17 | 组件监控：Kubernetes Node组件的关键指标与数据采集

2023-02-15 秦晓辉 来自北京

讲述：秦晓辉

时长 15:33 大小 14.20M

你好，我是秦晓辉。

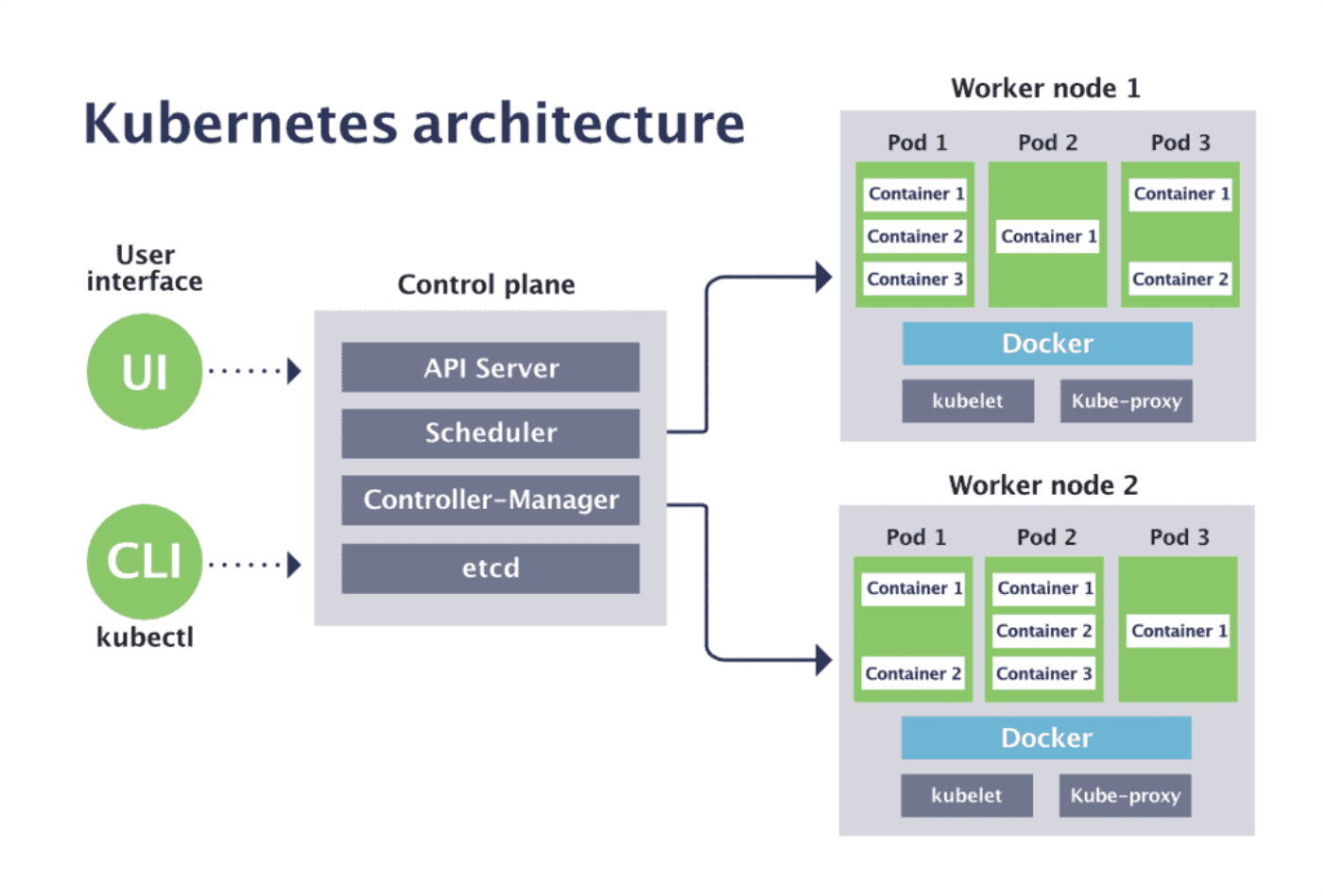
前面几讲我们介绍了 MySQL、Redis、Kafka、Elasticsearch 这些常见组件的监控方法，相信你对各类中间件的典型监控逻辑已经有了一定的认识。接下来我要介绍的是云原生时代的扛把子 Kubernetes 的监控，云原生这个词就是随着 Kubernetes 火起来的。Kubernetes 架构比较复杂，我会用两讲的时间来分享。

虽然网上可以找到基于 Prometheus 做的 Operator，一键监控 Kubernetes，但是很多人仍然不知其所以然，这两讲我会按照组件粒度掰开来讲，争取让你理解其中的原理，至于后面你用什么工具来落地，那都是技术的层面了，好办。

要监控 Kubernetes，我们得先弄明白 Kubernetes 有哪些模块要监控，所以我们先来看一下 Kubernetes 的架构。

Kubernetes 架构

下面是 Kubernetes 的架构图，用户交互部分是 UI 和 CLI，这两个不需要监控，关键是 **Control plane（控制面）和 Worker node（工作负载节点）**。控制面的组件提供了管理和调度能力，如果控制面组件出了问题，我们就没法给 Kubernetes 下发指令了。工作负载节点运行了容器，以及管理这些容器的运行时引擎（图上的 Docker）、管理 Pod 的 Kubelet，以及转发规则的 Kube-Proxy。工作负载节点如果出问题，可能会直接影响业务流量，所以对这类节点的监控就显得更为重要了。



图片来自网络

当然，除了控制面组件和工作负载节点的监控，整个 Kubernetes 监控体系还应该包含另外三部分，一个是 Kubernetes 所在宿主的监控，一个是 Kubernetes 上面运行对象的监控，还有一个是 Pod 内业务的监控。宿主的监控就是机器的监控，第 11 讲我们介绍过。Kubernetes 的对象监控，使用 kube-state-metrics（简称 KSM） 监控。Pod 内的业务的监控，已经超过了组件监控的范畴，后面我会详细介绍。

所以 Kubernetes 组件监控的这两讲，我会重点介绍控制面、工作负载、KSM 三个方面的监控。随着越来越多公司选择公有云的 Kubernetes 托管服务，控制面的组件直接交给云厂商来

托管了，我们只需要关注工作负载节点，所以这一讲我们先来介绍工作负载节点的监控。

工作负载节点我们重点关注两部分，一个是**容器负载**，一个是**组件**，组件又包括 Kube-Proxy、Kubelet、容器引擎。容器引擎一般不会出问题，所以我们重点关注 Kubernetes 的两个组件 Kube-Proxy 和 Kubelet。按照先易后难，循序渐进的顺序，我们先来看一下 Kube-Proxy 的监控方案。

监控 Kube-Proxy

所有的 Kubernetes 组件，都提供了 `/metrics` 接口用来暴露监控数据，Kube-Proxy 也不例外。通过 `ss` 或者 `netstat` 命令可以看到 Kube-Proxy 监听的端口，一个是 10249，用来暴露监控指标，一个是 10256，作为健康检查的端口，一般我们只关注前一个端口。下面我来测试一下。

```
[root@tt-fc-dev01.nj ~]# curl -s localhost:10249/metrics | head -n 6
# HELP apiserver_audit_event_total [ALPHA] Counter of audit events generated an
# TYPE apiserver_audit_event_total counter
apiserver_audit_event_total 0
# HELP apiserver_audit_requests_rejected_total [ALPHA] Counter of apiserver req
# TYPE apiserver_audit_requests_rejected_total counter
apiserver_audit_requests_rejected_total 0
```

不需要认证直接就可以拿到指标，很简单，我们只要有个采集器能够抓取这个数据就可以了。支持 Prometheus 协议数据抓取的采集器挺多的，这里我还是使用 Categraf 给你演示，相信通过前面几讲的演示，你对 Categraf 已经很熟悉了。

配置采集规则

抓取 Prometheus 协议的数据，使用 Categraf 的 input.prometheus 插件，配置文件在 `conf/input.prometheus/prometheus.toml`，要抓取哪个目标地址，就直接把 URL 配置到抓取地址中，可以参考下面的样例。

```
interval = 15
[[instances]]
urls = [
     "http://localhost:10249/metrics"
]
```

```
labels = { job="kube-proxy" }
```

之后我们就可以使用 `./categraf --test --inputs prometheus` 测试了，如果一切正常，在控制台就能看到采集到的 Kube-Proxy 相关的指标，具体哪些指标比较关键呢？先不急，后面我会谈到。

Kube-Proxy 在 Kubernetes 集群的所有节点上部署，如果使用上面的采集规则配置方式，就需要在所有 Kubernetes 节点的 Categraf 上配置采集规则，未来扩容节点的时候，也要记得在新节点配置采集规则。如果机器初始化流程做得不错，也还好，否则的话就会比较麻烦。我更推荐的方式，是把 Categraf 作为 Daemonset 部署，这样每次新节点扩容，Kubernetes 会自动调度，省事不少。下面我来演示一下如何部署 Categraf Daemonset。

使用 Daemonset 部署采集器

要把 Categraf 部署为 Daemonset，需要先创建一个 namespace，然后把相关的配置做成 ConfigMap，下面我做一个演示，先创建 namespace。

复制代码

```
# 创建 namespace
[work@tt-fc-dev01.nj categraf]$ kubectl create namespace flashcat
namespace/flashcat created

# 查询刚刚创建的namespace，看是否创建成功
[work@tt-fc-dev01.nj categraf]$ kubectl get ns | grep flashcat
flashcat                                Active   29s
```

然后创建 ConfigMap，ConfigMap 用来放置 Categraf 的主配置 config.toml，以及 input.prometheus 插件的配置 prometheus.toml，你可以看一下相关的 YAML 内容。

复制代码

```
---
kind: ConfigMap
metadata:
  name: categraf-config
apiVersion: v1
data:
  config.toml: |
    [global]
    hostname = "$HOSTNAME"
```

```
    interval = 15
    providers = ["local"]
    [writer_opt]
    batch = 2000
    chan_size = 10000
    [[writers]]
    url = "http://10.206.0.16:19000/prometheus/v1/write"
    timeout = 5000
    dial_timeout = 2500
    max_idle_conns_per_host = 100
---
kind: ConfigMap
metadata:
  name: categraf-input-prometheus
apiVersion: v1
data:
  prometheus.toml: |
    [[instances]]
    urls = ["http://127.0.0.1:10249/metrics"]
    labels = { job="kube-proxy" }
```

上例中的 `http://10.206.0.16:19000/prometheus/v1/write` 是一个支持 Prometheus Remote Write 协议的数据接收地址，可以使用你的 n9e-server，也可以使用 vminsert、prometheus 等其他支持 RemoteWrite 协议的地址。`hostname = "$HOSTNAME"` 这个配置用了 `$` 符号，后面创建 Daemonset 的时候会注入 HOSTNAME 这个环境变量，让 Categraf 自动拿到。

prometheus.toml 的配置中，除了给出 Kube-Proxy 的抓取地址，还手工指定了一个 job 标签，用来标识这个数据来自哪个组件。如果公司有多套 Kubernetes 集群，所有监控数据都会进到一个时序库，为了区分不同的集群，我建议你在标签里再加一个 cluster 的标签。

比如：

复制代码

```
labels = { job="kube-proxy", cluster="beijing01" }
```

下面我们把 ConfigMap 创建出来。

复制代码

```
[work@tt-fc-dev01.nj yamls]$ kubectl apply -f categraf-configmap-v1.yaml -n fla
configmap/categraf-config created
```

```
configmap/categraf-input-prometheus created

[work@tt-fc-dev01.nj yamls]$ kubectl get configmap -n flashcat
NAME                         DATA   AGE
categraf-config              1      19s
categraf-input-prometheus    1      19s
kube-root-ca.crt             1      22m
```

配置文件准备好了，接下来就可以创建 Daemonset 了。这里要注意，把 HOSTNAME 作为环境变量注入进去，你可以参考我给出的这个 Daemonset 的 YAML 文件。

复制代码

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  labels:
    app: categraf-daemonset
  name: categraf-daemonset
spec:
  selector:
    matchLabels:
      app: categraf-daemonset
  template:
    metadata:
      labels:
        app: categraf-daemonset
    spec:
      containers:
      - env:
        - name: TZ
          value: Asia/Shanghai
        - name: HOSTNAME
          valueFrom:
            fieldRef:
              apiVersion: v1
              fieldPath: spec.nodeName
        - name: HOSTIP
          valueFrom:
            fieldRef:
              apiVersion: v1
              fieldPath: status.hostIP
        image: flashcatcloud/categraf:v0.2.18
        imagePullPolicy: IfNotPresent
        name: categraf
        volumeMounts:
        - mountPath: /etc/categraf/conf
          name: categraf-config
        - mountPath: /etc/categraf/conf/input.prometheus
          name: categraf-input-prometheus
```

```
      hostNetwork: true
      restartPolicy: Always
      tolerations:
      - effect: NoSchedule
        operator: Exists
      volumes:
      - configMap:
          name: categraf-config
        name: categraf-config
      - configMap:
          name: categraf-input-prometheus
        name: categraf-input-prometheus
```

最后一步，apply 一下这个 Daemonset 的 YAML 文件。

复制代码

```
[work@tt-fc-dev01.nj yamls]$ kubectl apply -f categraf-daemonset-v1.yaml -n fla
daemonset.apps/categraf-daemonset created

[work@tt-fc-dev01.nj yamls]$ kubectl get ds -o wide -n flashcat
NAME                 DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
categraf-daemonset   6         6         6       6            6           <none

[work@tt-fc-dev01.nj yamls]$ kubectl get pods -o wide -n flashcat
NAME                       READY   STATUS    RESTARTS   AGE     IP             N
categraf-daemonset-4qlt9   1/1     Running   0          2m10s   10.206.0.7     1
categraf-daemonset-s9bk2   1/1     Running   0          2m10s   10.206.0.11    1
categraf-daemonset-w77lt   1/1     Running   0          2m10s   10.206.16.3    1
categraf-daemonset-xgwf5   1/1     Running   0          2m10s   10.206.0.16    1
categraf-daemonset-z9rk5   1/1     Running   0          2m10s   10.206.16.8    1
categraf-daemonset-zdp8v   1/1     Running   0          2m10s   10.206.0.17    1
```

看起来一切正常，去监控服务端查询一下 kubeproxy 打头的指标，理论上就能看到采集到的数据了。Kube-Proxy 暴露了不少指标，下面我挑选一些关键指标稍作解释。

Kube-Proxy 指标解释

1. **通用的 Go 程序相关的指标**

通用的 Go 程序相关的指标	解释
go_gc_duration_seconds	GC 时间
go_goroutines	goroutine 数量
go_threads	线程数量
process_max_fds	进程可以打开的最大文件句柄数量
process_open_fds	进程当前打开了多少文件句柄
process_resident_memory_bytes	内存使用大小
process_cpu_seconds_total	进程使用的 CPU 时间
process_start_time_seconds	进程启动时间戳

以上指标，只要是通过 Prometheus Go SDK 埋点的程序都会有，除了 Kube-Proxy，后面介绍的 Kubelet、APIServer、Scheduler 等，也全部都有，这里你记住了，后面我就不会重复介绍了。

2. **请求 APIServer 的指标**

Kubernetes 中多个组件都要调用 APIServer 的接口，每秒调用多少次、有多少成功多少失败、耗时情况如何，这些指标也比较关键。

比如：

- rest_client_request_duration_seconds：请求 APIServer 的耗时统计
- rest_client_requests_total：请求 APIServer 的调用量统计

3. **规则同步类指标**

Kube-Proxy 的核心职能，就是去 APIServer 获取转发规则，修改本地的 iptables 或者 ipvs 的规则，所以这些规则同步相关的指标，就至关重要了。这里我给你列出几个核心指标。

规则同步类指标	解释
kubeproxy_sync_proxy_rules_duration_seconds	规则同步耗时
kubeproxy_sync_proxy_rules_endpoint_changes_total	endpoint 变化的总次数
kubeproxy_sync_proxy_rules_iptables_restore_failures_total	iptables restore 失败总次数
kubeproxy_sync_proxy_rules_last_queued_timestamp_seconds	最近一次开始同步的时间戳
kubeproxy_sync_proxy_rules_last_timestamp_seconds	最近一次完成同步的时间戳

Categraf 内置了 Kube-Proxy 的监控大盘，关键的核心指标都已经做到监控大盘里了，导入夜莺就能使用。

因为 Kube-Proxy 是我们讲解的第一个组件，讲得啰嗦了一些，后面介绍其他组件的时候有些相同的逻辑就不重复了。下面我们继续看工作负载节点的第二个组件 Kubelet 应该如何监控。

监控 Kubelet

Kubelet 也是在所有 Kubernetes 节点上部署的，理论上可以采用和 Kube-Proxy 完全一样的监控方法，但是 Kubelet 的接口需要认证，我们来测试一下。


```
[root@tt-fc-dev01.nj ~]# ps aux|grep kubelet
root      163490  0.0  0.0  12136  1064 pts/1    S+   13:34   0:00 grep --color
root      166673  3.2  1.0 3517060 81336 ?       Ssl  Aug16 4176:52 /usr/bin/ku

[root@tt-fc-dev01.nj ~]# cat /var/lib/kubelet/config.yaml | grep 102
healthzPort: 10248

[root@tt-fc-dev01.nj ~]# curl localhost:10248/healthz
ok

[root@tt-fc-dev01.nj ~]# curl localhost:10250/metrics
Client sent an HTTP request to an HTTPS server.

[root@tt-fc-dev01.nj ~]# curl https://localhost:10250/metrics
curl: (60) SSL certificate problem: self signed certificate in certificate chai
More details here: https://curl.haxx.se/docs/sslcerts.html

curl failed to verify the legitimacy of the server and therefore could not
establish a secure connection to it. To learn more about this situation and
```

```
how to fix it, please visit the web page mentioned above.
[root@tt-fc-dev01.nj ~]# curl -k https://localhost:10250/metrics
Unauthorized
```

这几条测试命令可以说明很多问题，首先是 Kubelet 监听了两个端口，一个是 10248，是个健康检查端口，另一个是 10250，暴露 metrics 指标，但是访问这个接口需要传入 Authorization 的 Token，下面我们就来创建 ServiceAccount。Kubernetes 会为 ServiceAccount 自动分配 Token。

引入认证信息

只创建 ServiceAccount 没什么用，还需要为这个账号绑定权限，Kubernetes 中使用 ClusterRole 来定义权限，使用 ClusterRoleBinding 来绑定 ClusterRole 和 ServiceAccount，下面是相关 YAML 定义。

复制代码

```yaml
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: categraf-daemonset
rules:
- apiGroups:
  - ""
  resources:
  - nodes/metrics
  - nodes/stats
  - nodes/proxy
  verbs:
  - get
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: categraf-daemonset
  namespace: flashcat
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: categraf-daemonset
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: categraf-daemonset
subjects:
```

```
- kind: ServiceAccount
  name: categraf-daemonset
  namespace: flashcat
```

把上面的内容保存为 auth.yaml，apply 一下，然后我们从 ServiceAccount 中提取 Token，做一下 metrics 接口的请求测试。

复制代码

```
[work@tt-fc-dev01.nj yamls]$ kubectl apply -f auth.yaml
clusterrole.rbac.authorization.k8s.io/categraf-daemonset created
serviceaccount/categraf-daemonset created
clusterrolebinding.rbac.authorization.k8s.io/categraf-daemonset created

[work@tt-fc-dev01.nj yamls]$ kubectl get sa -n flashcat
NAME                 SECRETS   AGE
categraf-daemonset   1         90m
default              1         4d23h

[root@tt-fc-dev01.nj qinxiaohui]# kubectl get sa categraf-daemonset -n flashcat
apiVersion: v1
kind: ServiceAccount
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","kind":"ServiceAccount","metadata":{"annotations":{},"
  creationTimestamp: "2022-11-14T03:53:54Z"
  name: categraf-daemonset
  namespace: flashcat
  resourceVersion: "120570510"
  uid: 22f5a785-871c-4454-b82e-12bf104450a0
secrets:
- name: categraf-daemonset-token-7mccq

[root@tt-fc-dev01.nj qinxiaohui]# token=`kubectl get secret categraf-daemonset-
[root@tt-fc-dev01.nj qinxiaohui]# curl -s -k -H "Authorization: Bearer $token"
[root@tt-fc-dev01.nj qinxiaohui]# head -n 5 aaaa
# HELP apiserver_audit_event_total [ALPHA] Counter of audit events generated an
# TYPE apiserver_audit_event_total counter
apiserver_audit_event_total 0
# HELP apiserver_audit_requests_rejected_total [ALPHA] Counter of apiserver req
# TYPE apiserver_audit_requests_rejected_total counter
apiserver_audit_requests_rejected_total 0
```

这几个命令看起来比较清晰了，创建的 ServiceAccount 名为 categraf-daemonset，导出为 YAML 之后，看到 secret 的 name 是 categraf-daemonset-token-7mccq，然后从这个 secret

中解出 Token，放到 Header 里，请求 Kubelet 的 metrics 接口，最终拿到了数据，搞定收工。

后面我们把 Categraf 作为采集器做成 Daemonset，再为 Categraf 这个 Daemonset 指定 ServiceAccountName，Kubernetes 就会自动把 Token 的内容挂到 Daemonset 的目录里，下面我们开始实操。

升级 Categraf Daemonset

采集 Kube-Proxy 的时候，我们已经准备好了 Categraf Daemonset 用到的 ConfigMap，当时只是抓取了 Kube-Proxy 的 metrics 数据，下面我们升级一下这个 ConfigMap 的内容，加上对 Kubelet 的数据抓取规则。

复制代码

```
---
kind: ConfigMap
metadata:
  name: categraf-config
apiVersion: v1
data:
  config.toml: |
    [global]
    hostname = "$HOSTNAME"
    interval = 15
    providers = ["local"]
    [writer_opt]
    batch = 2000
    chan_size = 10000
    [[writers]]
    url = "http://10.206.0.16:19000/prometheus/v1/write"
    timeout = 5000
    dial_timeout = 2500
    max_idle_conns_per_host = 100
---
kind: ConfigMap
metadata:
  name: categraf-input-prometheus
apiVersion: v1
data:
  prometheus.toml: |
    [[instances]]
    urls = ["http://127.0.0.1:10249/metrics"]
    labels = { job="kube-proxy" }
    [[instances]]
    urls = ["https://127.0.0.1:10250/metrics"]
    bearer_token_file = "/var/run/secrets/kubernetes.io/serviceaccount/token"
```

```
    use_tls = true
    insecure_skip_verify = true
    labels = { job="kubelet" }
    [[instances]]
    urls = ["https://127.0.0.1:10250/metrics/cadvisor"]
    bearer_token_file = "/var/run/secrets/kubernetes.io/serviceaccount/token"
    use_tls = true
    insecure_skip_verify = true
    labels = { job="cadvisor" }
```

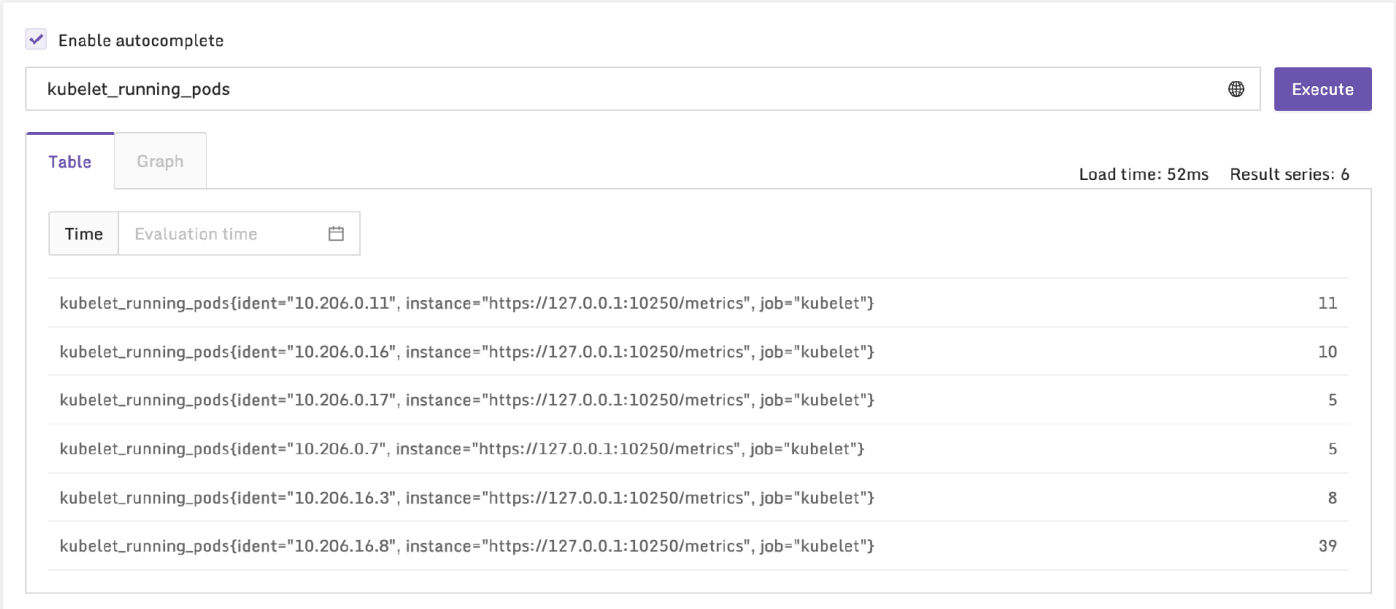
Kubelet 在 10250 端口暴露了两类 metrics 数据，一个是 `/metrics`，暴露的是 Kubelet 自身的监控数据，另一个是 `/metrics/cadvisor`，暴露的是容器的监控数据。

然后修改一下之前的 Daemonset 的 yaml 文件，在 hostNetwork 这一行下面增加 ServiceAccountName 配置，你可以看下示例。

复制代码

```
hostNetwork: true
serviceAccountName: categraf-daemonset
restartPolicy: Always
```

我们可以把之前的 Daemonset 直接删除，使用新的 yaml 重新创建，稍等片刻，就能在服务端查询到 Kubelet 和容器的监控数据了。



Kubelet 关键指标

Kubelet 也会吐出 Go 进程相关的通用指标以及和 APIServer 通信相关的度量指标，和 Kube-Proxy 类似。Kubelet 核心职能是管理 Pod，操作各种 CNI、CSI 相关的接口，和容器引擎打交道，度量这类操作的指标就显得尤为关键。

比如：

Kubelet 关键指标	解释
kubelet_docker_operations_duration_seconds	操作 Docker 引擎的耗时
kubelet_docker_operations_errors_total	操作 Docker 引擎的失败次数
kubelet_docker_operations_timeout_total	操作 Docker 引擎的超时次数
kubelet_docker_operations_total	操作 Docker 引擎的累计次数
kubelet_network_plugin_operations_duration_seconds	网络插件操作耗时
kubelet_network_plugin_operations_errors_total	网络插件操作失败的次数
kubelet_network_plugin_operations_total	网络插件操作累计次数

Categraf 内置了 Kubelet 的监控大盘，关键的核心指标都已经做到监控大盘里了，导入夜莺就能使用。

刚才我们在升级 Categraf Daemonset 的 ConfigMap 的时候，不只采集了 Kubelet 的指标，还一并采集了容器的指标，Categraf 也提供了容器的监控大盘。关于容器，我这里也选几个核心的指标给你解释一下。

容器负载指标

容器负载主要是关心 CPU、内存、网络、IO，尤其是 CPU 和内存，我们一起看一下相关指标的说明。

CPU 指标


```
sum(
```

```
irate(container_cpu_usage_seconds_total[3m])
) by (pod,id,namespace,container,ident,image)
/
sum(
container_spec_cpu_quota/container_spec_cpu_period
) by (pod,id,namespace,container,ident,image)
```

这是计算 CPU 使用率，整体是一个除法运算，分子部分是容器每秒耗费的 CPU 时间，分母部分是每秒分配给容器的 CPU 时间。里边的 ident 标签是 Categraf 采集时自动加的，如果你的采集方式和我演示的不同，可能要适当调整 by 后面的标签集。

复制代码

```
increase(container_cpu_cfs_throttled_periods_total[1m])
/
increase(container_cpu_cfs_periods_total[1m]) * 100
```

这是在计算 CPU 被限制的时间比例，如果这个值很高，说明容器在使用 CPU 资源的时候经常被限制，需要提高这个容器的 CPU Quota。延迟敏感型的应用，需要特别关注这个指标。

内存指标

复制代码

```
container_memory_working_set_bytes
/
container_spec_memory_limit_bytes
and
container_spec_memory_limit_bytes != 0
```

计算内存使用率的时候，核心也是一个除法运算，分子是容器的内存占用，分母是内存 Limit 大小。当然，有些容器没有指定内存 Limit，所以还需要有个 and 语句来做限制，只有 limit_bytes 不等于 0，这个除法运算才有意义。

Pod 网络流量

复制代码

```
irate(container_network_transmit_bytes_total[1m]) * 8
irate(container_network_receive_bytes_total[1m]) * 8
```

这个指标名字非常清晰，transmit 是出向，receive 是入向，这两个指标都是 Counter 类型的值，单调递增，所以使用 irate 计算每秒速率。因为网络流量一般都是用 bit 作为单位，所以最后乘以 8，把 byte 换算成 bit。

Pod 硬盘 IO 读写流量

```
irate(container_fs_reads_bytes_total[1m])
irate(container_fs_writes_bytes_total[1m])
```

这个指标名字一看就知道是 Counter 类型，我们不关心当前值是多少，而是关心最近一段时间每秒的速率是多少，所以使用 irate 做了二次计算。

刚刚我们说的这些就是容器负载相关的关键指标。到这里，我们就介绍完工作负载节点的核心知识点了，下面我们对整体内容做一个小结。

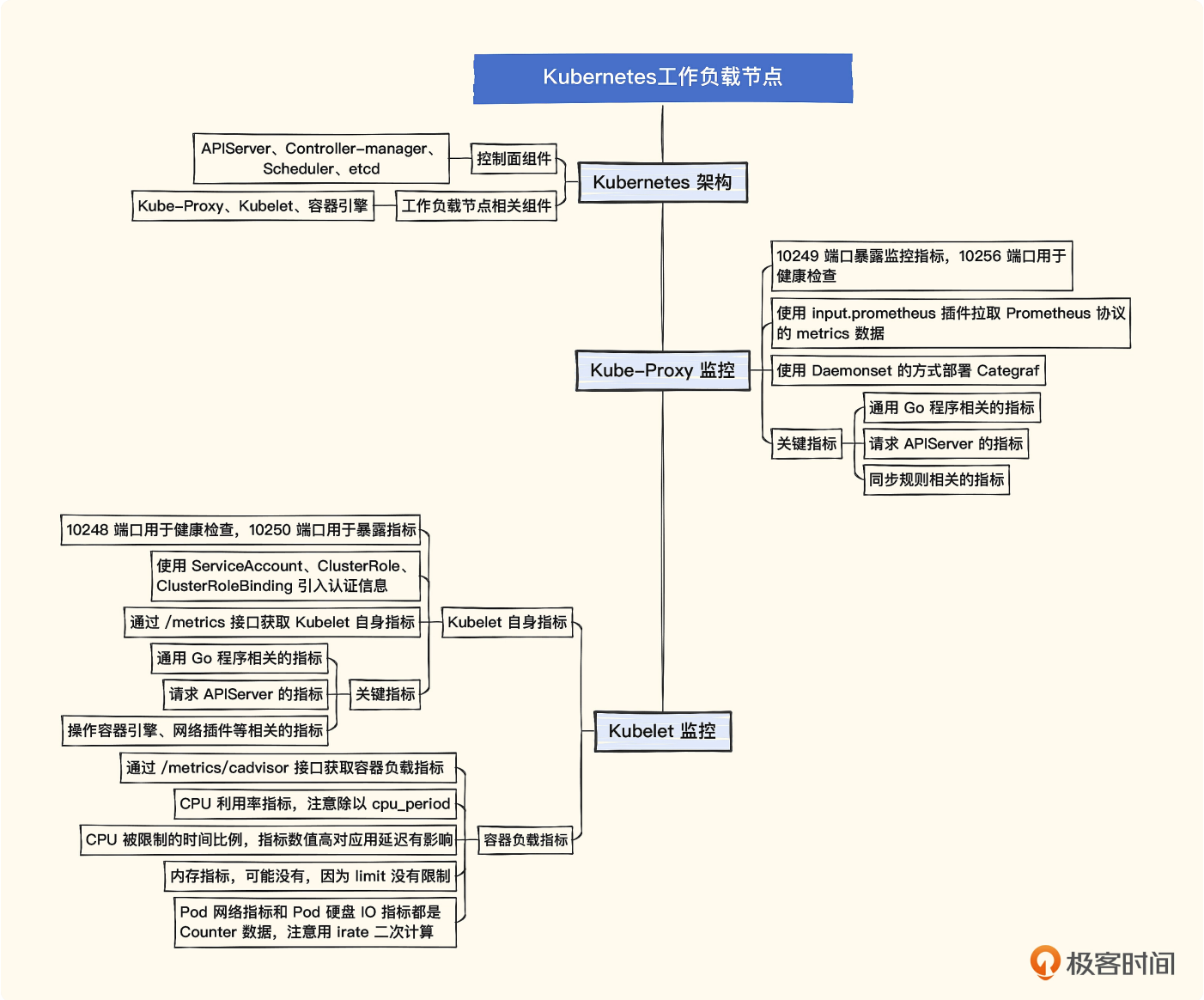
小结

这一讲我们介绍了 Kubernetes 的监控，Kubernetes 组件众多，通过架构图可以看出，大体上可以分为两部分，一个是控制面组件，一个是工作负载节点相关组件。我们重点介绍了工作负载节点相关的组件，包括 Kube-Proxy、Kubelet、容器负载。当然，容器引擎是否存活也是需要关注的，不过容器引擎一般都通过 systemd 托管，挂掉之后会自动拉起，出问题的概率很小。

Kube-Proxy 的监控比较简单，通过 metrics 接口直接暴露监控指标，没有认证鉴权，使用 Categraf 直接拉取就可以了。为了便于管理，建议你把 Categraf 做成 Daemonset。Kube-Proxy 的关键指标分三类：一是通用的 Go 程序相关的指标，所有的 Kubernetes 组件都有这类指标；二是请求 APIServer 相关的指标，所有请求 APIServer 的模块都有这类指标；三是规则同步类指标，因为 Kube-Proxy 核心职能就是要做好规则同步，所以这类指标是最关键的。

Kubelet 的监控，相对更复杂，因为有认证鉴权的要求，需要创建 ServiceAccount、ClusterRole、ClusterRoleBinding 等对象。因为 Kubelet 也是在所有宿主上的，所以采集器也可以部署为 Daemonset。Kubelet 的关键指标是跟操作相关的，比如操作 Docker 引擎，操作网络插件等，这些操作的次数、成功与否，都非常关键。

Kubelet 的接口还暴露了容器负载指标，通过 /metrics/cadvisor 来抓取，重点关注 CPU、内存、网络、硬盘 IO 等指标。CPU 方面尤其要注意容器被限制的时间比例，对于延迟敏感型业务有较大影响。



互动时刻

通过 Kubelet 的 `/metrics/cadvisor` 接口虽然可以采集到容器指标，但是拿不到应用标签，比如某个 Deployment 名字叫 n9e-webapi，打了两个标签：`region=beijing`, `dept=cloud`，通过这个 Deployment 创建出来的 Pod 的监控数据，比如 `container_fs_writes_bytes_total`，我也希望带有这个标签，或者虽然这个指标没有直接带有这个标签，也希望能有种方式让我通过 region、dept 这类标签来过滤容器监控数据，应该如何实现呢？

欢迎留言分享你的实践方式，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！

上一篇 16丨组件监控：Elasticsearch的关键指标及采集方法有哪些？

下一篇 18丨组件监控：Kubernetes 控制面组件的关键指标与数据采集

精选留言 (6)

写留言

Geek_1a3949
2023-02-15 来自上海

尝试回答一下课后题：

使用group_left，为container_*添加kube_pod_labels标签
container_fs_writes_bytes_total * on(namespace, pod) group_left(label_region, label_dept) kube_pod_labels

作者回复:

3

StackOverflow
2023-03-01 来自上海

在 k8s 1.24之后的版本，创建的 serviceaccount 没有自动生成 secret，需要手动创建一个

共 1 条评论>

1

隆哥
2023-03-27 来自北京

老师请教一下，我如何监控所有的pod，只要pod异常我就报警呢？

作者回复: 用ksm采集数据，然后试试这个promql:

(sum(kube_pod_container_status_last_terminated_reason{reason=~\"Error|OOMKilled\"}) by (namespace,pod,container) >0)
* on(namespace,pod,container)
sum(increase(kube_pod_container_status_restarts_total) >0) by(namespace,pod,container)


隆哥
2023-03-06 来自上海

老师请教一下，ServiceAccount在k8s高版本我是1.26.0版本，无法自动创建secret，如果是自己手动创建secret的话，是帐号密码形式的嘛，如果自己创建secret

作者回复: apiVersion: v1
kind: Secret
metadata:
  name: xxxxx
  namespace: "service account 所在的 namespace"
  annotations:
    kubernetes.io/service-account.name: "这里写 service account 的 name"
type: kubernetes.io/service-account-token


晴空万里
2023-02-28 来自广东

公司准备搭建自己的运维监控系统 怎么把老师讲的这些结合起来？老师介绍了监控数据采集 存储 但是运维系统搭建 原型上 我感觉有点懵逼

作者回复: 专栏更多的是介绍思路。先吸收。然后用开源的先搭建使用起来。再之后根据自己公司的情况演进


peter
2023-02-16 来自北京

请教老师几个问题：
Q1：Go程序指标只对Go程序有效吗？
Kube-proxy的Go程序指标，是针对Go应用吗？如果不是Go应用呢？比如Java应用就不会有这个指标吧。
Q2：[1m]是什么意思？
多个指标后面有"[1m]"，比如：irate(container_network_transmit_bytes_total[1m])，是表示1

分钟吗？

Q3：监控大盘的配置是categraf内置的吗？

文中提到“Categraf 内置了 Kube-Proxy 的监控大盘；

Categraf 内置了 Kubelet 的监控大盘；Categraf 也提供了容器的监控大盘。”点开链接后，是一个json格式的配置文件。请问：这些配置文件是categraf本身就有的吗？

Q4：配置文件所在的github链接是专栏的吗？

问题三中提到的几个监控大盘，都对应一个github链接，https://github.com/flashcatcloud这个链接是本专栏自己的吗？还是一个开源项目？

作者回复: 1，是

2，表示时间范围，前面的章节有介绍range query

3，我的意思是categraf代码仓库里提供了相关的大盘，所以称为内置，这些大盘导入夜莺就可以用

4，那是一个开源项目，借此机会介绍如何监控，并非是专门为专栏而生