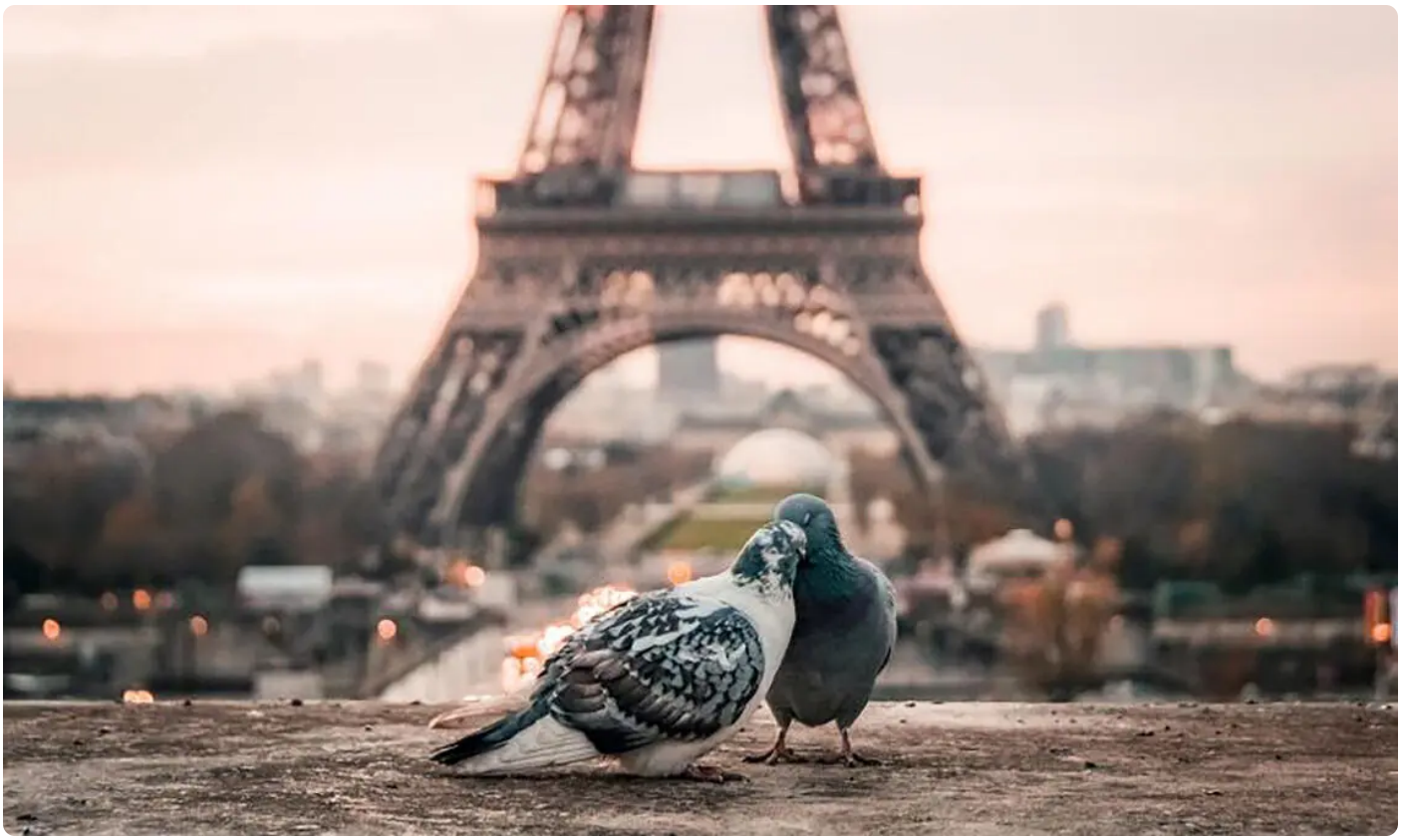
18 | 组件监控：Kubernetes 控制面组件的关键指标与数据采集

2023-02-17 秦晓辉 来自北京

《运维监控系统实战笔记》

课程介绍



讲述：秦晓辉

时长 14:45　大小 13.47M

你好，我是秦晓辉。

上一讲我们介绍了 Kubernetes **工作负载节点**的监控，了解了 Kube-Proxy、Kubelet、容器负载监控的方法。这一讲我们介绍**控制面组件**的监控，包括 APIServer、Controller-manager（简称 CM）、Scheduler、etcd 四个组件，我会讲解这几个组件监控数据的采集方法和关键指标，并给出监控大盘。此外，我们还会学习如何使用 kube-state-metrics（简称 KSM）来监控 Kubernetes 的各类对象。

数据采集

自行搭建 Kubernetes 控制面的朋友，大都是选择 Kubeadm 这样的工具，Kubeadm 会把控制面的组件以静态容器的方式放到容器里运行，之后我会重点给你演示在这种部署方式下如何采集监控数据。

不过很多大一些的互联网公司会选择直接使用二进制的方式来部署，因为二进制的方式对于监控数据采集来说其实更简单，直接在采集器里配置要抓取的这几个组件的目标地址就可以了。

如果想要调用 Kubernetes 服务端 APIServer、Controller-manager、Scheduler 这三个组件的 `/metrics` 接口，需要有 Token 做鉴权，我们还是通过创建 ServiceAccount 的方式拿到 Token，后面也会把采集器直接部署到 Kubernetes 容器里，这样 Token 信息就可以自动 mount 到容器里，比较方便。

创建认证信息

相比前面为 Categraf 准备的 ServiceAccount，用于访问控制面组件的 ServiceAccount 会要求更多权限，这里我重新给出一个升级后的 YAML 文件，供你参考。

复制代码

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

```
kind: ClusterRoleBinding
metadata:
  name: categraf
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: categraf
subjects:
- kind: ServiceAccount
  name: categraf
  namespace: flashcat
```

使用 `kubectl apply` 一下这个 YAML 文件即可。有了认证信息，后面就是选型并部署采集器了。

部署采集器

我们希望能够自动感知到组件实例的变化，也就是要抓取的目标地址的变化。毫无疑问要读取 Kubernetes 的元信息，就需要具备类似 Prometheus 的 Kubernetes 服务发现能力。支持这个服务发现能力的采集器，比较常用的是 Telegraf、Prometheus、Categraf、Grafana-agent、vmagent 等，这里最原汁原味的显然是 Prometheus 自身。

Prometheus 从 v2.32.0 开始支持 agent mode 模式，相当于把 Prometheus 当做一个采集 agent 来使用，只负责采集数据，这个玩法最为简便清晰，我们就使用这个方式来采集数据。

agent mode 模式的 Prometheus，重点要配置 scrape 规则和 remote write 地址，我们把配置文件做成 ConfigMap，你可以看一下具体的 YAML 文件。

复制代码

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-agent-conf
  labels:
    name: prometheus-agent-conf
  namespace: flashcat
data:
  prometheus.yml: |-
    global:
      scrape_interval: 15s
      evaluation_interval: 15s

    scrape_configs:
```

```
    - job_name: 'apiserver'
      kubernetes_sd_configs:
      - role: endpoints
      scheme: https
      tls_config:
        insecure_skip_verify: true
      authorization:
        credentials_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - source_labels: [__meta_kubernetes_namespace, __meta_kubernetes_servic
        action: keep
        regex: default;kubernetes;https

    - job_name: 'controller-manager'
      kubernetes_sd_configs:
      - role: endpoints
      scheme: https
      tls_config:
        insecure_skip_verify: true
      authorization:
        credentials_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - source_labels: [__meta_kubernetes_namespace, __meta_kubernetes_servic
        action: keep
        regex: kube-system;kube-controller-manager;https

    - job_name: 'scheduler'
      kubernetes_sd_configs:
      - role: endpoints
      scheme: https
      tls_config:
        insecure_skip_verify: true
      authorization:
        credentials_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - source_labels: [__meta_kubernetes_namespace, __meta_kubernetes_servic
        action: keep
        regex: kube-system;kube-scheduler;https

    - job_name: 'etcd'
      kubernetes_sd_configs:
      - role: endpoints
      scheme: http
      relabel_configs:
      - source_labels: [__meta_kubernetes_namespace, __meta_kubernetes_servic
        action: keep
        regex: kube-system;etcd;http

    - job_name: 'kube-state-metrics'
      kubernetes_sd_configs:
      - role: endpoints
      scheme: http
```

```
        relabel_configs:
        - source_labels: [__meta_kubernetes_namespace, __meta_kubernetes_servic
          action: keep
          regex: kube-system;kube-state-metrics;http-metrics

    remote_write:
        -
```

我来简单介绍一下，这段代码中包含了 5 个抓取 job，分别是 APIServer、Controller-manager、Scheduler、ectd、KSM。前面 3 个走的是 HTTPS ，后面两个走的是 HTTP，重点关注 relabel 规则。keep 的规则实际就是在做过滤，只过滤自己 job 感兴趣的那些 endpoint。最后两行配置了 remote write 地址，采集到的数据通过 remote write 协议推给远端，我这里是推给了 n9e-server，n9e-server 后面是 VictoriaMetrics 集群。

准备好配置文件之后，接下来部署 Prometheus。因为只是抓取几个服务端组件，抓取量不大，不用做分片，我这里把 Prometheus agent mode 做成单副本的 Deployment，你可以看一下我给出的 YAML 文件。

复制代码

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: prometheus-agent
  namespace: flashcat
  labels:
    app: prometheus-agent
spec:
  replicas: 1
  selector:
    matchLabels:
      app: prometheus-agent
  template:
    metadata:
      labels:
        app: prometheus-agent
    spec:
      serviceAccountName: categraf
      containers:
        - name: prometheus
          image: prom/prometheus
          args:
            - "--config.file=/etc/prometheus/prometheus.yml"
            - "--web.enable-lifecycle"
            - "--enable-feature=agent"
          ports:
            - containerPort: 9090
```

```
          resources:
            requests:
              cpu: 500m
              memory: 500M
            limits:
              cpu: 1
              memory: 1Gi
          volumeMounts:
            - name: prometheus-config-volume
              mountPath: /etc/prometheus/
            - name: prometheus-storage-volume
              mountPath: /prometheus/
      volumes:
        - name: prometheus-config-volume
          configMap:
            defaultMode: 420
            name: prometheus-agent-conf
        - name: prometheus-storage-volume
          emptyDir: {}
```

这里要注意的关键点，一个是 serviceAccountName，配置是 categraf，和前面创建的 ServiceAccount 对应，另一个是 args 部分，给出了相关的启动参数，`--enable-feature=agent` 就是作为 agent mode 模式运行。

最后，执行 `kubectl apply -f prometheus-agent-deployment.yaml`，让 Kubernetes 把 Deployment 拉起来就可以了。稍等片刻，去页面上查询一下 APIServer 的监控数据，理论上是可以查到的，但是其他组件的监控数据，大概率是没有的，下面我们来一一修复。

修复 Controller-manager 和 Scheduler

上面的抓取规则走的都是 Kubernetes 服务发现机制，发现 endpoint，然后过滤。我们先通过下面的命令查看一下 kube-system 这个 namespace 下有哪些 endpoint。

复制代码

```
kubectl get endpoints -n kube-system
```

如果有 kube-controller-manager、kube-scheduler 这两个 endpoint，理论上通过上面的抓取规则就可以抓到数据，如果没有的话，我们可以创建相关的 service。

你可以参考我给出的这两个 YAML 文件来创建 service。

- https://github.com/flashcatcloud/categraf/blob/main/k8s/controller-service.yaml
- https://github.com/flashcatcloud/categraf/blob/main/k8s/scheduler-service.yaml

另外就是得确保 Controller-manager 和 Scheduler 没有监听在 127.0.0.1，否则采集器落在其他机器上就访问不通了。具体怎么做呢？

在这两个组件的启动参数里加上 `--bind-address=0.0.0.0` 就可以了。如果是 Kubeadm 安装的，可以在 /etc/kubernetes/manifests/kube-controller-manager.yaml 和 /etc/kubernetes/manifests/kube-scheduler.yaml 里调整参数。调整完之后，理论上就可以抓到数据了。

修复 etcd

etcd 默认端口是 2379，如果从这个端口获取监控数据，就需要有比较复杂的认证鉴权。但其实 etcd 的监控数据也不是什么太关键的信息，而且是内网，直接开放就可以了。etcd 提供了一个启动参数，可以为暴露监控指标单独监听一个地址。

具体参数是：

```
--listen-metrics-urls=http://0.0.0.0:2381
```

之后创建相关的 service ，prometheus agent 就可以发现这个 HTTP 的 endpoint 了。

修复 KSM

KSM 是监控各类 Kubernetes 对象的组件，通过 KSM 我们可以知道 Service、Deployment、Statefulset、Node 等组件的各类元信息，比如某个 Deployment 期望有几个副本、实际有几个 Pod 在运行这种问题，就是靠 KSM 来回答的。KSM 是如何知道这些信息的呢？它需要跟 APIServer 通信，订阅各类资源对象的变更。下面我们安装一下 KSM，相关指标就有了。

```
git clone https://github.com/kubernetes/kube-state-metrics
```

```
kubectl apply -f kube-state-metrics/examples/standard/
```

KSM 在代码仓库里提供了相关的 YAML 文件，我们 clone 下来直接 apply 就可以。KSM 默认暴露了两个端口，8080 用于返回各类 Kubernetes 对象信息，8081 用于返回 KSM 自身的指标，我们在抓取规则里重点抓取的是 8080 的数据。

KSM 要返回所有 Kubernetes 对象的指标，数据量比较大，从 8080 拉取监控数据可能会拉取十几秒甚至几十秒，KSM 为此支持了分片逻辑，examples/standard 下面提供的 YAML 文件是把 KSM 部署为单副本的 Deployment，分片的话使用 Daemonset，每个 Node 上都跑一个 KSM，这个 KSM 只同步与自身节点相关的数据，KSM 的官方 README 里说得很清楚了，你可以看一下 Daemonset 样例。

复制代码

```
apiVersion: apps/v1
kind: DaemonSet
spec:
  template:
    spec:
      containers:
      - image: registry.k8s.io/kube-state-metrics/kube-state-metrics:IMAGE_TAG
        name: kube-state-metrics
        args:
        - --resource=pods
        - --node=$(NODE_NAME)
        env:
        - name: NODE_NAME
          valueFrom:
            fieldRef:
              apiVersion: v1
              fieldPath: spec.nodeName
```

另外，KSM 提供了两种方式来过滤要 watch 的对象类型，一个是通过白名单的方式指定具体要 watch 哪类对象，通过命令行启动参数中的 `--resources=daemonsets,deployments`，表示只 watch daemonsets 和 deployments。虽然已经限制了对象资源类型，但如果采集的某些指标仍然不想要，可以采用黑名单的方式来过滤指标：`--metric-denylist=kube_deployment_spec_.*`。这个过滤规则支持正则写法，多个正则之间可以使用逗号分隔。

做完这些操作之后，我们就可以采集到这些组件的监控数据了，下面我们继续看哪些指标更为关键。

关键指标

Categraf 的代码仓库里已经内置了 Kubernetes 各个组件的监控大盘，只要是出现在监控大盘上的指标，理论上就是相对比较重要的，要不然也没有必要放到大盘上了。你可以看一下 APIServer、Controller-manager、Scheduler、etcd、KSM 的大盘。

如果你使用 Grafana 来做可视化，可以参考下面两个项目中提供的 Dashboard。

- https://github.com/kubernetes-monitoring/kubernetes-mixin
- https://github.com/dotdc/grafana-dashboards-kubernetes

因为所有组件都是 Go 实现的，都暴露了 Go 程序通用的那些 CPU、内存、Goroutine、句柄等指标，这一部分内容我们在上一讲已经介绍过，这里不再赘述。下面我们分别看一下这几个组件一些其他类型的关键指标。

APIServer

APIServer 的核心职能是 Kubernetes 集群的 API 总入口，Kube-Proxy、Kubelet、Controller-Manager、Scheduler 等都需要调用 APIServer，所以 APIServer 的监控，完全按照 RED 方法论来梳理即可，最核心的就是请求吞吐和延迟。

- apiserver_request_total：请求量的指标，可以统计每秒请求数、成功率。
- apiserver_request_duration_seconds：请求耗时的指标。
- apiserver_current_inflight_requests：APIServer 当前处理的请求数，分为 mutating（非 get、list、watch 的请求）和 readOnly（get、list、watch 请求）两种，请求量过大就会被限流，所以这个指标对我们观察容量水位很有帮助。

Controller-manager

Controller-manager 负责监听对象状态，并与期望状态做对比。如果状态不一致则进行调谐，重点关注的是**任务数量、队列深度**等。

- workqueue_adds_total：各个 controller 接收到的任务总数。
- workqueue_depth：各个 controller 的队列深度，表示各个 controller 中的任务的数量，数量越大表示越繁忙。
- workqueue_queue_duration_seconds：任务在队列中的等待耗时，按照控制器分别统计。
- workqueue_work_duration_seconds：任务出队到被处理完成的时间，按照控制器分别统计。
- workqueue_retries_total：任务进入队列的重试次数。

Scheduler

Scheduler 在 Kubernetes 架构中负责把对象调度到合适的 Node 上，在这个过程中会有一系列的规则计算和筛选，重点关注**调度**这个动作的相关指标。

- leader_election_master_status：调度器的选主状态，1 表示 master，0 表示 backup。
- scheduler_queue_incoming_pods_total：进入调度队列的 Pod 数量。
- scheduler_pending_pods：Pending 的 Pod 数量。
- scheduler_pod_scheduling_attempts：Pod 调度成功前，调度重试的次数分布。
- scheduler_framework_extension_point_duration_seconds：调度框架的扩展点延迟分布，按 extension_point 统计。
- scheduler_schedule_attempts_total：按照调度结果统计的尝试次数，“unschedulable”表示无法调度，“error”表示调度器内部错误。

etcd

etcd 在 Kubernetes 的架构中作用巨大，相对也比较稳定，不过 etcd 对硬盘 IO 要求较高，因此需要着重关注 IO 相关的指标，生产环境建议至少使用 SSD 的盘做存储。

- etcd_server_has_leader ：etcd 是否有 leader。
- etcd_server_leader_changes_seen_total：偶尔切主问题不大，频繁切主就要关注了。
- etcd_server_proposals_failed_total：提案失败次数。
- etcd_disk_backend_commit_duration_seconds：提交花费的耗时。

- etcd_disk_wal_fsync_duration_seconds ：wal 日志同步耗时。

KSM

Kube-state-metrics 这个组件，采集的很多指标都只是充当元信息，单独拿出来未必那么有用，但是和其他指标做 group_left、group_right 连接的时候可能又会很有用，还记得第 6 讲介绍的那个按照 version 绘制饼图的例子吗？那就是个典型用法。下面我挑选一些相对常用的指标解释一下。

- kube_node_status_condition：Node 节点状态，状态不正常、有磁盘压力等都可以通过这个指标发现。
- kube_pod_container_status_last_terminated_reason：容器停止原因。
- kube_pod_container_status_waiting_reason：容器处于 waiting 状态的原因。
- kube_pod_container_status_restarts_total：容器重启次数。
- kube_deployment_spec_replicas：deployment 配置期望的副本数。
- kube_deployment_status_replicas_available：deployment 实际可用的副本数。

基于 KSM 数据的比较典型的告警规则，我也举个例子，让你有一个直观的认识。

复制代码

```
# 长时间版本不一致需要告警
kube_deployment_status_observed_generation{job="kube-state-metrics"}
!=
kube_deployment_metadata_generation{job="kube-state-metrics"}

# deployment 副本数不一致
(
kube_deployment_spec_replicas{job="kube-state-metrics"}
!=
kube_deployment_status_replicas_available{job="kube-state-metrics"}
)
and
(
changes(kube_deployment_status_replicas_updated{job="kube-state-metrics"}[5m])
)

# 容器有 Error 或者 OOM 导致的退出
(sum(kube_pod_container_status_last_terminated_reason{reason=~"Error|OOMKilled"
* on(namespace,pod,container)
sum(increase(kube_pod_container_status_restarts_total[10m]) > 0) by(namespace,p
```

上面我只是举了 Deployment 的例子，Statefulset 也是类似的。到这里控制面的核心组件以及 KSM 相关的知识就讲完了，下面我们做个总结。

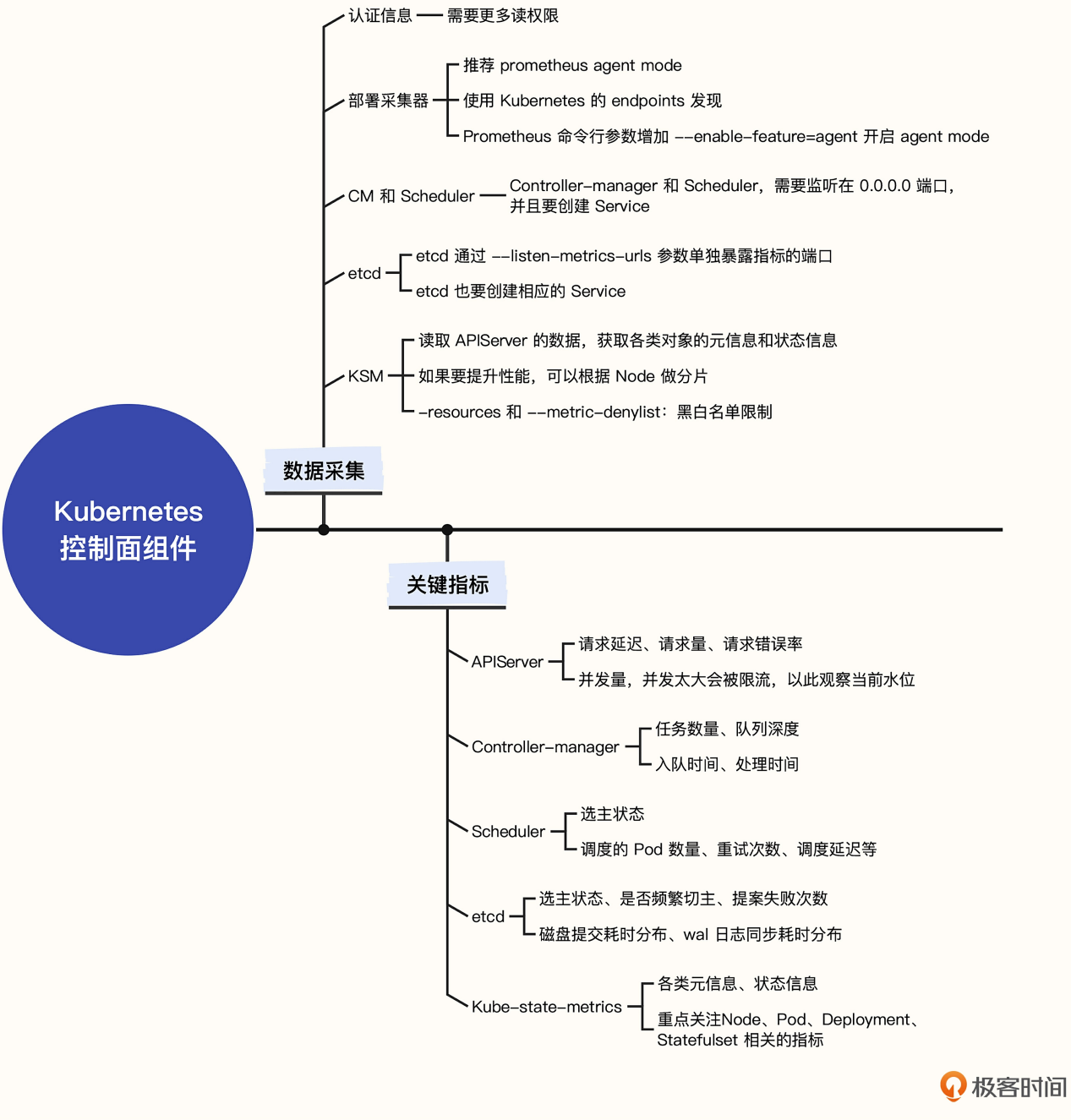
小结

Kubernetes 体系确实非常庞大，这一讲我们重点介绍控制面的组件监控，包括 APIServer、Controller-manager、Scheduler、etcd 等。当然，Kubernetes 对象的监控也很关键，可以使用 KSM 完成。

核心内容主要是两部分，一个是数据采集，一个是关键指标。数据采集我们引入了 prometheus agent mode，支持 Kubernetes 服务发现，非常轻量，通过 remote write 协议把数据推给后端存储。关键指标的话需要看各个模块的核心职能以及重点依赖，比如 Scheduler 是做调度的，那就要看调度相关的指标，etcd 强依赖硬盘，就要多关注硬盘 IO 相关的指标。

Kubernetes 控制面的组件全部都要认证鉴权，相比之前演示的 Kubelet 数据采集，需要给更多的权限。Controller-manager、Scheduler 可能默认没有创建 Service，需要手工创建并且修改监听地址，etcd 要开启 HTTP 协议的指标暴露端口，这些都是坑，需要依次修复。

关键指标部分我提供了一些最为常见的指标说明，你也可以参考监控大盘中的配置，指标既然放到大盘中了，就表示相对重要。不得不说，监控大盘是一个很好的知识沉淀的手段。



互动时刻

在 Prometheus 的抓取配置中，我们给出了几个抓取 Job 的配置。如果一个 Kubernetes 集群的数据只写入一个时序库，这样的配置是没问题，如果多个 Kubernetes 集群的数据都写入一个时序库，就要通过额外的标签来区分了。你知道如何为抓取的数据附加新的标签吗？欢迎在评论区留下你的思考，也欢迎你把今天的内容分享给你身边的朋友，邀他一起学习。我们下一讲再见！

赞 6　提建议

上一篇　17丨组件监控：Kubernetes Node组件的关键指标与数据采集

下一篇　19丨应用监控：如何使用埋点方式对应用监控？

精选留言 (5)

写留言

Geek_1a3949
2023-02-17 来自上海

尝试回答一下课后题：

可以为prometheus增加global.external_labels配置，增加cluster的标识以区分不同的集群：

```
global:
  external_labels:
    cluster:  prod-bigdata-sh
    ....
```

另外，请教老师一个问题，ksm的分片，官网上有statefulset、daemonset的分片方式，它们各自的适用场景是什么，在生产环境下，更推荐哪种方式？

作者回复: 量小的话sts即可，量大的话得daemonset

4

胡飞
2023-03-15 来自上海

你好老师，promtheus 如果开启了remote write后，存储会用两份吗？prom本地一份数据，第三方一份数据？

作者回复: 是的

1

晴空万里
2023-03-01 来自广东

自己做监控系统 但是用了公有云产品 比如华为云的k8s 请问怎么监控哈 公司都是用公有云saas 自己公司卖paas产品

作者回复: 只用了一个云的话，其实直接用云的监控就可以了，未来要是有多云或者混合云的场景，再考虑自建监控。如果自己搞监控来监控K8s，主要关注工作负载节点就可以了，托管的K8s控制面组件的可靠性让云厂商来做

LiLian
2023-02-24 来自广东

请问老师："prometheus agent mode，支持 Kubernetes 服务发现" 本质上还是通过list & watch 监听来自api server的信息吗？

作者回复: 是

peter
2023-02-18 来自北京

请教老师几个问题：
Q1：KSM是k8s自身的组件吗？还是一个第三方的软件？
Q2：一般性的问题，公司的实际运营中，日志数据一般保存多长时间？指标数据一般保存多长时间？（到期后是直接删除数据吧）

作者回复: 1，不是，但可以看做是伴生的重要性
2，有些日志有政策审计要求，有些只查问题，有些用于做数据统计，日志保存时间都不一样，指标的话默认15天即可，一些业务指标可能会需要永久保存