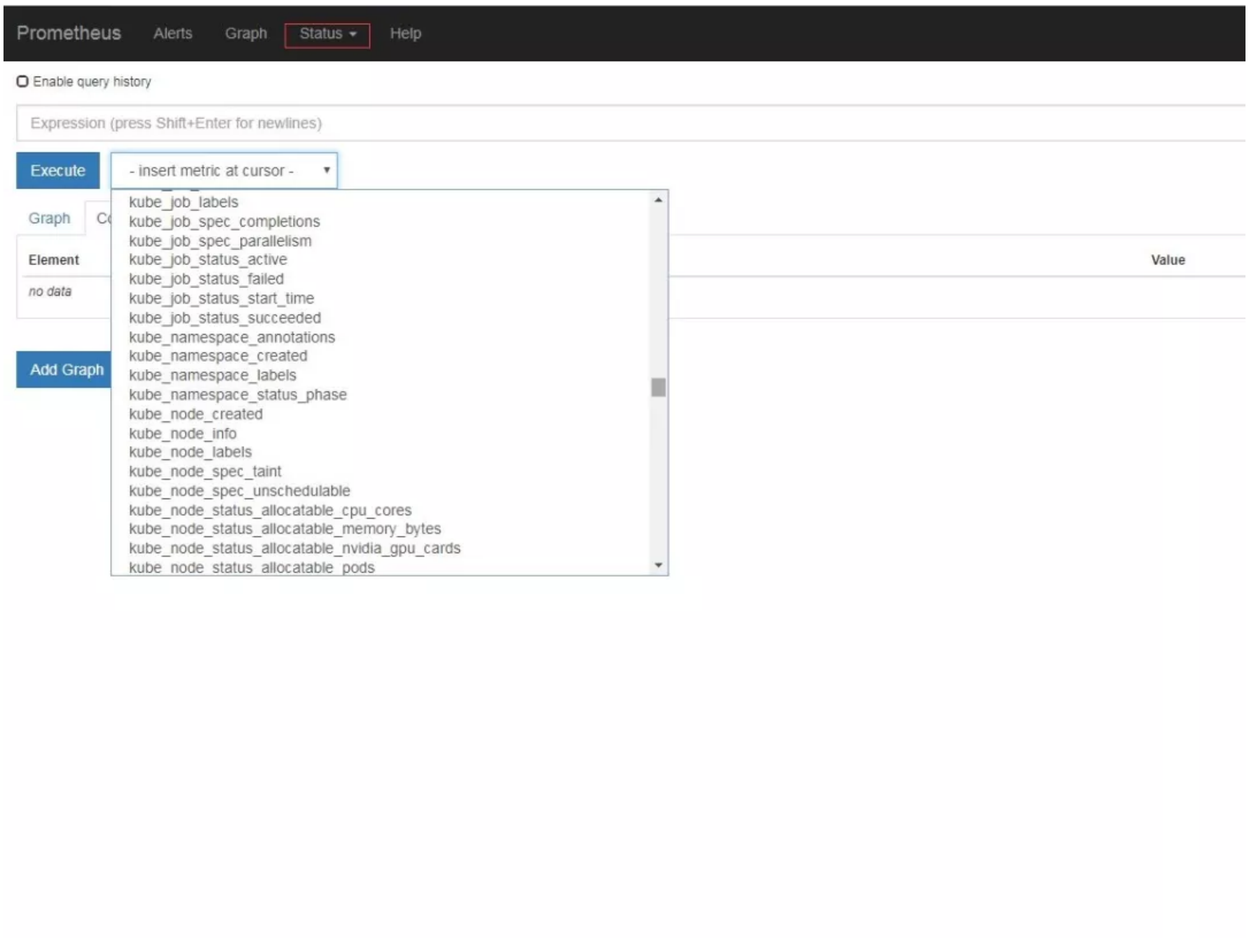
prometheus-server-svc.yaml (参考上面的prometheus-svc.yaml做些许调整)

```
kind: Service
```

```
apiVersion: v1
metadata:
  name: prometheus-server-svc
  namespace: prometheus-server
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 9090
  selector:
    app: prometheus-server-dep
```

到这儿，数据采集和数据汇总就已经OK了。

Prometheus-server部署成功之后，在浏览器中可以看到监控数据汇总信息了



```
kube_job_labels
kube_job_spec_completions
kube_job_spec_parallelism
kube_job_status_active
kube_job_status_failed
kube_job_status_start_time
kube_job_status_succeeded
kube_namespace_annotations
kube_namespace_created
kube_namespace_labels
kube_namespace_status_phase
kube_node_created
kube_node_info
kube_node_labels
kube_node_spec_taint
kube_node_spec_unschedulable
kube_node_status_allocatable_cpu_cores
kube_node_status_allocatable_memory_bytes
kube_node_status_allocatable_nvidia_gpu_cards
kube_node_status_allocatable_pods
```

Status --> Configuration 中可以看到Prometheus-server的配置

Status --> Rules 中可以看到规则文件内容

Status --> Targets 中可以看到刮取目标的状态信息

告警配置

遵循上篇文章中的架构，告警使用Prometheus官方提供的组件Alertmanager

alertmanager-config-configmap.yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: alertmanager-config
  namespace: prometheus
data:
  config.yml: |
    global:
    resolve_timeout: 5m

    route:
      receiver: default
      group_wait: 30s
      group_interval: 5m
      repeat_interval: 4h
      group_by: ['alertname', 'k8scluster', 'node', 'container', 'exported_job', 'daemonset']
      routes:
      - receiver: send_msg_warning
        group_wait: 60s
        match:
          severity: warning

    receivers:
    - name: default
      webhook_configs:
      - url: 'http://msg.x.com/xxx/'
        send_resolved: true
        http_config:
          bearer_token: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'

    - name: send_msg_warning
      webhook_configs:
      - url: 'http://msg.x.com/xxx/'
        send_resolved: true
        http_config:
          bearer_token: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
```

alertmanager-config-configmap.yaml定义了alertmanager的配置文件

route：路由。分级匹配，然后交给指定 receivers，其中route.group_by中的k8scluster是prometheus-server-config.yaml中自定义的标签

receivers：发送。这里使用webhook方式发送给自研的send_msg模块

email、wechat、webhook、slack等发送方式配置请见官网文档：https://prometheus.io/docs/alerting/configuration/

---

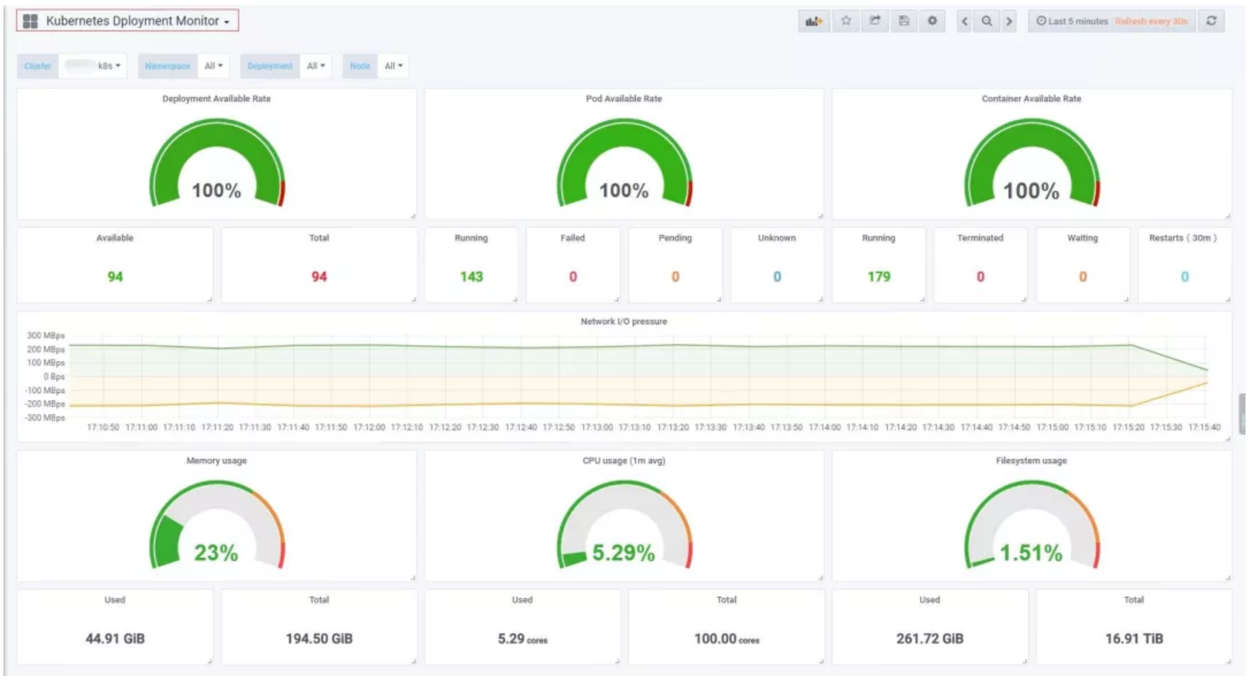
alertmanager-dep.yaml

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: alertmanager-dep
  namespace: alertmanager
spec:
  replicas: 1
  selector:
    matchLabels:
      app: alertmanager-dep
  template:
    metadata:
      labels:
        app: alertmanager-dep
    spec:
      containers:
      - image: prom/alertmanager:v0.15.2
        name: alertmanager
        args:
        - "--config.file=/etc/alertmanager/config.yml"
        - "--storage.path=/alertmanager"
        - "--data.retention=720h"
        volumeMounts:
        - mountPath: "/alertmanager"
          name: data
        - mountPath: "/etc/alertmanager"
```

```
              name: config-volume
          resources:
            requests:
              cpu: 100m
              memory: 100Mi
            limits:
              cpu: 500m
              memory: 2500Mi
        volumes:
        - name: data
          emptyDir: {}
        - name: config-volume
          configMap:
            name: alertmanager-config
```
alertmanager-dep.yaml定义了Alertmanager的部署。

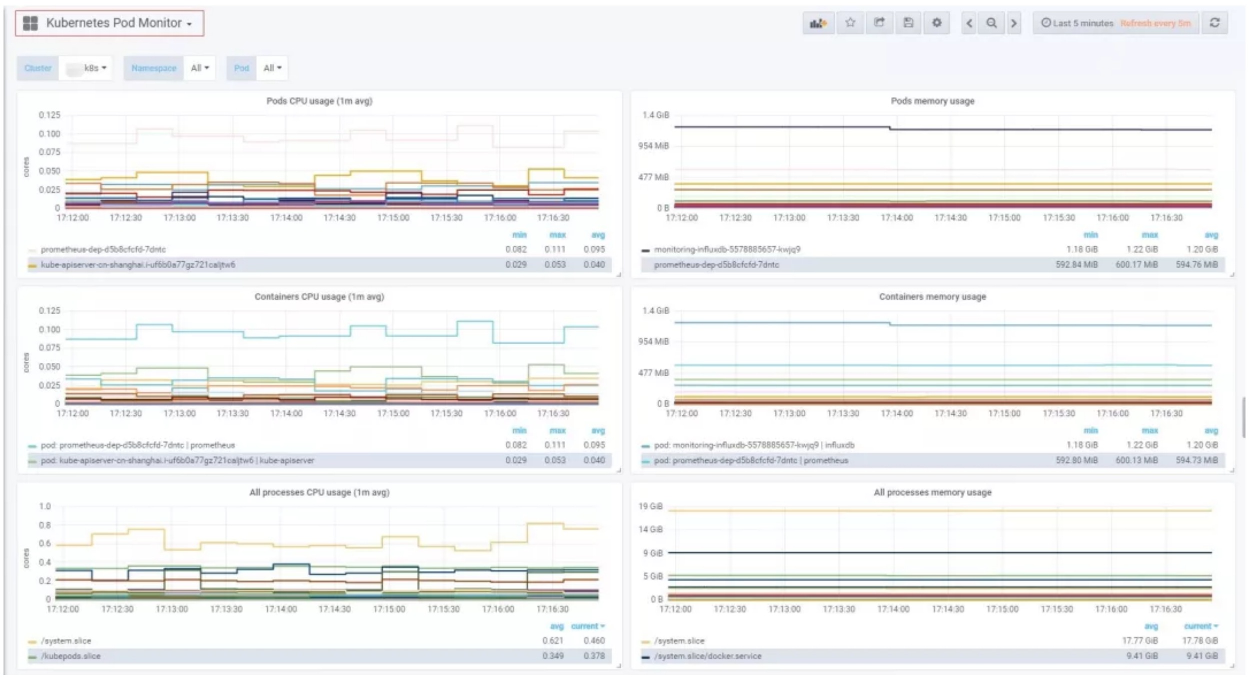
展示

遵循上篇文章中的架构，展示使用开源的Grafana。Grafana的部署方式就不详细描述了，下面展示两个Dashboard



Deployment Available Rate 100%
Pod Available Rate 100%
Container Available Rate 100%
Available 94
Total 94
Running 143
Failed 0
Pending 0
Unknown 0
Running 179
Terminated 0
Waiting 0
Restarts (30m) 0
Memory usage 23%
CPU usage (1m avg) 5.29%
Filesystem usage 1.51%
Used 44.91 GiB
Total 194.50 GiB
Used 5.29 cores
Total 100.00 cores
Used 261.72 GiB
Total 16.91 TiB

kubernetes-deployment-dashboard，展示了大多关于deployment的信息。左上角的Cluster选项就是利用prometheus-server-config.yaml中自定义的labels.k8scluster标签实现的。



Pods CPU usage (1m avg)

	min	max	avg
prometheus-dep-d5b8cfcfd-7dntc	0.082	0.111	0.095
kube-apiserver-cn-shanghai.i-uf6b0a77gz721caljtw6	0.029	0.053	0.040

Pods memory usage

	min	max	avg
monitoring-influxdb-5578885657-kwjq9	1.18 GiB	1.22 GiB	1.20 GiB
prometheus-dep-d5b8cfcfd-7dntc	592.84 MiB	600.17 MiB	594.76 MiB

Containers CPU usage (1m avg)

	min	max	avg
pod: prometheus-dep-d5b8cfcfd-7dntc | prometheus	0.082	0.111	0.095
pod: kube-apiserver-cn-shanghai.i-uf6b0a77gz721caljtw6 | kube-apiserver	0.029	0.053	0.040

Containers memory usage

	min	max	avg
pod: monitoring-influxdb-5578885657-kwjq9 | influxdb	1.18 GiB	1.22 GiB	1.20 GiB
pod: prometheus-dep-d5b8cfcfd-7dntc | prometheus	592.80 MiB	600.13 MiB	594.73 MiB

All processes CPU usage (1m avg)

	avg	current
/system.slice	0.621	0.460
/kubepods.slice	0.349	0.378

All processes memory usage

	avg	current
/system.slice	17.77 GiB	17.78 GiB
/system.slice/docker.service	9.41 GiB	9.41 GiB

kubernetes-pod-dashboard，展示的都是关于pod和container的信息，包括CPU、mem使用监控。此页面数据量一般比较大。左上角的Cluster选项也是利用prometheus-server-config.yaml中自定义的labels.k8scluster做的。

kubernetes-deployment-dashboard下载地址：

https://grafana.com/dashboards/9730

kubernetes-pod-dashboard下载地址：https://grafana.com/dashboards/9729

结束

详细监控Kubernetes集群本身就是一项复杂的工作，好在有Prometheus、Grafana、kube-state-metrics这些优秀的开源工具，才让我们的工作复杂度得以缓解，Thanks。

此文章也是“使用prometheus完美监控kubernetes集群”系列的第二篇，如果在部署过程中遇到问题或者有不理解的地方，欢迎随时后台留言。

推荐阅读

Istio中使用Prometheus进行应用程序指标度量

全手动部署prometheus operator监控kubernetes集群以及一些坑

Envoy service mesh、Prometheus和Grafana下的微服务监控

对使用Kubernetes和Istio管理的基于容器的基础设施进行全面服务监控