

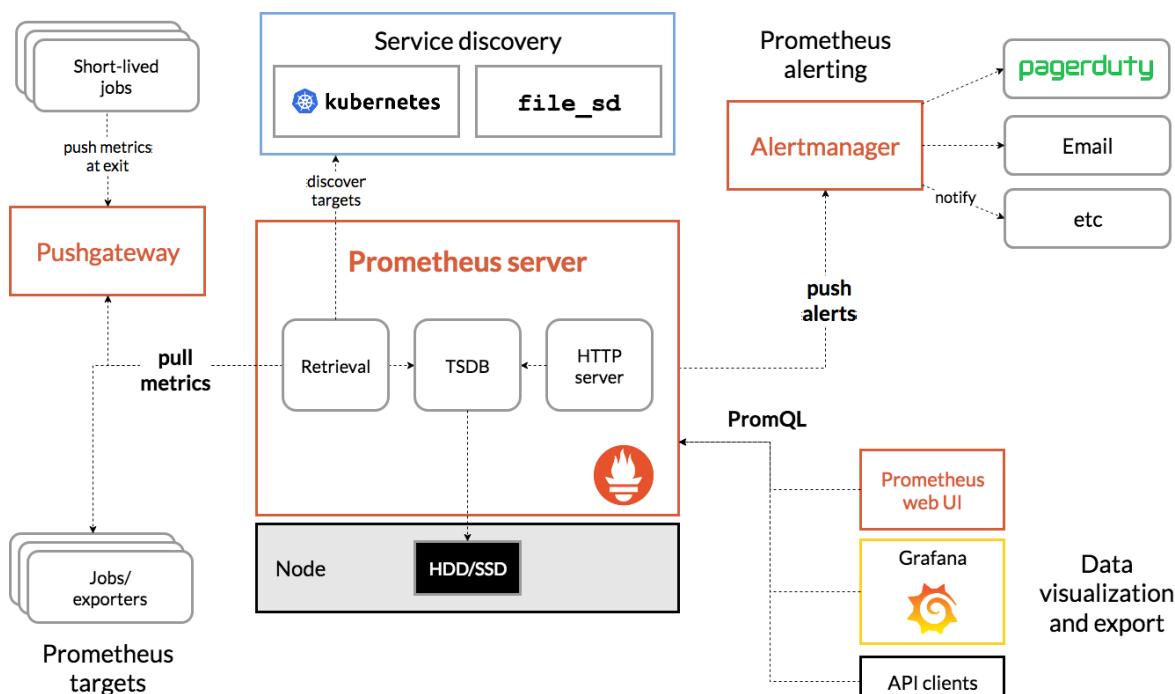
第x章 Prometheus介绍

官方地址

1 | <https://prometheus.io/docs/introduction/overview/>

组件架构

- | | | |
|---|-------------------|----------------------|
| 1 | Prometheus Server | 服务端，主动拉数据，存入TSDB数据库 |
| 2 | TSDB | 时序数据库，用于存储拉取来的监控数据 |
| 3 | exporter | 暴露指标的组件，需要独立安装 |
| 4 | Pushgatway | push的方式将指标数据推送到网关 |
| 5 | Alertmanager | 报警组件 |
| 6 | Prometheus Web UI | prometheus自带的简单web界面 |



第x章 安装部署

官方配置文件说明

1 | <https://prometheus.io/docs/prometheus/latest/configuration/configuration/>

编写configmap

```
1 cat > prom-cm.yaml << 'EOF'  
2 apiVersion: v1  
3 kind: ConfigMap  
4 metadata:  
5   name: prometheus-config  
6   namespace: prom
```

```

7 data:
8   prometheus.yml: |
9     global:          #全局配置
10    scrape_interval: 15s #抓取数据间隔
11    scrape_timeout: 15s #抓取超时时间
12    scrape_configs:   #拉取配置
13      - job_name: 'prometheus' #任务名称
14        static_configs:       #静态配置
15          - targets: ['localhost:9090'] #抓取数据节点的IP端口
16 EOF

```

创建PV和PVC

```

1 cat > prom-pv-pvc.yaml << 'EOF'
2 apiVersion: v1
3 kind: PersistentVolume
4 metadata:
5   name: prometheus-local
6   labels:
7     app: prometheus
8 spec:
9   accessModes:
10  - ReadWriteOnce
11  capacity:
12    storage: 10Gi
13  storageClassName: local-storage
14  local:
15    path: /data/k8s/prometheus
16 nodeAffinity:
17   required:
18     nodeSelectorTerms:
19       - matchExpressions:
20         - key: kubernetes.io/hostname
21           operator: In
22           values:
23             - node2
24   persistentVolumeReclaimPolicy: Retain
25 ---
26 apiVersion: v1
27 kind: PersistentVolumeClaim
28 metadata:
29   name: prometheus-data
30   namespace: prom
31 spec:
32   selector:
33     matchLabels:
34       app: prometheus
35   accessModes:
36     - ReadWriteOnce
37   resources:
38     requests:
39       storage: 10Gi
40   storageClassName: local-storage
41 EOF

```

编写RBAC

```
1 cat > prom-rbac.yaml << 'EOF'
2 apiVersion: v1
3 kind: ServiceAccount
4 metadata:
5   name: prometheus
6   namespace: prom
7 ---
8 apiVersion: rbac.authorization.k8s.io/v1
9 kind: ClusterRole
10 metadata:
11   name: prometheus
12 rules:
13 - apiGroups:
14   - ""
15   resources:
16     - nodes
17     - services
18     - endpoints
19     - pods
20     - nodes/proxy
21   verbs:
22     - get
23     - list
24     - watch
25 - apiGroups:
26   - "extensions"
27   resources:
28     - ingresses
29   verbs:
30     - get
31     - list
32     - watch
33 - apiGroups:
34   - ""
35   resources:
36     - configmaps
37     - nodes/metrics
38   verbs:
39     - get
40 - nonResourceURLs:
41   - /metrics
42   verbs:
43     - get
44 ---
45 apiVersion: rbac.authorization.k8s.io/v1
46 kind: ClusterRoleBinding
47 metadata:
48   name: prometheus
49 roleRef:
50   apiGroup: rbac.authorization.k8s.io
51   kind: ClusterRole
52   name: prometheus
53 subjects:
54 - kind: ServiceAccount
55   name: prometheus
56   namespace: prom
57 EOF
```

编写deployment

```
1 cat > prom-dp.yaml << 'EOF'
2 apiVersion: apps/v1
3 kind: Deployment
4 metadata:
5   name: prometheus
6   namespace: prom
7   labels:
8     app: prometheus
9 spec:
10  selector:
11    matchLabels:
12      app: prometheus
13  template:
14    metadata:
15      labels:
16        app: prometheus
17    spec:
18      serviceAccountName: prometheus          #引用RBAC创建的ServiceAccount
19      volumes:
20      - name: data
21        persistentVolumeClaim:
22          claimName: prometheus-data
23      - name: config-volume
24        configMap:
25          name: prometheus-config
26      initContainers:                         #由初始化容器将数据目录的属性修改为
nobody用户和组
27      - name: fix-permissions
28        image: busybox
29        command: [chown, -R, "nobody:nobody", /prometheus]
30        volumeMounts:
31        - name: data
32          mountPath: /prometheus
33  containers:
34  - name: prometheus
35    image: prom/prometheus:v2.24.1
36    args:
37    - "--config.file=/etc/prometheus/prometheus.yaml"    #指定配置文件
38    - "--storage.tsdb.path=/prometheus"        #tsdb数据库保存路径
39    - "--storage.tsdb.retention.time=24h"       #数据保留时间, 默认15天
40    - "--web.enable-admin-api" #控制对admin HTTP API的访问
41    - "--web.enable-lifecycle" #支持热更新, 直接执行localhost:9090/-/reload
立即生效
42  ports:
43  - name: http
44    containerPort: 9090
45    volumeMounts:
46    - name: config-volume
47      mountPath: "/etc/prometheus"
48    - name: data
49      mountPath: "/prometheus"
50    resources:
51      requests:
52        cpu: 100m
53        memory: 512Mi
```

```
54     limits:  
55         cpu: 100m  
56         memory: 512Mi  
57 EOF
```

编写Service

```
1 cat > prom-svc.yml << 'EOF'  
2 apiVersion: v1  
3 kind: Service  
4 metadata:  
5     name: prometheus  
6     namespace: prom  
7     labels:  
8         app: prometheus  
9 spec:  
10    selector:  
11        app: prometheus  
12    ports:  
13        - name: web  
14          port: 9090  
15          targetPort: http  
16 EOF
```

编写ingress

```
1 cat > prom-ingress.yml << 'EOF'  
2 apiVersion: networking.k8s.io/v1  
3 kind: Ingress  
4 metadata:  
5     name: prometheus  
6     namespace: prom  
7     labels:  
8         app: prometheus  
9 spec:  
10    rules:  
11        - host: prom.k8s.com  
12          http:  
13              paths:  
14                  - path: /  
15                      pathType: ImplementationSpecific  
16                      backend:  
17                          service:  
18                              name: prometheus  
19                              port:  
20                                  number: 9090  
21 EOF
```

访问promtheus

```
1 | http://prom.k8s.com
```

The screenshot shows the Prometheus Targets page. At the top, there are navigation links for Prometheus, Alerts, Graph, Status, Help, and Classic UI. Below that, a title 'Targets' is displayed. Under the 'Targets' section, there are two tabs: 'All' (selected) and 'Unhealthy'. A single target is listed: 'prometheus (1/1 up)'. The target details are as follows:

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9090/metrics	UP	instance="localhost:9090" job="prometheus"	1.43s	3.242ms	

第x章 应用检测

应用监控说明

prometheus的数据指标都是通过http实现的metrics接口获取到的，所以应用只需要暴露metrics接口，prometheus就可以定期的去拉取数据。

随着容器和k8s的流行，现在很多服务都自己内置了metrics接口，对于本身没有提供metrics的应用，promtheus官方也提供了很多可以直接使用的exporter来获取指标数据，比如redis_exporter,mysql_exporter等。

自带/metrics接口的应用检测

k8s里的coredns自带的metrics接口，所以我们可以先拿来试试手，查看coredns的配置文件可以发现提供prometheus服务采集的端口是9153。

```
1 [root@node1 prom]# kubectl -n kube-system describe cm coredns
2 Name:          coredns
3 Namespace:    kube-system
4 Labels:        <none>
5 Annotations:   <none>
6
7 Data
8 ====
9 Corefile:
10 -----
11 .:53 {
12     errors
13     health {
14         lameduck 5s
15     }
16     ready
17     kubernetes cluster.local in-addr.arpa ip6.arpa {
18         pods insecure
19         fallthrough in-addr.arpa ip6.arpa
20         ttl 30
21     }
22     prometheus :9153      #自带的prometheus监控暴露服务
23     forward . /etc/resolv.conf {
24         max_concurrent 1000
25     }
26     cache 30
27     loop
```

```
28     reload
29     loadbalance
30 }
31
32 Events: <none>
```

查看CoreDNS的Pod地址

```
1 [root@node1 prom]# kubectl -n kube-system get pod -l k8s-app=kube-dns -o wide
2 NAME                  READY   STATUS    RESTARTS   AGE     IP           NODE
3   NOMINATED-NODE   READINESS GATES
4 coredns-6d56c8448f-ckwhg   1/1     Running   5          10d    10.2.0.17
5   node1   <none>           <none>
6 coredns-6d56c8448f-rvmdf   1/1     Running   5          10d    10.2.0.16
7   node1   <none>           <none>
```

直接访问/metrics接口

```
1 [root@node1 prom]# curl -I 10.2.0.17:9153/metrics
2 HTTP/1.1 200 OK
3 Content-Type: text/plain; version=0.0.4; charset=utf-8
4 Date: Wed, 11 Aug 2021 07:14:25 GMT
5
6 [root@node1 prom]# curl 10.2.0.17:9153/metrics
7 # HELP coredns_build_info A metric with a constant '1' value labeled by
8 # version, revision, and goversion from which CoreDNS was built.
9 # TYPE coredns_build_info gauge
10 coredns_build_info{goversion="go1.14.4",revision="f59c03d",version="1.7.0"} 1
11 # HELP coredns_cache_entries The number of elements in the cache.
12 # TYPE coredns_cache_entries gauge
13 coredns_cache_entries{server="dns://:53",type="denial"} 12
14 coredns_cache_entries{server="dns://:53",type="success"} 1
15 # HELP coredns_cache_misses_total The count of cache misses.
16 # TYPE coredns_cache_misses_total counter
17 coredns_cache_misses_total{server="dns://:53"} 13
18 # HELP coredns_dns_request_duration_seconds Histogram of the time (in
19 # seconds) each request took.
# TYPE coredns_dns_request_duration_seconds histogram
.......
```

知道了端口，也确认了可以访问，那么接下来我们就可以编辑prometheus的配置文件来发现这个服务了。

编辑prom-cm配置文件

```
1 [root@node1 prom]# cat prom-cm.yaml
2 apiVersion: v1
3 kind: ConfigMap
4 metadata:
5   name: prometheus-config
6   namespace: prom
7 data:
8   prometheus.yaml: |
9     global:
10       scrape_interval: 15s
11       scrape_timeout: 15s
```

```

12     scrape_configs:
13       - job_name: 'prometheus'
14         static_configs:
15           - targets: ['localhost:9090']
16
17       - job_name: 'coredns'          #任务名称
18         static_configs:            #静态配置
19           - targets: ['10.2.0.16:9153', '10.2.0.17:9153']      #这里我们直接写
coredns的ClusterIP

```

更新prom-cm资源配置

```
1 | [root@node1 prom]# kubectl apply -f prom-cm.yaml
```

因为我们在prometheus的配置文件里配置了热更新的参数，所以可以不用重启pod在线热更新配置使其生效。

热更新promtheus配置

```

1 | [root@node1 prom]# kubectl -n prom get pod -o wide
2 |   NAME        READY   STATUS    RESTARTS   AGE     IP
3 |   NODE   NOMINATED-NODE   READINESS   GATES
4 |   prometheus-796566c67c-lhrns   1/1     Running   0        22m   10.2.2.86
5 |   node2   <none>           <none>
6 |   #注意要等一会，因为configmap更新到pod里需要点时间
7 |   [root@node1 prom]# curl -X POST "http://10.2.2.86:9090/-/reload"

```

查看promtheus发现

The screenshot shows the Prometheus UI with the 'Targets' section selected. It displays two groups of targets:

- coredns (2/2 up)**: Two targets are listed, both marked as UP. The first target is at `http://10.2.0.16:9153/metrics` with labels `instance="10.2.0.16:9153" job="coredns"`. The second target is at `http://10.2.0.17:9153/metrics` with labels `instance="10.2.0.17:9153" job="coredns"`.
- prometheus (1/1 up)**: One target is listed, marked as UP. The target is at `http://localhost:9090/metrics` with labels `instance="localhost:9090" job="prometheus"`.

使用exporter监控

刚才我们说了，有些应用自带的metrics接口，那么对于没有自带metrics接口的应用，我们可以使用各种exporter监控，官方已经给我们提供了非常多的exporter,具体可以去官网查阅，地址如下：

```
1 | https://prometheus.io/docs/instrumenting/exporters/
```

下面以mysql的exporter举例，具体的做法就是在每个mysql的pod里部署一个exporter服务来监控mysql的各项数据。

```
1 cat >mysql-prom.yaml <<EOF
2 apiVersion: apps/v1
3 kind: Deployment
4 metadata:
5   name: mysql-dp
6   namespace: prom
7 spec:
8   replicas: 1
9   selector:
10    matchLabels:
11      app: mysql
12   template:
13     metadata:
14       labels:
15         app: mysql
16     spec:
17       containers:
18         - name: mysql
19           image: mysql:5.7
20           ports:
21             - containerPort: 3306
22           env:
23             - name: MYSQL_ROOT_PASSWORD
24               value: "123456"
25             - name: mysql-exporter
26               image: prom/mysqld-exporter
27               ports:
28                 - containerPort: 9104
29               env:
30                 - name: DATA_SOURCE_NAME
31                   value: "root:123456@localhost:3306/"
32 ---
33 kind: Service
34 apiVersion: v1
35 metadata:
36   name: mysql-svc
37   namespace: prom
38 spec:
39   selector:
40     app: mysql
41   ports:
42     - name: mysql
43       port: 3306
44       targetPort: 3306
45     - name: mysql-prom
46       port: 9104
47       targetPort: 9104
48 EOF
```

应用后查看

```

1 [root@node1 prom]# kubectl apply -f mysql-prom.yaml
2 deployment.apps/mysql-dp created
3 service/mysql-svc created
4
5 [root@node1 prom]# kubectl -n prom get pod
6 NAME                      READY   STATUS    RESTARTS   AGE
7 mysql-dp-79b48cff96-m96bz   2/2     Running   0          73s
8 prometheus-796566c67c-1hrns 1/1     Running   0          46m
9
10 [root@node1 prom]# kubectl -n prom get svc
11 NAME        TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)           AGE
12 mysql-svc   ClusterIP   10.1.213.31   <none>          3306/TCP,9104/TCP   4m9s
13 prometheus  ClusterIP   10.1.95.152   <none>          9090/TCP          44m

```

修改prom配置文件

```

1 cat > prom-cm.yaml << EOF
2 apiVersion: v1
3 kind: ConfigMap
4 metadata:
5   name: prometheus-config
6   namespace: prom
7 data:
8   prometheus.yaml: |
9     global:
10       scrape_interval: 15s
11       scrape_timeout: 15s
12       scrape_configs:
13         - job_name: 'prometheus'
14           static_configs:
15             - targets: ['localhost:9090']
16
17         - job_name: 'coredns'
18           static_configs:
19             - targets: ['10.2.0.16:9153','10.2.0.17:9153']
20
21         - job_name: 'mysql'
22           static_configs:
23             - targets: ['mysql-svc:9104']
24 EOF

```

更新配置

```

1 [root@node1 prom]# kubectl apply -f prom-cm.yaml
2 configmap/prometheus-config configured
3
4 [root@node1 prom]# curl -X POST "http://10.2.2.86:9090/-/reload"

```

查看promtheus

All Unhealthy

coredns (2/2 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://10.2.0.16:9153/metrics	UP	instance="10.2.0.16:9153" job="coredns"	11.283s	3.784ms	
http://10.2.0.17:9153/metrics	UP	instance="10.2.0.17:9153" job="coredns"	4.780s	3.202ms	

mysql (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://mysql-svc:9104/metrics	UP	instance="mysql-svc:9104" job="mysql"	4.105s	15.193ms	

prometheus (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9090/metrics	UP	instance="localhost:9090" job="prometheus"	8.624s	3.931ms	

第x章 监控集群节点

集群监控组件介绍

前面我们学习了如何监控k8s里的应用，但是k8s集群自己的状态信息监控也很重要，比如说以下资源信息：

- 1 k8s节点的运行状态：内存，cpu，磁盘，网络等。
- 2 k8s组件的运行状态：kube-apiserver,kube-controller,kube-scheduler,coredns等。
- 3 k8s资源的运行状态：Deployment,DaemonSet,Service,Ingress等。

对于k8s集群的监控，主要有以下几种方案：

- 1 cAdvisor：监控容器内部的资源使用情况，已经集成在kubelet里了
- 2 kube-state-metrics：监控k8s资源的使用状态
- 3 metrics-server：监控k8s的cpu，内存，磁盘，网络等状态

监控集群节点

传统架构监控节点运行状态我们有非常成熟的zabbix可以使用，在k8s里我们可以直接使用prometheus官方的node-exporter来获取节点的各种指标，比如cpu,mem,disk,net等信息。

项目地址：

```
1 | https://github.com/prometheus/node_exporter
```

因为是节点监控，所以我们采用DaemonSet模式，在每个节点都部署 node-exporter

```
1 | cat > prom-node-exporter.yaml << 'EOF'
2 | apiVersion: apps/v1
3 | kind: DaemonSet
4 | metadata:
5 |   name: node-exporter
```

```

6   namespace: prom
7   labels:
8     app: node-exporter
9   spec:
10    selector:
11      matchLabels:
12        app: node-exporter
13    template:
14      metadata:
15        labels:
16          app: node-exporter
17    spec:
18      hostPID: true           #使用主机的 pid 命名空间
19      hostIPC: true          #使用主机的 ipc 命名空间
20      hostNetwork: true      #用主机的网络命名空间
21      nodeSelector:
22        kubernetes.io/os: linux
23    containers:
24      - name: node-exporter
25        image: prom/node-exporter:v1.1.1
26        args:
27          - --web.listen-address=$(HOSTIP):9100      #监控指标数据的地址端口号
28          - --path.procfs=/host/proc                #监控proc路径
29          - --path.sysfs=/host/sys                  #监控sys路径
30          - --path.rootfs=/host/root               #监控/rootfs路径
31          - --collector.filesystem.ignored-mount-
32            points=^(dev|proc|sys|var/lib/docker/.+)(\$|/)      #忽略监控的磁盘信息
33            - --collector.filesystem.ignored-fs-
34            types=^(autofs|binfmt_misc|cgroup|configfs|debugfs|devpts|devtmpfs|fusectl|hu
35              getlbf|mqueue|overlay|proc|procfs|pstore|rpc_pipefs|securityfs|sysfs|tracefs
36            )$      #忽略监控的文件系统
37            ports:
38              - containerPort: 9100
39            env:
40              - name: HOSTIP
41                valueFrom:
42                  fieldRef:
43                    fieldPath: status.hostIP          #读取node本身的IP地址
44            resources:
45              requests:
46                cpu: 150m
47                memory: 180Mi
48              limits:
49                cpu: 150m
50                memory: 180Mi
51            securityContext:                      #全上下文，用于定义
52              Container的权限和访问控制
53              runAsNonRoot: true                 #容器不以root运行
54              runAsUser: 65534                  #使用指定的uid用户运行
55            volumeMounts:
56              - name: proc
57                mountPath: /host/proc
58              - name: sys
59                mountPath: /host/sys
60              - name: root
61                mountPath: /host/root
62                mountPropagation: HostToContainer #mountPropagation确定挂载如何从主
63                机传播到容器

```

```

58         readOnly: true
59     tolerations: #容忍
60     - operator: "Exists" #Exists相当于值的通配符, 因此pod可以容忍
61     特定类别的所有污点。
62     volumes:
63     - name: proc
64         hostPath:
65             path: /proc
66     - name: dev
67         hostPath:
68             path: /dev
69     - name: sys
70         hostPath:
71             path: /sys
72     - name: root
73         hostPath:
74             path: /
74 EOF

```

配置文件解释:

```

1 #获取pod信息
2 https://kubernetes.io/docs/tasks/inject-data-application/environment-
variable-expose-pod-information/
3
4 env:
5 - name: HOSTIP
6   valueFrom:
7     fieldRef:
8       fieldPath: status.hostIP #读取node本身的IP地址
9 -----
10 securityContext: #全上下文, 用于定义Container的权限和访问控制
11   runAsNonRoot: true #容器不以root运行
12   runAsUser: 65534 #使用指定的uid用户运行
13 -----
14 tolerations: #容忍
15 - operator: "Exists" #Exists相当于值的通配符, 因此pod可以容忍特定类别的
16 所有污点。
17 -----
18 securityContext: #全上下文, 用于定义Container的权限和访问控制
19   runAsNonRoot: true #容器不以root运行
20   runAsUser: 65534 #使用指定的uid用户运行
21 -----
22 hostPID: true #使用主机的 pid 命名空间
23 hostIPC: true #使用主机的 ipc 命名空间
24 hostNetwork: true #用主机的网络命名空间

```

应用资源配置

```

1 [root@node1 prom]# kubectl apply -f prom-node-exporter.yaml
2 daemonset.apps/node-exporter created

```

查看pod创建状态

```
1 [root@node1 prom]# kubectl -n prom get pod
2 NAME                      READY   STATUS    RESTARTS   AGE
3 mysql-dp-79b48cff96-m96bz   2/2     Running   0          15m
4 node-exporter-7zhjf        1/1     Running   0          41s
5 node-exporter-kzqvz        1/1     Running   0          41s
6 node-exporter-zbsrl       1/1     Running   0          41s
7 prometheus-796566c67c-lhrns 1/1     Running   0          60m
```

服务发现

我们已经在每个节点上部署了node-exporter，那我们要如何采集到的每个节点的数据呢？如果按照我们上面的操作方法写死IP工作量太大，写service的话只会显示一条数据，这两种方式都不太方便。那么有什么好办法可以解决这种情况吗？这种时候就需要使用prometheus的服务发现功能了。

在k8s里，prometheus通过与k8s的API集成，支持5种服务发现模式，分别为：Node、Service、Pod、Endpoints、Ingress

官方文档：

```
1 | https://prometheus.io/docs/prometheus/latest/configuration/configuration/#kubernetes\_sd\_config
```

node节点自动发现

修改prometheus配置

```
1 cat > prom-cm.yaml << 'EOF'
2 apiVersion: v1
3 kind: ConfigMap
4 metadata:
5   name: prometheus-config
6   namespace: prom
7 data:
8   prometheus.yaml: |
9     global:
10       scrape_interval: 15s
11       scrape_timeout: 15s
12       scrape_configs:
13         - job_name: 'prometheus'
14           static_configs:
15             - targets: ['localhost:9090']
16
17         - job_name: 'coredns'
18           static_configs:
19             - targets: ['10.2.0.16:9153', '10.2.0.17:9153']
20
21         - job_name: 'mysql'
22           static_configs:
23             - targets: ['mysql-svc:9104']
24
25         - job_name: 'nodes'
26           kubernetes_sd_configs:      #k8s自动服务发现
27             - role: node            #自动发现类型为node
28 EOF
```

生效配置

```
1 [root@node1 prom]# kubectl apply -f prom-cm.yaml
2 configmap/prometheus-config configured
3
4 [root@node1 prom]# curl -X POST "http://10.2.2.86:9090/-/reload"
```

查看prometheus

mysql (1/1 up)					
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://mysql-svc:9104/metrics	UP	instance="mysql-svc:9104" job="mysql"	14.245s	19.124ms	
nodes (0/3 up)					
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://10.0.0.12:10250/metrics	DOWN	instance="node2" job="nodes"	5.960s	0.580ms	server returned HTTP status 400 Bad Request
http://10.0.0.13:10250/metrics	DOWN	instance="node3" job="nodes"	4.741s	1.120ms	server returned HTTP status 400 Bad Request
http://10.0.0.11:10250/metrics	DOWN	instance="node1" job="nodes"	9.265s	2.125ms	server returned HTTP status 400 Bad Request
prometheus (1/1 up)					
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9090/metrics	UP	instance="localhost:9090" job="prometheus"	3.767s	3.481ms	

这是发现提示有问题，因为prometheus默认去访问的是10250，我们需要把10250替换为9100

这里就需要使用prometheus提供的relabel_configs的replace功能。replace可以在采集数据之前通过Target的Metadata信息，动态重写Label的值。我们这里可以把_address_这个标签的端口修改为9100
relabel_config配置官方说明：

```
1 | https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config
```

修改prometheus配置：

```
1 cat > prom-cm.yaml << EOF
2 apiVersion: v1
3 kind: ConfigMap
4 metadata:
5   name: prometheus-config
6   namespace: prom
7 data:
8   prometheus.yml: |
9     global:
10       scrape_interval: 15s
11       scrape_timeout: 15s
12       scrape_configs:
13         - job_name: 'prometheus'
14           static_configs:
15             - targets: ['localhost:9090']
16
17         - job_name: 'coredns'
18           static_configs:
19             - targets: ['10.2.0.16:9153', '10.2.0.17:9153']
20
21         - job_name: 'mysql'
```

```

22     static_configs:
23       - targets: ['mysql-svc:9104']
24
25       - job_name: 'nodes'
26         kubernetes_sd_configs:
27           - role: node
28             relabel_configs: #配置重写
29               - action: replace #基于正则表达式匹配
30                 执行的操作
31                   source_labels: ['__address__'] #从源标签里选择值
32                     regex: '(.*):10250' #提取的值与之匹配的正则
33
34 表达式
35             replacement: '${1}:9100' #执行正则表达式替换的值
36             target_label: __address__ #结果值在替换操作中写入的标签
37
38 EOF

```

生效配置：

```

1 [root@node1 prom]# kubectl apply -f prom-cm.yaml
2 configmap/prometheus-config configured
3
4 等待一会再更新
5 [root@node1 prom]# curl -X POST "http://10.2.2.86:9090/-/reload"

```

prometheus查看：

mysql (1/1 up)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://mysql-svc:9104/metrics	UP	instance="mysql-svc:9104" job="mysql"	5.366s	15.673ms	

nodes (3/3 up)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://10.0.0.13:9100/metrics	UP	instance="node3" job="nodes"	5.327s	18.312ms	
http://10.0.0.11:9100/metrics	UP	instance="node1" job="nodes"	1.578s	111.167ms	
http://10.0.0.12:9100/metrics	UP	instance="node2" job="nodes"	758.000ms	21.670ms	

这里还有一个问题，就是节点标签目前只有主机名和任务名，不适合我们后期分组查询数据时候，所以我们需要把更多的标签添加进去，prometheus对于k8s的自动发现node模式支持以下的标签，详情可以查询官网：

```

1 | https://prometheus.io/docs/prometheus/latest/configuration/configuration/#kubernetes_sd_config

```

<kubernetes_sd_config>

Kubernetes SD configurations allow retrieving scrape targets from Kubernetes' REST API and always staying synchronized with the cluster state.

One of the following `role` types can be configured to discover targets:

`node`

The `node` role discovers one target per cluster node with the address defaulting to the Kubelet's HTTP port. The target address defaults to the first existing address of the Kubernetes node object in the address type order of `NodeInternalIP`, `NodeExternalIP`, `NodeLegacyHostIP`, and `NodeHostName`.

Available meta labels:

- `__meta_kubernetes_node_name`: The name of the node object.
- `__meta_kubernetes_node_label_<labelname>`: Each label from the node object.
- `__meta_kubernetes_node_labelpresent_<labelname>`: true for each label from the node object.
- `__meta_kubernetes_node_annotation_<annotationname>`: Each annotation from the node object.
- `__meta_kubernetes_node_annotationpresent_<annotationname>`: true for each annotation from the node object.
- `__meta_kubernetes_node_address_<address_type>`: The first address for each node address type, if it exists.

In addition, the `instance` label for the node will be set to the node name as retrieved from the API server.

标签解释：

- 1 `__meta_kubernetes_node_name`: 节点对象的名称。
- 2 `__meta_kubernetes_node_label`: 节点对象的每个标签
- 3 `__meta_kubernetes_node_annotation`: 每个节点的注释
- 4 `__meta_kubernetes_node_address`: 每个节点地址类型的第一个地址

修改配置：

```

1 apiVersion: v1
2 kind: ConfigMap
3 metadata:
4   name: prometheus-config
5   namespace: prom
6 data:
7   prometheus.yml: |
8     global:
9       scrape_interval: 15s
10      scrape_timeout: 15s
11      scrape_configs:
12        - job_name: 'prometheus'
13          static_configs:
14            - targets: ['localhost:9090']
15
16        - job_name: 'coredns'
17          static_configs:
18            - targets: ['10.2.0.16:9153', '10.2.0.17:9153']
19
20        - job_name: 'mysql'
21          static_configs:
22            - targets: ['mysql-svc:9104']
23
24        - job_name: 'nodes'
25          kubernetes_sd_configs:
26            - role: node
27          relabel_configs:
28            - action: replace
29              source_labels: ['__address__']
30              regex: '(.*):10250'
```

```

31     replacement: '${1}:9100'
32     target_label: __address__
33
34     - action: labelmap          #将正则表达式与所有标签名称匹配
35       regex: __meta_kubernetes_node_label_(.+)
      #提取符合正则匹配的标签，然后
      #交到Label里

```

配置解释：

```

1 官方说明：
2 https://prometheus.io/docs/prometheus/latest/configuration/configuration/#label\_config
3
4 添加一个动作为labelmap，作用是将符合__meta_kubernetes_node_label_(.)正则表达式提取
  的标签添加到Label里。

```

生效配置：

```

1 [root@node1 prom]# kubectl apply -f prom-cm.yaml
2 configmap/prometheus-config configured
3
4 等一会再更新
5 [root@node1 prom]# curl -X POST "http://10.2.2.86:9090/-/reload"

```

查看prometheus

nodes (3/3 up) show less					
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://10.0.0.12:9100/metrics	UP	<code>beta_kubernetes_io_arch="amd64"</code> <code>beta_kubernetes_io_os="linux"</code> <code>disktype="SSD" [instance="node2"]</code> <code>[job="nodes" kubernetes_io_arch="amd64"]</code> <code>kubernetes_io_hostname="node2"</code> <code>kubernetes_io_os="linux"</code>	1.376s	23.519ms	
http://10.0.0.13:9100/metrics	UP	<code>beta_kubernetes_io_arch="amd64"</code> <code>beta_kubernetes_io_os="linux"</code> <code>instance="node3" [job="nodes"]</code> <code>kubernetes_io_arch="amd64"</code> <code>kubernetes_io_hostname="node3"</code> <code>kubernetes_io_os="linux"</code>	5.547s	111.383ms	

kubelet节点自动发现

prometheus自动发现kubelet访问的是10250端口，但是必须是https协议，而且必须提供证书，我们可以直接使用k8s的证书。除此之外访问集群资源还需要相应的权限，还需要带上我们刚才为prometheus创建的service-account-token,实际上我们为prometheus创建的RBAC资源产生的secrets会以文件挂载的形式挂载到Pod里，所以我们查询的时候只要带上这个token就具备了查询集群资源的权限。另外我们还设置了跳过证书检查。

资源配置如下：

```

1 cat > prom-cm.yaml << 'EOF'
2 apiVersion: v1
3 kind: ConfigMap
4 metadata:
5   name: prometheus-config
6   namespace: prom
7 data:
8   prometheus.yaml: |

```

```

9   global:
10    scrape_interval: 15s
11    scrape_timeout: 15s
12    scrape_configs:
13      - job_name: 'prometheus'
14        static_configs:
15          - targets: ['localhost:9090']
16
17      - job_name: 'coredns'
18        static_configs:
19          - targets: ['10.2.0.16:9153', '10.2.0.17:9153']
20
21      - job_name: 'mysql'
22        static_configs:
23          - targets: ['mysql-svc:9104']
24
25      - job_name: 'nodes'
26        kubernetes_sd_configs:
27          - role: node
28        relabel_configs:
29          - source_labels: ['__address__']
30            regex: '(.*):10250'
31            replacement: '${1}:9100'
32            target_label: __address__
33            action: replace
34          - action: labelmap
35            regex: __meta_kubernetes_node_label_(.+)
36
37      - job_name: 'kubelet'
38        kubernetes_sd_configs:
39          - role: node
40        scheme: https
41        tls_config:
42          ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
43          insecure_skip_verify: true
44        bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
45        relabel_configs:
46          - action: labelmap
47            regex: __meta_kubernetes_node_label_(.+)
48 EOF

```

配置解释：

```

1  - job_name: 'kubelet'
2    kubernetes_sd_configs:
3      - role: node
4        scheme: https      #配置用于请求的协议
5        tls_config:         #tls配置
6          ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt      #ca证书
7          insecure_skip_verify: true      #禁用证书检查
8        bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
#serviceaccount授权， 默认securt会以文件形式挂载到pod里
9        relabel_configs:
10          - action: labelmap
11            regex: __meta_kubernetes_node_label_(.+)

```

应用配置：

```
1 [root@node1 prom]# kubectl apply -f prom-cm.yaml
2 configmap/prometheus-config configured
3
4 稍等一会配置
5 [root@node1 prom]# curl -X POST "http://10.2.2.86:9090/-/reload"
```

查看效果：

Prometheus					
Alerts Graph Status Help Classic UI					
kubelet (3/3 up) show less					
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
https://10.0.0.12:10250/metrics	UP	<code>beta_kubernetes_io_arch="amd64"</code> <code>beta_kubernetes_io_os="linux"</code> <code>disktype="SSD" instance="node2"</code> <code>job="kubelet" kubernetes_io_arch="amd64"</code> <code>kubernetes_io_hostname="node2"</code> <code>kubernetes_io_os="linux"</code>	1.551s	23.324ms	
https://10.0.0.13:10250/metrics	UP	<code>beta_kubernetes_io_arch="amd64"</code> <code>beta_kubernetes_io_os="linux"</code> <code>instance="node3" job="kubelet"</code> <code>kubernetes_io_arch="amd64"</code> <code>kubernetes_io_hostname="node3"</code> <code>kubernetes_io_os="linux"</code>	3.131s	31.238ms	
https://10.0.0.11:10250/metrics	UP	<code>app="traefik-ingress"</code> <code>beta_kubernetes_io_arch="amd64"</code> <code>beta_kubernetes_io_os="linux"</code> <code>instance="node1" job="kubelet"</code> <code>kubernetes_io_arch="amd64"</code> <code>kubernetes_io_hostname="node1"</code> <code>kubernetes_io_os="linux"</code>	6.111s	21.936ms	

第x章 容器监控

学习Docker的时候我们已经知道收集docker容器使用的是cAdvisor,而k8s的kubelet已经内置了cAdvisor。所以我们只需要访问即可，这里我们可以使用访问kubelet的暴露的地址访问cAdvisor数据。地址为：nodeip/metrics/cadvisor

配置文件：

```
1 apiVersion: v1
2 kind: ConfigMap
3 metadata:
4   name: prometheus-config
5   namespace: prom
6 data:
7   prometheus.yaml: |
8     global:
9       scrape_interval: 15s
10      scrape_timeout: 15s
11      scrape_configs:
12        - job_name: 'prometheus'
13          static_configs:
14            - targets: ['localhost:9090']
15
16        - job_name: 'coredns'
```

```

17     static_configs:
18     - targets: ['10.2.0.16:9153','10.2.0.17:9153']
19
20     - job_name: 'mysql'
21       static_configs:
22         - targets: ['mysql-svc:9104']
23
24     - job_name: 'nodes'
25       kubernetes_sd_configs:
26         - role: node
27       relabel_configs:
28         - source_labels: ['__address__']
29           regex: '(.*):10250'
30           replacement: '${1}:9100'
31           target_label: __address__
32           action: replace
33         - action: labelmap
34           regex: __meta_kubernetes_node_label_(.+)
35
36     - job_name: 'kubelet'
37       kubernetes_sd_configs:
38         - role: node
39       scheme: https
40       tls_config:
41         ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
42         insecure_skip_verify: true
43       bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
44       relabel_configs:
45         - action: labelmap
46           regex: __meta_kubernetes_node_label_(.+)
47
48     - job_name: 'cadvisor'
49       kubernetes_sd_configs:
50         - role: node
51       scheme: https
52       tls_config:
53         ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
54         insecure_skip_verify: true
55       bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
56       relabel_configs:
57         - action: labelmap
58           regex: __meta_kubernetes_node_label_(.+)
59           replacement: $1
60         - source_labels: [__meta_kubernetes_node_name]
61           regex: (.*)
62           replacement: /metrics/cadvisor
63           target_label: __metrics_path__

```

应用配置:

```

1 [root@node1 prom]# kubectl apply -f prom-cm.yaml
2 configmap/prometheus-config configured
3
4 [root@node1 prom]# curl -X POST "http://10.2.2.86:9090/-/reload"

```

查看结果:

Targets

All Unhealthy

cadvisor (3/3 up) [show less](#)

Endpoint	State	Labels	Last Scrape	Scrape Duration
https://10.0.0.11:10250/metrics/cadvisor or	UP	<code>app="traefik-ingress"</code> <code>beta_kubernetes_io_arch="amd64"</code> <code>beta_kubernetes_io_os="linux"</code> <code>instance="node1" job="cAdvisor"</code> <code>kubernetes_io_arch="amd64"</code> <code>kubernetes_io_hostname="node1"</code> <code>kubernetes_io_os="linux"</code>	24.830s	88.742ms
https://10.0.0.12:10250/metrics/cadvisor or	UP	<code>beta_kubernetes_io_arch="amd64"</code> <code>beta_kubernetes_io_os="linux"</code> <code>disktype="SSD" instance="node2"</code> <code>job="cAdvisor"</code> <code>kubernetes_io_arch="amd64"</code> <code>kubernetes_io_hostname="node2"</code> <code>kubernetes_io_os="linux"</code>	23.430s	48.214ms
https://10.0.0.13:10250/metrics/cadvisor or	UP	<code>beta_kubernetes_io_arch="amd64"</code> <code>beta_kubernetes_io_os="linux"</code> <code>instance="node3" job="cAdvisor"</code> <code>kubernetes_io_arch="amd64"</code> <code>kubernetes_io_hostname="node3"</code> <code>kubernetes_io_os="linux"</code>	21.402s	36.157ms

关于查询的数据可以查看官网的说明：

```
1 | https://github.com/google/cadvisor/blob/master/docs/storage/prometheus.md
```

第x章 API Server监控

API Server是k8s的核心组件，对于API Server的监控我们可以直接通过k8s的Service来获取：

```
1 | [root@node1 prom]# kubectl get svc
2 |   NAME      TYPE      CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
3 |   kubernetes  ClusterIP  10.1.0.1     <none>        443/TCP   10d
```

修改配置：

```
1 |   - job_name: 'apiservers'
2 |     kubernetes_sd_configs:
3 |       - role: endpoints
```

应用配置：

```
1 | kubectl apply -f prom-cm.yaml
2 | curl -X POST "http://10.2.2.86:9090/-/reload"
```

查看结果：

Targets

All Unhealthy

apiservers (4/10 up) show less

Endpoint	State	Labels
http://10.2.1.103/metrics	DOWN	instance="10.2.1.103:80" job="apiservers"
http://10.2.0.16:53/metrics	DOWN	instance="10.2.0.16:53" job="apiservers"
http://10.2.0.17:53/metrics	DOWN	instance="10.2.0.17:53" job="apiservers"
http://10.0.0.11:6443/metrics	DOWN	instance="10.0.0.11:6443" job="apiservers"
http://10.2.2.86:9090/metrics	UP	instance="10.2.2.86:9090" job="apiservers"
http://10.2.1.104:9104/metrics	UP	instance="10.2.1.104:9104" job="apiservers"
http://10.2.0.16:9153/metrics	UP	instance="10.2.0.16:9153" job="apiservers"
http://10.2.0.17:9153/metrics	UP	instance="10.2.0.17:9153" job="apiservers"
http://10.0.0.12:4443/metrics	DOWN	instance="10.0.0.12:4443" job="apiservers"
http://10.2.1.104:3306/metrics	DOWN	instance="10.2.1.104:3306" job="apiservers"

这时我们发现prometheus把所有的endpoint都找出来了，那么哪个才是我们需要的呢？通过查看API Server的svc可以发现API Server的通讯端口是6443，所以6443端口的服务才是我们需要的。

```

1 [root@node1 prom]# kubectl describe svc kubernetes
2 Name:           kubernetes
3 Namespace:      default
4 Labels:         component=apiserver
5               provider=kubernetes
6 Annotations:   <none>
7 Selector:      <none>
8 Type:          ClusterIP
9 IP:            10.1.0.1
10 Port:         https 443/TCP
11 TargetPort:   6443/TCP
12 Endpoints:    10.0.0.11:6443
13 Session Affinity: None
14 Events:       <none>
```

要想保留我们发现的的API Server，那么就需要查看他都有什么标签，然后将拥有这些标签的服务保留下来。

apiservers (4/10 up) [show less](#)

Endpoint	State	Labels	Last Scrape
http://10.2.0.16:53/metrics	DOWN	instance="10.2.0.16:53" job="apiservers"	16.79s
http://10.2.0.17:53/metrics	DOWN	instance="10.2.0.17:53" job="apiservers"	11.190s
http://10.0.0.11:6443/metrics	DOWN	instance="10.0.0.11:6443" job="apiservers"	10.798s
http://10.2.2.86:9090/metrics	UP		
http://10.2.1.103/metrics	DOWN		
http://10.2.0.16:9153/metrics	UP		
http://10.2.0.17:9153/metrics	UP		
http://10.0.0.12:4443/metrics	DOWN		
http://10.2.1.104:3306/metrics	DOWN		
http://10.2.1.104:9104/metrics	UP		

[cadvisor \(3/3 up\)](#) [show more](#)

[coredns \(2/2 up\)](#) [show more](#)

我们需要匹配的条件是标签名为`_meta_kubernetes_service_label_component`的值为"apiserver"的服务。

因为这个端口是https协议的，所以我们还需要带上认证的证书。

修改配置文件：

```
1 apiVersion: v1
2 kind: ConfigMap
3 metadata:
4   name: prometheus-config
5   namespace: prom
6 data:
7   prometheus.yml: |
8     global:
9       scrape_interval: 15s
10      scrape_timeout: 15s
11      scrape_configs:
12        - job_name: 'prometheus'
13          static_configs:
14            - targets: ['localhost:9090']
15
16        - job_name: 'coredns'
17          static_configs:
18            - targets: ['10.2.0.16:9153', '10.2.0.17:9153']
19
20        - job_name: 'mysql'
21          static_configs:
22            - targets: ['mysql-svc:9104']
23
24        - job_name: 'nodes'
25          kubernetes_sd_configs:
26            - role: node
27          relabel_configs:
28            - source_labels: ['__address__']
29              regex: '(.*):10250'
```

```

30     replacement: '${1}:9100'
31     target_label: __address__
32     action: replace
33   - action: labelmap
34     regex: __meta_kubernetes_node_label_(.+)
35
36   - job_name: 'kubelet'
37     kubernetes_sd_configs:
38       - role: node
39     scheme: https
40     tls_config:
41       ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
42       insecure_skip_verify: true
43     bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
44     relabel_configs:
45       - action: labelmap
46         regex: __meta_kubernetes_node_label_(.+)
47
48   - job_name: 'cadvisor'
49     kubernetes_sd_configs:
50       - role: node
51     scheme: https
52     tls_config:
53       ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
54       insecure_skip_verify: true
55     bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
56     relabel_configs:
57       - action: labelmap
58         regex: __meta_kubernetes_node_label_(.+)
59         replacement: $1
60       - source_labels: [__meta_kubernetes_node_name]
61         regex: (.*)
62         replacement: /metrics/cadvisor
63         target_label: __metrics_path__
64
65   - job_name: 'apiservers'
66     kubernetes_sd_configs:
67       - role: endpoints
68     scheme: https
69     tls_config:
70       ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
71       insecure_skip_verify: true
72     bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
73     relabel_configs:
74       - source_labels: [__meta_kubernetes_service_label_component]
75         action: keep
76         regex: apiserver

```

应用修改:

```

1 | kubectl apply -f prom-cm.yaml
2 | curl -X POST "http://10.2.2.86:9090/-/reload"

```

查看结果:

Targets

All Unhealthy

apiservers (1/1 up) [show less](#)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
https://10.0.0.11:6443/metrics	UP	<code>instance="10.0.0.11:6443"</code> <code>job="apiservers"</code>	8.771s	1.648s	

[cadvisor \(3/3 up\)](#) [show more](#)
[coredns \(2/2 up\)](#) [show more](#)
[kubelet \(3/3 up\)](#) [show more](#)
[mysql \(1/1 up\)](#) [show more](#)
[nodes \(3/3 up\)](#) [show more](#)
[prometheus \(1/1 up\)](#) [show more](#)

第x章 Pod监控

我们这里采集到的Pod监控也是使用自动发现Endpoints。只不过这里需要做一些匹配的处理.

配置如下：

```
1   - job_name: 'pods'
2     kubernetes_sd_configs:
3       - role: endpoints
4         relabel_configs:
5           - source_labels:
6             [__meta_kubernetes_service_annotation_prometheus_io_scrape]
7               action: keep
8               regex: true
9             - action: labelmap
10               regex: __meta_kubernetes_service_label_(.+)
11             - source_labels: [__address__,
12               __meta_kubernetes_service_annotation_prometheus_io_port]
13                 action: replace
14                 target_label: __address__
15                 regex: ([^:])(?::\d+)?;(\d+)
16                 replacement: $1:$2
17             - source_labels: [__meta_kubernetes_namespace]
18               action: replace
19               target_label: kubernetes_namespace
20             - source_labels: [__meta_kubernetes_service_name]
21               action: replace
22               target_label: kubernetes_name
23             - source_labels: [__meta_kubernetes_pod_name]
24               action: replace
25               target_label: kubernetes_pod_name
```

生效配置：

```
1 | kubectl apply -f prom-cm.yaml
2 | curl -X POST "http://10.2.2.86:9090/-/reload"
```

查看效果：

pods (2/2 up) [show less](#)

Endpoint	State	Labels	Last Scrape	Scrape Duration
http://10.2.0.17:9153/metrics	UP	<code>instance="10.2.0.17:9153" job="pods"</code> <code>k8s_app="kube-dns"</code> <code>kubernetes_io_cluster_service="true"</code> <code>kubernetes_io_name="KubeDNS"</code> <code>kubernetes_name="kube-dns"</code> <code>kubernetes_namespace="kube-system"</code> <code>kubernetes_pod_name="coredns-6d56c8448f-ckwhg"</code>	1.977s	11.130ms
http://10.2.0.16:9153/metrics	UP	<code>instance="10.2.0.16:9153" job="pods"</code> <code>k8s_app="kube-dns"</code> <code>kubernetes_io_cluster_service="true"</code> <code>kubernetes_io_name="KubeDNS"</code> <code>kubernetes_name="kube-dns"</code> <code>kubernetes_namespace="kube-system"</code> <code>kubernetes_pod_name="coredns-6d56c8448f-rvmdf"</code>	5.823s	4.332ms

匹配参数解释：

- 1 `__meta_kubernetes_service_annotation_prometheus_io_scrape` 为 `true`
- 2 `__address__` 的端口和 `__meta_kubernetes_service_annotation_prometheus_io_port` 的端口一样

自动发现原理：

我们创建svc的时候添加了prometheus和metrics端口的注解，这样就能被prometheus自动发现到

```
1 [root@node1 prom]# kubectl -n kube-system get svc kube-dns -o yaml
2 apiVersion: v1
3 kind: Service
4 metadata:
5   annotations:
6     prometheus.io/port: "9153"
7     prometheus.io/scrape: "true"
8   labels:
9     k8s-app: kube-dns
10    kubernetes.io/cluster-service: "true"
11    kubernetes.io/name: KubeDNS
```

对应promethues自动发现里的数据：

```
__meta_kubernetes_service_annotation_prometheus_io_port="9153"
__meta_kubernetes_service_annotation_prometheus_io_scrape="true"
__meta_kubernetes_service_annotationpresent_prometheus_io_port="true"
__meta_kubernetes_service_annotationpresent_prometheus_io_scrape="true"
```

那也就意味着以后发布的应用，我们只需要添加这两条注解就可以被自动发现了，现在我们可以来修改下刚才创建的mysql配置，添加相关注解就可以自动被发现了。

```
1 ---
2   kind: Service
3   apiVersion: v1
4   metadata:
5     name: mysql-svc
6     namespace: prom
```

```

7 annotations:
8   prometheus.io/scrape: "true"
9   prometheus.io/port: "9104"
10 spec:
11   selector:
12     app: mysql
13   ports:
14     - name: mysql
15       port: 3306
16       targetPort: 3306
17     - name: mysql-prom
18       port: 9104
19       targetPort: 9104

```

查看prometheus可以发现mysql的pod已经自动被发现了

[pods \(3/3 up\)](#) [show less](#)

Endpoint	State	Labels	Last Scrape	Scrape Duration
http://10.2.1.104:9104/metrics	UP	<code>instance="10.2.1.104:9104" job="pods"</code> <code>kubernetes_name="mysql-svc"</code> <code>kubernetes_namespace="prom"</code> <code>kubernetes_pod_name="mysql-dp-79b48cff96-m96bz"</code>	2.228s	19.243ms
http://10.2.0.16:9153/metrics	UP	<code>instance="10.2.0.16:9153" job="pods"</code> <code>k8s_app="kube-dns"</code> <code>kubernetes_io_cluster_service="true"</code> <code>kubernetes_io_name="KubeDNS"</code> <code>kubernetes_name="kube-dns"</code> <code>kubernetes_namespace="kube-system"</code> <code>kubernetes_pod_name="coredns-6d56c8448f-rvmdf"</code>	10.617s	5.563ms
http://10.2.0.17:9153/metrics	UP	<code>instance="10.2.0.17:9153" job="pods"</code> <code>k8s_app="kube-dns"</code> <code>kubernetes_io_cluster_service="true"</code> <code>kubernetes_io_name="KubeDNS"</code> <code>kubernetes_name="kube-dns"</code> <code>kubernetes_namespace="kube-system"</code> <code>kubernetes_pod_name="coredns-6d56c8448f-ckwhg"</code>	6.771s	3.110ms

现在就可以删除刚才我们配置的静态mysql了，同理，我们刚才静态配置的coredns也可以删掉了，同样使用自动发现来处理，配置如下：

```

1 apiVersion: v1
2 kind: ConfigMap
3 metadata:
4   name: prometheus-config
5   namespace: prom
6 data:
7   prometheus.yml: |
8     global:
9       scrape_interval: 15s
10      scrape_timeout: 15s
11      scrape_configs:
12        - job_name: 'prometheus'
13          static_configs:
14            - targets: ['localhost:9090']
15
16        #- job_name: 'coredns'
17        #  static_configs:
18        #    - targets: ['10.2.0.16:9153','10.2.0.17:9153']

```

```
19
20
21     #- job_name: 'mysql'
22     # static_configs:
23     #   - targets: ['mysql-svc:9104']
24
25     - job_name: 'nodes'
26       kubernetes_sd_configs:
27         - role: node
28         relabel_configs:
29           - source_labels: ['__address__']
30             regex: '(.*):10250'
31             replacement: '${1}:9100'
32             target_label: __address__
33             action: replace
34           - action: labelmap
35             regex: __meta_kubernetes_node_label_(.+)
36
37     - job_name: 'kubelet'
38       kubernetes_sd_configs:
39         - role: node
40         scheme: https
41         tls_config:
42           ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
43           insecure_skip_verify: true
44         bearer_token_file:
45           /var/run/secrets/kubernetes.io/serviceaccount/token
46         relabel_configs:
47           - action: labelmap
48             regex: __meta_kubernetes_node_label_(.+)
49
50     - job_name: 'cadvisor'
51       kubernetes_sd_configs:
52         - role: node
53         scheme: https
54         tls_config:
55           ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
56           insecure_skip_verify: true
57         bearer_token_file:
58           /var/run/secrets/kubernetes.io/serviceaccount/token
59         relabel_configs:
60           - action: labelmap
61             regex: __meta_kubernetes_node_label_(.+)
62             replacement: $1
63           - source_labels: [__meta_kubernetes_node_name]
64             regex: (.*)
65             replacement: /metrics/cadvisor
66             target_label: __metrics_path__
67
68     - job_name: 'apiservers'
69       kubernetes_sd_configs:
70         - role: endpoints
71         scheme: https
72         tls_config:
73           ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
74           insecure_skip_verify: true
75         bearer_token_file:
76           /var/run/secrets/kubernetes.io/serviceaccount/token
```

```
74     relabel_configs:
75       - source_labels: [__meta_kubernetes_service_label_component]
76         action: keep
77         regex: apiserver
78
79       - job_name: 'pods'
80         kubernetes_sd_configs:
81           - role: endpoints
82         relabel_configs:
83           - source_labels:
84             [__meta_kubernetes_annotation_prometheus_io_scrape]
85               action: keep
86               regex: true
87             - action: labelmap
88               regex: __meta_kubernetes_service_label_(.+)
89             - source_labels: [__address__,
90               __meta_kubernetes_annotation_prometheus_io_port]
91               action: replace
92               target_label: __address__
93               regex: ([^:]+)(?:\d+)?;(\d+)
94               replacement: $1:$2
95             - source_labels: [__meta_kubernetes_namespace]
96               action: replace
97               target_label: kubernetes_namespace
98             - source_labels: [__meta_kubernetes_service_name]
99               action: replace
100              target_label: kubernetes_name
101            - source_labels: [__meta_kubernetes_pod_name]
102              action: replace
103              target_label: kubernetes_pod_name
104
105            - job_name: 'coredns'
106              kubernetes_sd_configs:
107                - role: endpoints
108                relabel_configs:
109                  - source_labels:
110                    [__meta_kubernetes_annotation_prometheus_io_scrape]
111                      action: keep
112                      regex: true
113                      - action: labelmap
114                        regex: __meta_kubernetes_service_label_(.+)
115                      - source_labels: [__address__,
116                        __meta_kubernetes_annotation_prometheus_io_port]
117                        action: replace
118                        target_label: __address__
119                        regex: ([^:]+)(?:\d+)?;(\d+)
120                        replacement: $1:$2
121                      - source_labels: [__meta_kubernetes_namespace]
122                        action: replace
123                        target_label: kubernetes_namespace
124                      - source_labels: [__meta_kubernetes_service_name]
125                        action: replace
126                        target_label: kubernetes_name
127                      - source_labels: [__meta_kubernetes_pod_name]
128                        action: replace
129                        target_label: kubernetes_pod_name
130                      - source_labels: [__meta_kubernetes_endpoints_name]
131                        action: keep
```

应用配置：

```
1 | kubectl apply -f prom-cm.yaml
2 | curl -X POST "http://10.2.2.86:9090/-/reload"
```

查看结果：

coredns (2/2 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration
http://10.2.0.16:9153/metrics	UP	instance="10.2.0.16:9153" job="coredns" k8s_app="kube-dns" kubernetes_io_cluster_service="true" kubernetes_io_name="KubeDNS" kubernetes_name="kube-dns" kubernetes_namespace="kube-system" kubernetes_pod_name="coredns-6d56c8448f-rvmdf"	2m 15s	3.630ms
http://10.2.0.17:9153/metrics	UP	instance="10.2.0.17:9153" job="coredns" k8s_app="kube-dns" kubernetes_io_cluster_service="true" kubernetes_io_name="KubeDNS" kubernetes_name="kube-dns" kubernetes_namespace="kube-system" kubernetes_pod_name="coredns-6d56c8448f-ckwhg"	2m 24s	3.802ms

第x章 监控k8s资源对象

kube-state-metrics

我们刚才自动发现使用的是endpoints，监控的都是应用数据以，但是在k8s内部的pod,deployment,daemonset等资源也需要监控，比如当前有多少个pod，pod状态是什么样等等。

这些指标数据需要新的exporter来提供，那就是kube-state-metrics

项目地址：

```
1 | https://github.com/kubernetes/kube-state-metrics
```

安装kube-state-metrics

克隆代码：

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

修改配置：

这里我们主要修改镜像地址和添加自动发现的注解

```
1 | cd kube-state-metrics/examples/standard
2 | vim deployment.yaml
3 | -----
4 | - image: bitnami/kube-state-metrics:2.1.1
5 | -----
6 |
```

```
7 vim service.yaml
8 -----
9 apiVersion: v1
10 kind: Service
11 metadata:
12   labels:
13     app.kubernetes.io/name: kube-state-metrics
14     app.kubernetes.io/version: 2.1.1
15   annotations:
16     prometheus.io/scrape: "true"
17     prometheus.io/port: "8080"
18   name: kube-state-metrics
19   namespace: kube-system
20   .....
21 -----
```

创建资源配置：

```
1 [root@node1 standard]# kubectl apply -f ./kube-state-metrics
2 clusterrolebinding.rbac.authorization.k8s.io/kube-state-metrics created
3 clusterrole.rbac.authorization.k8s.io/kube-state-metrics created
4 deployment.apps/kube-state-metrics created
5 serviceaccount/kube-state-metrics created
6 service/kube-state-metrics created
```

查看prometheus：

http://10.2.1.105:8080/metrics	UP	app_kubernetes_io_name="kube-state-metrics" app_kubernetes_io_version="2.1.1" instance="10.2.1.105:8080" job="pods" kubernetes_name="kube-state-metrics" kubernetes_namespace="kube-system" kubernetes_pod_name="kube-state-metrics-686b48f8-zk9xw"	10.635s	8.790ms
--------------------------------	----	--	---------	---------

应用场景：

- 1 存在执行失败的Job: kube_job_status_failed
- 2 集群节点状态错误: kube_node_status_condition{condition="Ready", status!="true"}==1
- 3 集群中存在启动失败的 Pod: kube_pod_status_phase{phase=~"Failed|Unknown"}==1
- 4 最近30分钟内有 Pod 容器重启:
changes(kube_pod_container_status_restarts_total[30m])>0

更多使用方法可以查看官方文档：

```
1 | https://github.com/kubernetes/kube-state-metrics
```

第x章 grafana

grafana介绍

```
1 |
```

安装部署

```
1 cat > grafana.yml << 'EOF'
2 apiVersion: apps/v1
3 kind: Deployment
4 metadata:
5   name: grafana
6   namespace: prom
7 spec:
8   selector:
9     matchLabels:
10    app: grafana
11   template:
12     metadata:
13       labels:
14         app: grafana
15   spec:
16     volumes:
17     - name: storage
18       hostPath:
19         path: /data/k8s/grafana/
20   nodeSelector:
21     kubernetes.io/hostname: node2
22   securityContext:
23     runAsUser: 0
24   containers:
25     - name: grafana
26       image: grafana/grafana:7.4.3
27       imagePullPolicy: IfNotPresent
28       ports:
29         - containerPort: 3000
30           name: grafana
31       env:
32         - name: GF_SECURITY_ADMIN_USER
33           value: admin
34         - name: GF_SECURITY_ADMIN_PASSWORD
35           value: admin
36       readinessProbe:
37         failureThreshold: 10
38         httpGet:
39           path: /api/health
40           port: 3000
41           scheme: HTTP
42           initialDelaySeconds: 60
43           periodSeconds: 10
44           successThreshold: 1
45           timeoutSeconds: 30
46       livenessProbe:
47         failureThreshold: 3
48         httpGet:
49           path: /api/health
50           port: 3000
51           scheme: HTTP
52           periodSeconds: 10
53           successThreshold: 1
54           timeoutSeconds: 3
55       resources:
56         limits:
57           cpu: 150m
```

```

58      memory: 512Mi
59      requests:
60          cpu: 150m
61          memory: 512Mi
62      volumeMounts:
63          - mountPath: /var/lib/grafana
64          name: storage
65  ---
66  apiVersion: v1
67  kind: Service
68  metadata:
69      name: grafana
70      namespace: prom
71  spec:
72      ports:
73          - port: 3000
74      selector:
75          app: grafana
76  ---
77  apiVersion: networking.k8s.io/v1
78  kind: Ingress
79  metadata:
80      name: grafana
81      namespace: prom
82      labels:
83          app: grafana
84  spec:
85      rules:
86          - host: grafana.k8s.com
87              http:
88                  paths:
89                      - path: /
90                          pathType: ImplementationSpecific
91                      backend:
92                          service:
93                              name: grafana
94                          port:
95                              number: 3000
96 EOF

```

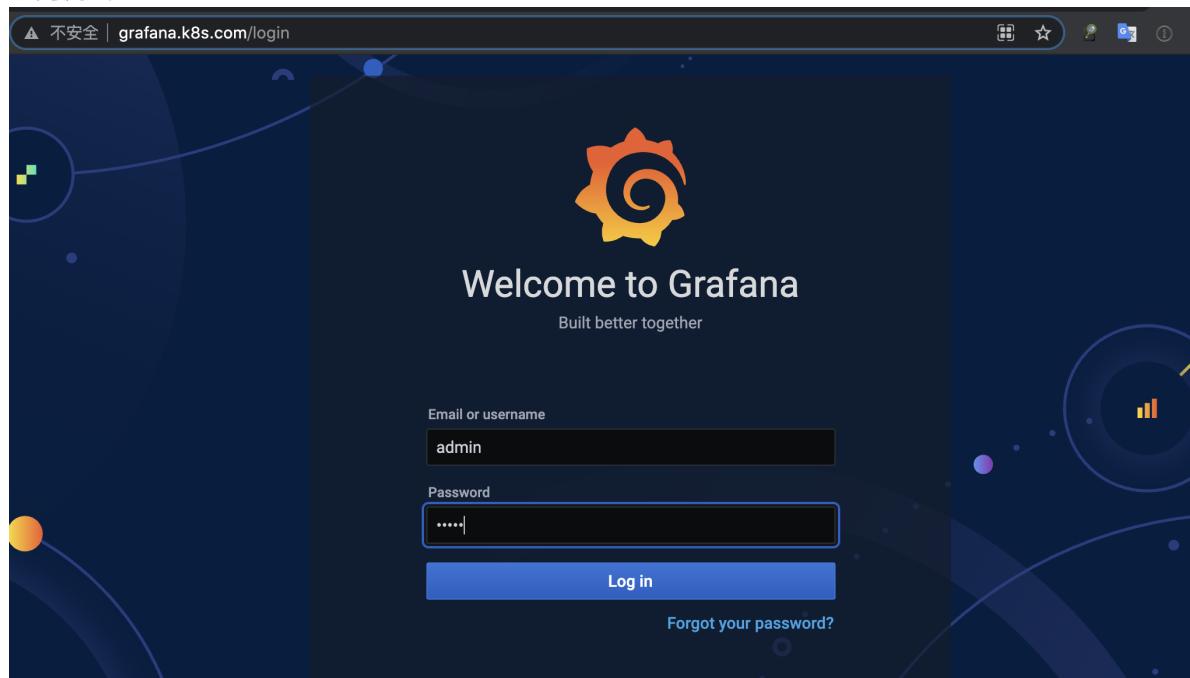
应用资源配置：

```

1 [root@node1 prom]# kubectl apply -f grafana.yaml
2 deployment.apps/grafana created
3 service/grafana created
4 ingress.networking.k8s.io/prometheus configured

```

访问测试：



添加数据源

A screenshot of the Grafana Configuration page. The title bar says "▲ 不安全 | grafana.k8s.com/datasources". The main header is "Configuration" with "Organization: Main Org.". On the left is a sidebar with icons for Home, Search, Add, Grid, Refresh, and Help. The "Data Sources" option is highlighted with a red box. The main content area shows a message "There are no data sources defined yet" and a large blue "Add data source" button. Below it is a pro tip: "ProTip: You can also define data sources through configuration files. [Learn more](#)". At the bottom are links for Documentation, Support, Community, Open Source, and the version v7.4.3 (010f20c1c8). The URL "grafana.k8s.com/datasources" is visible at the bottom left.

Configure your Prometheus data source below

Or skip the effort and get Prometheus (and Loki) as fully managed, scalable and hosted data sources from Grafana Labs with the [free-forever Grafana Cloud plan](#).

Name: Prometheus

Default:

HTTP

URL: (highlighted with a red box)

Access: Server (default)

Whitelisted Cookies: Add

Exemplars

+ Add

✓ Data source is working

Save & Test (highlighted with a red box)

Delete

Back

安装插件

grafana具有丰富的插件，这里我们使用一个非常强大的专门对k8s集群进行监控的插件：

DevOpsProdigy KubeGraf项目地址为：

```
1 | https://github.com/devopsprodigy/kubegraf/
```

安装这个插件需要我们进入grafana的pod内进行安装：

```
1 [root@node1 prom]# kubectl -n prom exec -it grafana-7f5b7455fc-z6ctx --  
/bin/bash  
2 bash-5.0# grafana-cli plugins install devopsprodigy-kubegraf-app  
3 installing devopsprodigy-kubegraf-app @ 1.5.2  
4 from: https://grafana.com/api/plugins/devopsprodigy-kubegraf-  
app/versions/1.5.2/download  
5 into: /var/lib/grafana/plugins  
6  
7 ✓ Installed devopsprodigy-kubegraf-app successfully  
8 installing grafana-piechart-panel @ 1.6.2  
9 from: https://grafana.com/api/plugins/grafana-piechart-  
panel/versions/1.6.2/download  
10 into: /var/lib/grafana/plugins  
11  
12 ✓ Installed grafana-piechart-panel successfully  
13 Installed dependency: grafana-piechart-panel ✓  
14
```

```
15 | Restart grafana after installing plugins . <service grafana-server restart>
16 |
17 | bash-5.0#
```

安装完成后我们还需要重启一下grafana才能生效，因为我们做了数据持久化，所以直接删除pod重新创建即可。

```
1 | [root@node1 prom]# kubectl -n prom delete pod grafana-7f5b7455fc-z6ctx
2 | pod "grafana-7f5b7455fc-z6ctx" deleted
```

重启之后我们在grafana页面激活插件

The screenshot shows the Grafana Configuration interface. On the left, there's a sidebar with icons for Configuration, Data Sources, Users, Teams, Plugins (which is highlighted with a red box), Preferences, and API Keys. The main area has tabs for Data Sources, Users, Teams, Plugins (which is active and highlighted with a yellow background), Preferences, and API Keys. A search bar at the top has the text 'dev'. Below it, a card for 'DevOpsProdigy KubeGraf' by DevOpsProdigy is shown, with a 'Signed' badge next to it. A blue button labeled 'Find more plugins on Grafana.com' is visible.

The screenshot shows the 'Plugins / DevOpsProdigy KubeGraf' configuration page. At the top, there's a logo for DevOpsProdigy and tabs for Readme, Config (which is active and highlighted with a yellow background), and Dashboards. Below the tabs, there's a 'Signed' badge. A note states: 'Grafana Labs checks each plugin to verify that it has a valid digital signature. Plugin signature verification is part of our security measures to ensure plugins are safe and trustworthy.' A link 'Read more about plugins signing' is provided. The main content area is titled 'DevOpsProdigy KubeGraf' and features a large blue 'Enable' button, which is highlighted with a red box.



Plugins / DevOpsProdigy KubeGraf

DevOpsProdigy

Readme

Config

Dashboards

Signed

Grafana Labs checks each plugin to verify that it has a valid digital signature. Plugin signature verification is part of our security measures to ensure plugins are safe and trustworthy.

[Read more about plugins signing](#)

DevOpsProdigy KubeGraf

Plugin is ready: Set up your first k8s-cluster

[Update](#)

[Disable](#)

这里需要对验证，我们使用kubectl的kubeconfig配置文件的内容来进行配置：

```
1 [root@node1 prom]# cat ~/.kube/config
2 apiVersion: v1
3 clusters:
4 - cluster:
5   certificate-authority-data:      #CA Cert的值
6   .....
7 kind: Config
8 preferences: {}
9 users:
10 - name: kubernetes-admin
11   user:
12     client-certificate-data:      #Client Cert的值
13     client-key-data:            #Client Key的值
14   .....
```

但是配置文件里的为base64编码后的，所以我们还需要进行解码，配置完成后的截图如下：



Add new cluster

Name

kubeadm-v1.19.3

HTTP

URL

https://kubernetes.default:443

Access

Server (default)

Help >

Whitelisted Cookies

Add Name

Add

Auth

Basic auth

With Credentials

TLS Client Auth

With CA Cert

Skip TLS Verify

Forward OAuth Identity

TLS Auth Details

CA Cert

```
DQEBCwUAA4IBAOBNrcDfI8yDMuo+IVf08tej6lx8w7Ab7kiGZejVha5Rv30hxY9  
M1ZgsISBeKRoOP+0t3q7952ez8r8HEOcdEzcw7LR25AVAOs20PI7cURHT5mP/PC  
c6xTyCtASVKK1veGtI/RB63S5DJrPoSdw7Ro0mkPeZxVSLm42cN0baSpfDgYSmX  
o2ejwPo1/5hvnAh0J5c800owoshSZSaA1sNgSRQLKytaTpFwG9XK+4TyYfHd3D77  
bYhH3wwCFBMMKnxu69BJPU3hBOlBO5f8u8P8GvPBzawhNliJz1+GgDUf3IG0Cc  
oWkGdm60u7gD262vyRCGWrPgu6/At+tr8Wpn  
---END CERTIFICATE---
```

ServerName

domain.example.com

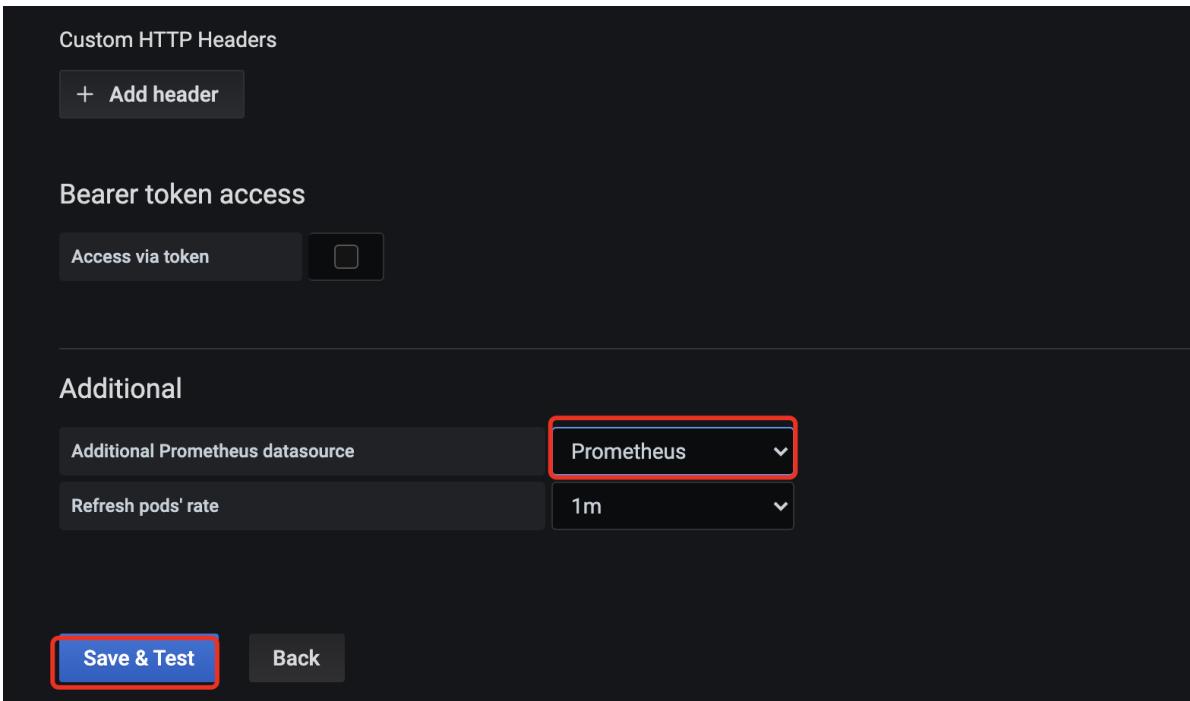
Client Cert

```
BQADggEBAM17uZhZeYviPhOhhl+KgY+MgEqQ0/HRIjolli50WKSvHeQjdJxanZy  
K2hJvC5u92mV6PtF59lijuLewaqTTJUTpB4PJK3oR8S6ewL6y0lTfRqgjLXd77P  
jdYw1lqqo4He1ZKJvGRAG2ma9/0fUhUdODM4Zl/p4exR+uQ4QkJXaeCub+sxeJ5Q  
w9mHerVlgRmRVn5lPcb8gmv6FIAJAZs1ynQHY1ajHM6hhA2vHliARyal7XAvAfqP  
E96x2PnuuUNSihzSoqudu5lq8Jg8Rc+h+crMRlwNJDtsdU89WiWSovU4Kc+Y/5Bn  
nwobERznrg55yAwRf23y1KnC61BkyeE  
---END CERTIFICATE---
```

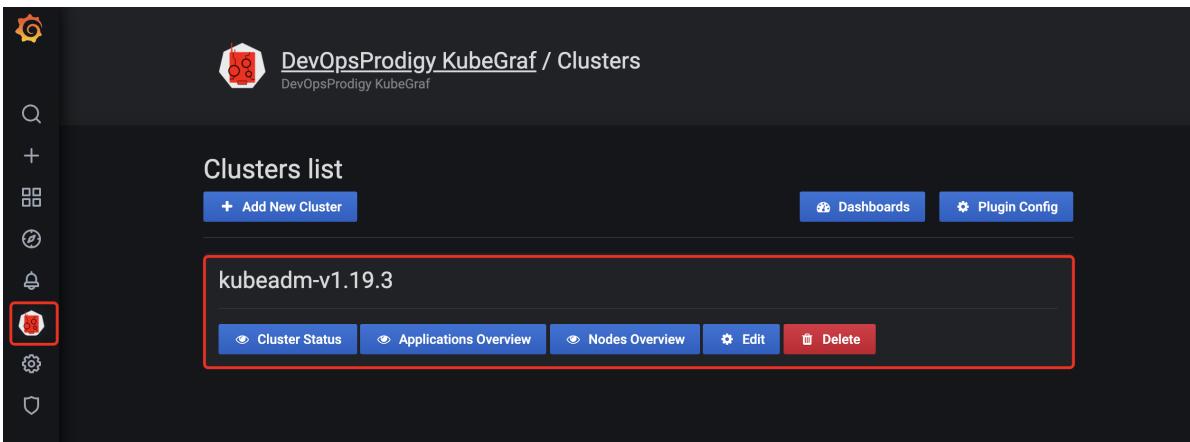
Client Key

```
--BEGIN RSA PRIVATE KEY--
```

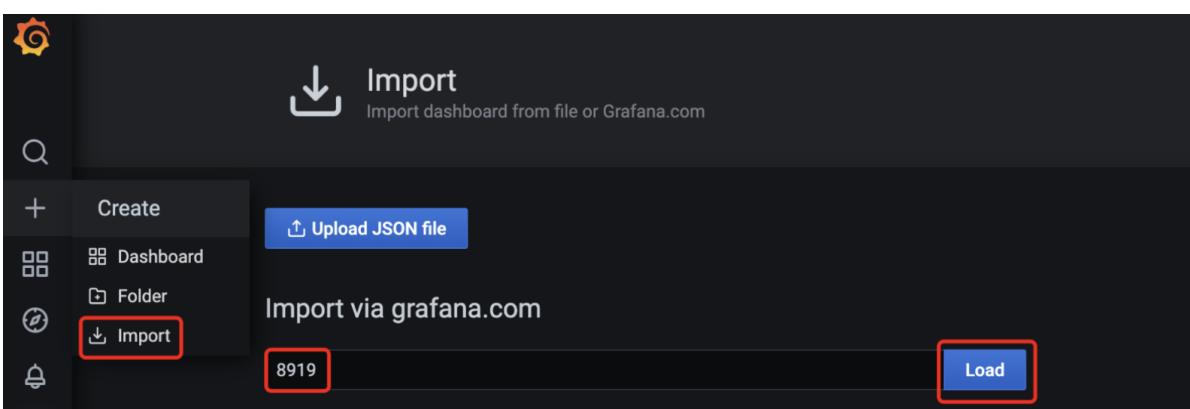
```
MIIEcwIBAAKCAQEAwYLU5aR+jovDRSZiuLDfazzTj4M3DkhE7roxdiqql/VwiqRB  
dhByn/rB0JE8BxoVqVX/DPxGPCV9IJFK8iCt0x3kz1fc1DU5yP10zUle5n2lrFXL  
X/H2iaR4rMp+40FRc0TdpC6HvCAbvtqAdQ1ah0HeiyK+NcTmBBAoJ+VAu5PEoE9L  
7U5g3qJ016ZpQaz4C3LEfUwxK5dQfzL5zYld0ZULpk3tDxjGwrfdKWu14+q1L7Z  
sGAhyrlxDOnK0wusyXowiMLTCd7ZyqvkSGv+/OA6h5yquB2drJZ7d8TOvcNK9Ys  
za4HMQy0INK9H+HRlv9z7uv5ezarMae1AQlVTQIDAQABAoIBABJoOYwmms4s/rDF1
```



保存之后左边就会出现插件的图标，点击就可以查看了



导入dashboard



Importing Dashboard from Grafana.com

Published by

StarsL.cn

Updated on

2021-01-30 03:15:45

Options

Name _____

1 Node Exporter for Prometheus Dashboard CN v20201010

Folder

General

Unique identifier (uid)

The unique identifier (uid) of a dashboard can be used for uniquely identify a dashboard between multiple Grafana installs. The uid allows having consistent URLs for accessing dashboards so changing the title of a dashboard will not break any bookmarked links to that dashboard.

9CWBz0bjk

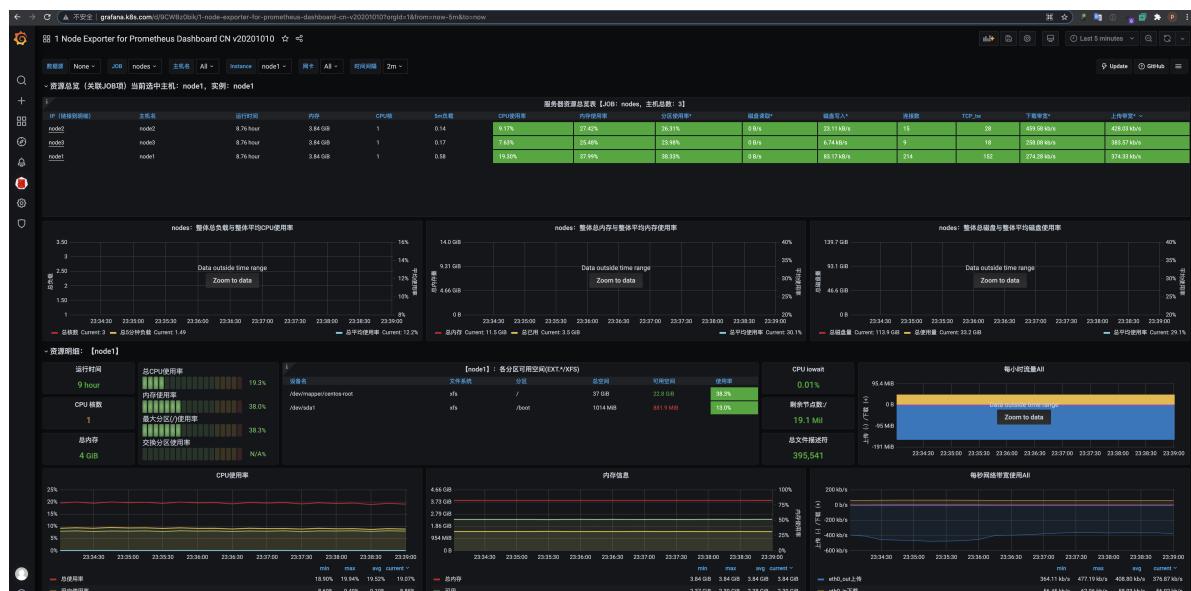
Change uid

VictoriaMetrics

 Prometheus

Import

Cancel



第x章 Alertmanager报警

安装

```
1 cat > alertmanager.yml << 'EOF'
2
3 apiVersion: v1
4 kind: ConfigMap
5 metadata:
6   name: alert-config
7   namespace: prom
8
9 data:
```

```
8 config.yml: |-
9   global:
10     # 当alertmanager持续多长时间未接收到告警后标记告警状态为 resolved
11     resolve_timeout: 5m
12     # 配置邮件发送信息
13     smtp_smarthost: 'smtp.qq.com:465'
14     smtp_from: '526195417@qq.com'
15     smtp_auth_username: '526195417@qq.com'
16     smtp_auth_password: 'mtbxhvqaonvrbgia'
17     smtp_hello: 'qq.com'
18     smtp_require_tls: false
19     # 所有报警信息进入后的根路由，用来设置报警的分发策略
20     route:
21       # 这里的标签列表是接收到报警信息后的重新分组标签，例如，接收到的报警信息里面有许多具有 cluster=A 和 alertname=LatencyHigh 这样的标签的报警信息将会批量被聚合到一个分组里面
22       group_by: ['alertname', 'cluster']
23       # 当一个新的报警分组被创建后，需要等待至少 group_wait 时间来初始化通知，这种方式可以确保您能有足够的时间为同一分组来获取多个警报，然后一起触发这个报警信息。
24       group_wait: 30s
25
26       # 相同的group之间发送告警通知的时间间隔
27       group_interval: 30s
28
29       # 如果一个报警信息已经发送成功了，等待 repeat_interval 时间来重新发送他们，不同类型告警发送频率需要具体配置
30       repeat_interval: 1h
31
32       # 默认的receiver: 如果一个报警没有被一个route匹配，则发送给默认的接收器
33       receiver: default
34
35       # 上面所有的属性都由所有子路由继承，并且可以在每个子路由上进行覆盖。
36       routes:
37         - receiver: email
38           group_wait: 10s
39           match:
40             team: node
41       receivers:
42         - name: 'default'
43           email_configs:
44             - to: '526195417@qq.com'
45             send_resolved: true # 接受告警恢复的通知
46         - name: 'email'
47           email_configs:
48             - to: '526195417@qq.com'
49             send_resolved: true
50
51   apiversion: apps/v1
52   kind: Deployment
53   metadata:
54     name: alertmanager
55     namespace: prom
56     labels:
57       app: alertmanager
58   spec:
59     selector:
60       matchLabels:
61         app: alertmanager
```

```
62 template:
63   metadata:
64     labels:
65       app: alertmanager
66   spec:
67     volumes:
68       - name: alertcfg
69         configMap:
70           name: alert-config
71     containers:
72       - name: alertmanager
73         image: prom/alertmanager:v0.21.0
74         imagePullPolicy: IfNotPresent
75         args:
76           - "--config.file=/etc/alertmanager/config.yml"
77         ports:
78           - containerPort: 9093
79             name: http
80         volumeMounts:
81           - mountPath: "/etc/alertmanager"
82             name: alertcfg
83         resources:
84           requests:
85             cpu: 100m
86             memory: 256Mi
87           limits:
88             cpu: 100m
89             memory: 256Mi
90 ---
91 apiVersion: v1
92 kind: Service
93 metadata:
94   name: alertmanager
95   namespace: prom
96   labels:
97     app: alertmanager
98 spec:
99   selector:
100     app: alertmanager
101   ports:
102     - name: web
103       port: 9093
104       targetPort: http
105 ---
106 apiVersion: networking.k8s.io/v1
107 kind: Ingress
108 metadata:
109   name: alertmanager
110   namespace: prom
111   labels:
112     app: alertmanager
113 spec:
114   rules:
115     - host: alertmanager.k8s.com
116       http:
117         paths:
118           - path: /
119             pathType: ImplementationSpecific
```

```
120     backend:  
121         service:  
122             name: alertmanager  
123             port:  
124                 number: 9093  
125 EOF
```

应用配置：

```
1 [root@node1 prom]# kubectl apply -f alertmanager.yaml  
2 configmap/alert-config created  
3 deployment.apps/alertmanager created  
4 service/alertmanager created  
5 ingress.networking.k8s.io/alertmanager created
```

访问查看

The screenshot shows the Alertmanager interface. At the top, there are tabs for Alertmanager, Alerts, Silences, Status, and Help, with 'Alerts' selected. A 'New Silence' button is in the top right. Below the tabs, there are two buttons: 'Filter' and 'Group'. To the right of these are buttons for 'Receiver: All', 'Silenced', and 'Inhibited'. There is a search bar with placeholder text 'Custom matcher, e.g. env="production"'. A blue '+' button is next to a 'Silence' icon. At the bottom left, there is a link to 'Expand all groups'. A yellow banner at the bottom states 'No alert groups found'.

配置报警规则

编写报警规则

```
1 cat > alert-rules.yaml << 'EOF'  
2 apiVersion: v1  
3 kind: ConfigMap  
4 metadata:  
5     name: prometheus-config  
6     namespace: kube-mon  
7 data:  
8     prometheus.yaml: |  
9         global:  
10            scrape_interval: 15s  
11            scrape_timeout: 15s  
12            evaluation_interval: 30s # 默认情况下每分钟对告警规则进行计算  
13         alerting:  
14             alertmanagers:  
15                 - static_configs:  
16                     - targets: ["alertmanager:9093"]  
17             rule_files:  
18                 - /etc/prometheus/rules.yaml  
19                 ..... # 省略prometheus其他部分  
20         rules.yaml: |  
21             groups:  
22                 - name: test-node-mem
```

```
23     rules:
24       - alert: NodeMemoryUsage
25         expr: (node_memory_MemTotal_bytes - (node_memory_MemFree_bytes +
26           node_memory_Buffers_bytes + node_memory_Cached_bytes)) /
27             node_memory_MemTotal_bytes * 100 > 20
28           for: 2m
29           labels:
30             team: node
31           annotations:
32             summary: "{{$labels.instance}}: High Memory usage detected"
33             description: "{{$labels.instance}}: Memory usage is above 20%
34               (current value is: {{ $value }}}"
35           EOF
```