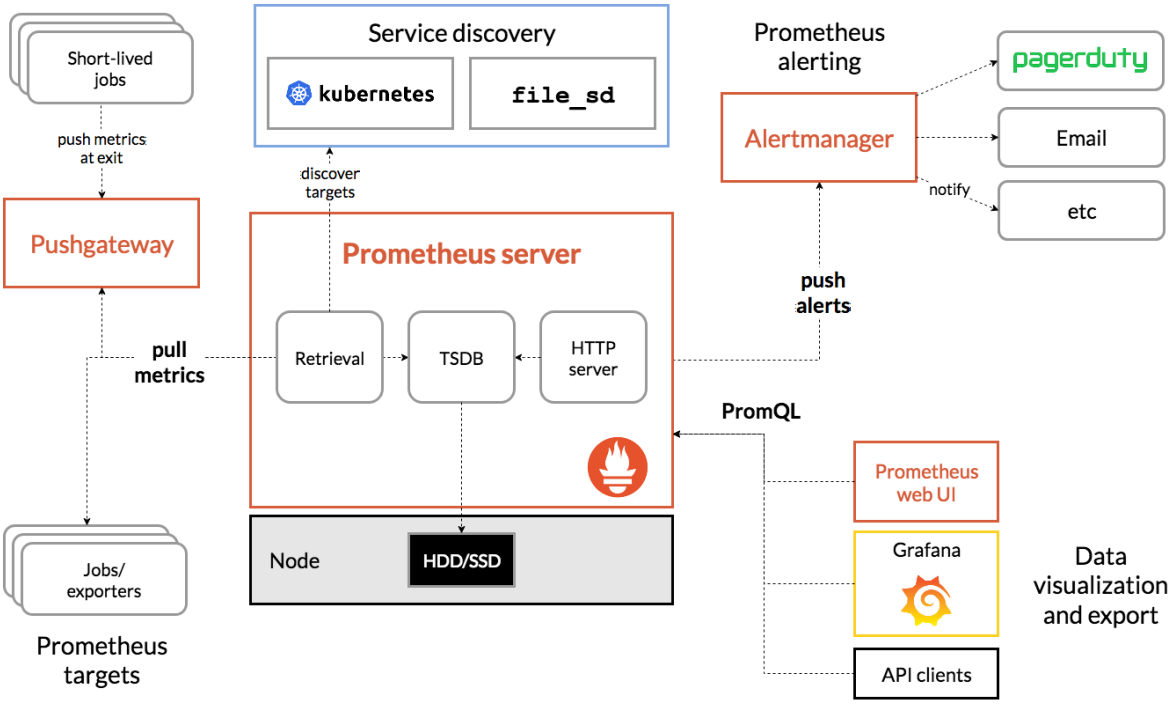
第x章 Prometheus介绍

官方地址

```
https://prometheus.io/docs/introduction/overview/
```

组件架构

```
Prometheus Server    服务端，主动拉数据，存入TSDB数据库
TSDB                 时序数据库，用于存储拉取来的监控数据
exporter             暴露指标的组件，需要独立安装
Pushgatway           push的方式将指标数据推送到网关
Alertmanager         报警组件
Promtheus 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

```
https://prometheus.io/docs/prometheus/latest/configuration/configuration/
```

编写configmap

```
cat > prom-cm.yml << 'EOF'
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: prom
```

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

创建PV和PVC

```
cat > prom-pv-pvc.yml << 'EOF'
apiVersion: v1
kind: PersistentVolume
metadata:
  name: prometheus-local
  labels:
    app: prometheus
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 10Gi
  storageClassName: local-storage
  local:
    path: /data/k8s/prometheus
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - node2
  persistentVolumeReclaimPolicy: Retain
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: prometheus-data
  namespace: prom
spec:
  selector:
    matchLabels:
      app: prometheus
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: local-storage
EOF
```

编写RBAC

```
cat > prom-rbac.yml << 'EOF'
apiVersion: v1
kind: ServiceAccount
metadata:
  name: prometheus
  namespace: prom
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus
rules:
- apiGroups:
  - ""
  resources:
  - nodes
  - services
  - endpoints
  - pods
  - nodes/proxy
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - "extensions"
  resources:
  - ingresses
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - configmaps
  - nodes/metrics
  verbs:
  - get
- nonResourceURLs:
  - /metrics
  verbs:
  - get
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: prometheus
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus
subjects:
- kind: ServiceAccount
  name: prometheus
  namespace: prom
EOF
```

编写deployment

```
cat > prom-dp.yml << 'EOF'
apiVersion: apps/v1
kind: Deployment
metadata:
  name: prometheus
  namespace: prom
  labels:
    app: prometheus
spec:
  selector:
    matchLabels:
      app: prometheus
  template:
    metadata:
      labels:
        app: prometheus
    spec:
      serviceAccountName: prometheus        #引用RBAC创建的ServiceAccount
      volumes:
      - name: data
        persistentVolumeClaim:
          claimName: prometheus-data
      - name: config-volume
        configMap:
          name: prometheus-config
      initContainers:                       #由初始化容器将数据目录的属性修改为nobody用户和组
      - name: fix-permissions
        image: busybox
        command: [chown, -R, "nobody:nobody", /prometheus]
        volumeMounts:
        - name: data
          mountPath: /prometheus
      containers:
      - name: prometheus
        image: prom/prometheus:v2.24.1
        args:
        - "--config.file=/etc/prometheus/prometheus.yml"    #指定配置文件
        - "--storage.tsdb.path=/prometheus"      #tsdb数据库保存路径
        - "--storage.tsdb.retention.time=24h"    #数据保留时间，默认15天
        - "--web.enable-admin-api"  #控制对admin HTTP API的访问
        - "--web.enable-lifecycle"  #支持热更新，直接执行localhost:9090/-/reload立即生效
        ports:
        - name: http
          containerPort: 9090
        volumeMounts:
        - name: config-volume
          mountPath: "/etc/prometheus"
        - name: data
          mountPath: "/prometheus"
        resources:
          requests:
            cpu: 100m
            memory: 512Mi
```

```
          limits:
            cpu: 100m
            memory: 512Mi
EOF
```

编写Service

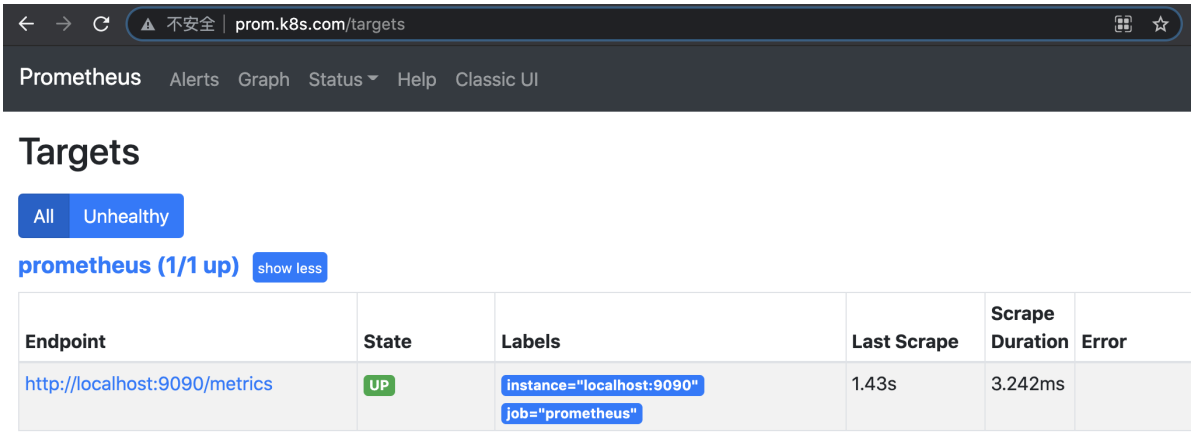
```
cat > prom-svc.yml << 'EOF'
apiVersion: v1
kind: Service
metadata:
  name: prometheus
  namespace: prom
  labels:
    app: prometheus
spec:
  selector:
    app: prometheus
  ports:
    - name: web
      port: 9090
      targetPort: http
EOF
```

编写ingress

```
cat > prom-ingress.yml << 'EOF'
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: prometheus
  namespace: prom
  labels:
    app: prometheus
spec:
  rules:
  - host: prom.k8s.com
    http:
      paths:
      - path: /
        pathType: ImplementationSpecific
        backend:
          service:
            name: prometheus
            port:
              number: 9090
EOF
```

访问promtheus

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



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croedns的配置文件可以发现提供prometheus服务采集的端口是9153。

```
[root@node1 prom]# kubectl -n kube-system describe cm coredns
Name:         coredns
Namespace:    kube-system
Labels:       <none>
Annotations:  <none>

Data
====
Corefile:
----
.:53 {
    errors
    health {
       lameduck 5s
    }
    ready
    kubernetes cluster.local in-addr.arpa ip6.arpa {
       pods insecure
       fallthrough in-addr.arpa ip6.arpa
       ttl 30
    }
    prometheus :9153        #自带的prometheus监控暴露服务
    forward . /etc/resolv.conf {
       max_concurrent 1000
    }
    cache 30
    loop
```

```
    reload
    loadbalance
}

Events:  <none>
```

查看CoreDNS的Pod地址

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

直接访问/metrics接口

```
[root@node1 prom]# curl -I 10.2.0.17:9153/metrics
HTTP/1.1 200 OK
Content-Type: text/plain; version=0.0.4; charset=utf-8
Date: Wed, 11 Aug 2021 07:14:25 GMT

[root@node1 prom]# curl 10.2.0.17:9153/metrics
# HELP coredns_build_info A metric with a constant '1' value labeled by
version, revision, and goversion from which CoreDNS was built.
# TYPE coredns_build_info gauge
coredns_build_info{goversion="go1.14.4",revision="f59c03d",version="1.7.0"} 1
# HELP coredns_cache_entries The number of elements in the cache.
# TYPE coredns_cache_entries gauge
coredns_cache_entries{server="dns://:53",type="denial"} 12
coredns_cache_entries{server="dns://:53",type="success"} 1
# HELP coredns_cache_misses_total The count of cache misses.
# TYPE coredns_cache_misses_total counter
coredns_cache_misses_total{server="dns://:53"} 13
# HELP coredns_dns_request_duration_seconds Histogram of the time (in
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

```
[root@node1 prom]# cat prom-cm.yml
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: prom
data:
  prometheus.yml: |
    global:
      scrape_interval: 15s
      scrape_timeout: 15s
```

```
    scrape_configs:
    - job_name: 'prometheus'
      static_configs:
      - targets: ['localhost:9090']

    - job_name: 'coredns'       #任务名称
      static_configs:               #静态配置
      - 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

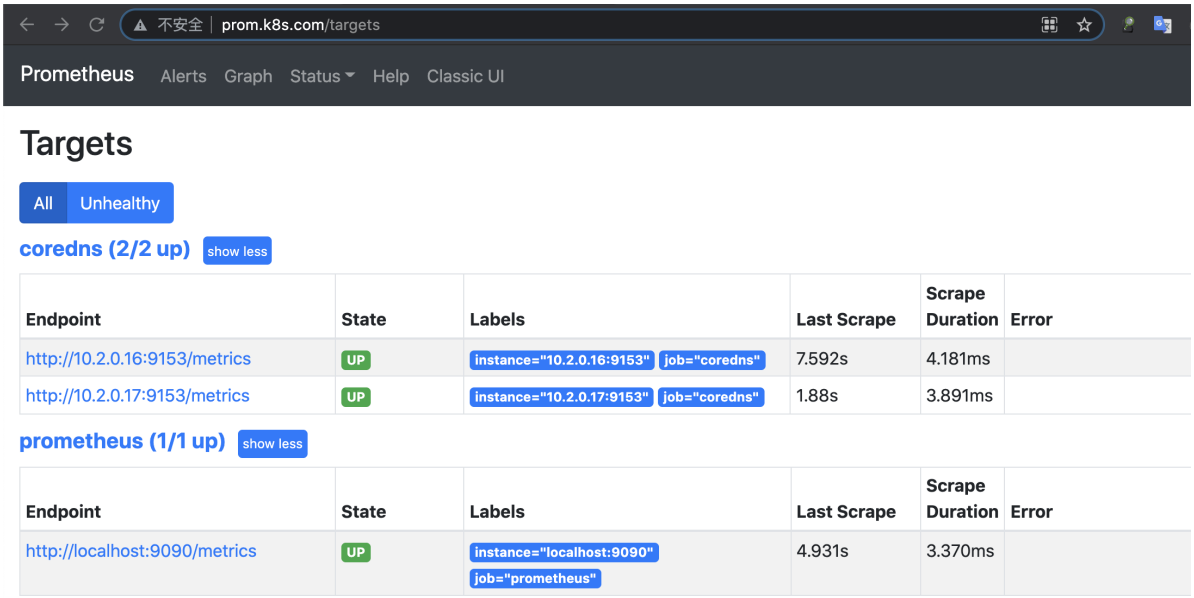
```
[root@node1 prom]# kubectl apply -f prom-cm.yml
```

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

```
[root@node1 prom]# kubectl -n prom get pod -o wide
NAME                          READY   STATUS    RESTARTS   AGE   IP
NODE    NOMINATED NODE   READINESS GATES
prometheus-796566c67c-lhrns   1/1     Running   0          22m   10.2.2.86
node2   <none>           <none>

#注意要等一会，因为configmap更新到pod里需要点时间
[root@node1 prom]# curl -X POST "http://10.2.2.86:9090/-/reload"
```

查看promtheus发现



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：

```
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。

```
cat >mysql-prom.yaml <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-dp
  namespace: prom
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - name: mysql
        image: mysql:5.7
        ports:
        - containerPort: 3306
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: "123456"
      - name: mysql-exporter
        image: prom/mysqld-exporter
        ports:
        - containerPort: 9104
        env:
        - name: DATA_SOURCE_NAME
          value: "root:123456@(localhost:3306)/"
---
kind: Service
apiVersion: v1
metadata:
  name: mysql-svc
  namespace: prom
spec:
  selector:
    app: mysql
  ports:
  - name: mysql
    port: 3306
    targetPort: 3306
  - name: mysql-prom
    port: 9104
    targetPort: 9104
EOF
```

应用后查看

```
[root@node1 prom]# kubectl apply -f mysql-prom.yaml
deployment.apps/mysql-dp created
service/mysql-svc created

[root@node1 prom]# kubectl -n prom get pod
NAME                          READY   STATUS    RESTARTS   AGE
mysql-dp-79b48cff96-m96bz     2/2     Running   0          73s
prometheus-796566c67c-lhrns   1/1     Running   0          46m

[root@node1 prom]# kubectl -n prom get svc
NAME         TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)             AGE
mysql-svc    ClusterIP   10.1.213.31   <none>        3306/TCP,9104/TCP   4m9s
prometheus   ClusterIP   10.1.95.152   <none>        9090/TCP            44m
```

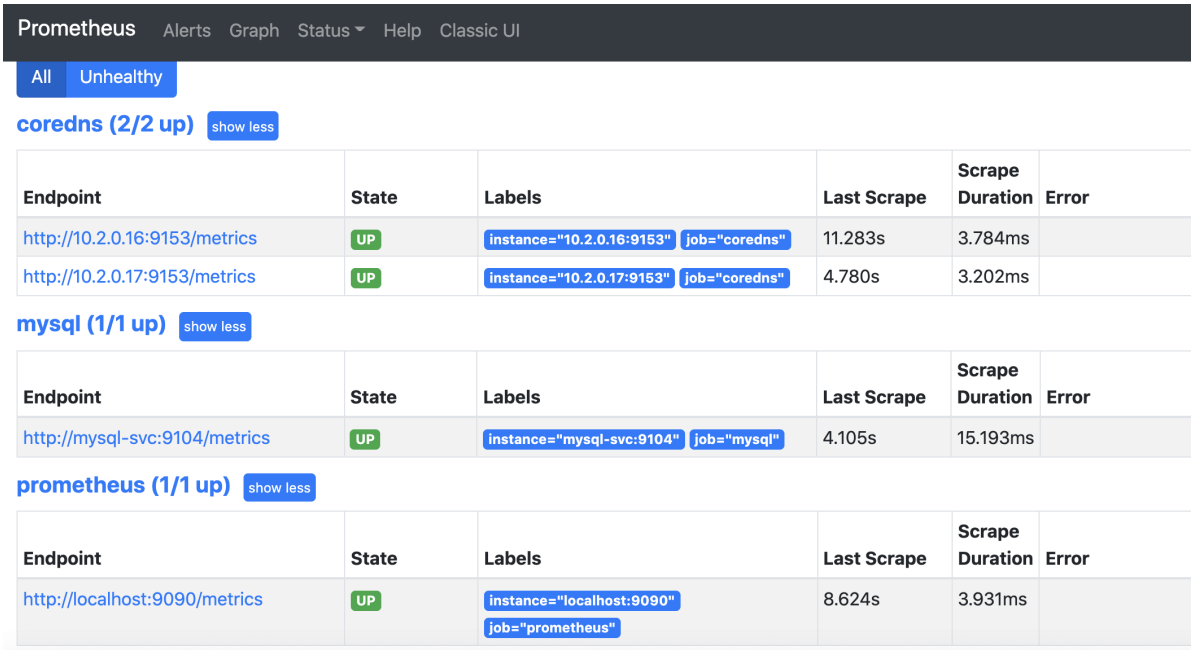
修改prom配置文件

```
cat > prom-cm.yml << 'EOF'
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: prom
data:
  prometheus.yml: |
    global:
      scrape_interval: 15s
      scrape_timeout: 15s
    scrape_configs:
    - job_name: 'prometheus'
      static_configs:
      - targets: ['localhost:9090']

    - job_name: 'coredns'
      static_configs:
      - targets: ['10.2.0.16:9153','10.2.0.17:9153']

    - job_name: 'mysql'
      static_configs:
      - targets: ['mysql-svc:9104']
EOF
```

更新配置

```
[root@node1 prom]# kubectl apply -f prom-cm.yml
configmap/prometheus-config configured

[root@node1 prom]# curl -X POST "http://10.2.2.86:9090/-/reload"
```

查看promtheus



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：

```
k8s节点的运行状态：内存，cpu，磁盘，网络等。

k8s组件的运行状态：kube-apiserver,kube-controller,kube-scheduler,coredns等。

k8s资源的运行状态：Deployment,Daemoset,Service,Ingress等。
```

对于k8s集群的监控，主要有以下几种方案：

```
cAdvisor:   监控容器内部的资源使用情况，已经集成在kubelet里了
kube-state-metrics：监控k8s资源的使用状态
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：

```
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

```
cat > prom-node-exporter.yaml << 'EOF'
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: node-exporter
```

```
  namespace: prom
  labels:
    app: node-exporter
spec:
  selector:
    matchLabels:
      app: node-exporter
  template:
    metadata:
      labels:
        app: node-exporter
    spec:
      hostPID: true             #使用主机的 pid 命名空间
      hostIPC: true             #使用主机的 ipc 命名空间
      hostNetwork: true     #用主机的网络命名空间
      nodeSelector:
        kubernetes.io/os: linux
      containers:
      - name: node-exporter
        image: prom/node-exporter:v1.1.1
        args:
        - --web.listen-address=$(HOSTIP):9100       #监控指标数据的地址端口号
        - --path.procfs=/host/proc                          #监控proc路径
        - --path.sysfs=/host/sys                              #监控sys路径
        - --path.rootfs=/host/root                          #监控/rootfs路径
        - --collector.filesystem.ignored-mount-
points=^/(dev|proc|sys|var/lib/docker/.+)($|/)       #忽略监控的磁盘信息
        - --collector.filesystem.ignored-fs-
types=^(autofs|binfmt_misc|cgroup|configfs|debugfs|devpts|devtmpfs|fusectl|hu
getlbfs|mqueue|overlay|proc|procfs|pstore|rpc_pipefs|securityfs|sysfs|tracefs
)$    #忽略监控的文件系统
        ports:
        - containerPort: 9100
        env:
        - name: HOSTIP
          valueFrom:
            fieldRef:
              fieldPath: status.hostIP          #读取node本身的IP地址
        resources:
          requests:
            cpu: 150m
            memory: 180Mi
          limits:
            cpu: 150m
            memory: 180Mi
        securityContext:                                    #全上下文，用于定义
Container的权限和访问控制
          runAsNonRoot: true                            #容器不以root运行
          runAsUser: 65534                              #使用指定的uid用户运行
        volumeMounts:
        - name: proc
          mountPath: /host/proc
        - name: sys
          mountPath: /host/sys
        - name: root
          mountPath: /host/root
          mountPropagation: HostToContainer  #mountPropagation确定挂载如何从主
机传播到容器
```

```
          readOnly: true
      tolerations:                                #容忍
      - operator: "Exists"              #Exists相当于值的通配符，因此pod可以容忍特定类别的所有污点。
      volumes:
      - name: proc
        hostPath:
          path: /proc
      - name: dev
        hostPath:
          path: /dev
      - name: sys
        hostPath:
          path: /sys
      - name: root
        hostPath:
          path: /
EOF
```

配置文件解释:

```
#获取pod信息
https://kubernetes.io/docs/tasks/inject-data-application/environment-variable-expose-pod-information/

env:
- name: HOSTIP
  valueFrom:
    fieldRef:
      fieldPath: status.hostIP  #读取node本身的IP地址
--------------------------------
securityContext:                    #全上下文，用于定义Container的权限和访问控制
  runAsNonRoot: true            #容器不以root运行
  runAsUser: 65534              #使用指定的uid用户运行
--------------------------------
tolerations:                                #容忍
- operator: "Exists"            #Exists相当于值的通配符，因此pod可以容忍特定类别的所有污点。
--------------------------------
securityContext:                    #全上下文，用于定义Container的权限和访问控制
  runAsNonRoot: true            #容器不以root运行
  runAsUser: 65534              #使用指定的uid用户运行
--------------------------------
hostPID: true                               #使用主机的 pid 命名空间
hostIPC: true                               #使用主机的 ipc 命名空间
hostNetwork: true                   #用主机的网络命名空间
```

应用资源配置

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

查看pod创建状态

```
[root@node1 prom]# kubectl -n prom get pod
NAME                            READY   STATUS    RESTARTS   AGE
mysql-dp-79b48cff96-m96bz       2/2     Running   0          15m
node-exporter-7zhjf             1/1     Running   0          41s
node-exporter-kzqvz             1/1     Running   0          41s
node-exporter-zbsrl             1/1     Running   0          41s
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：

```
https://prometheus.io/docs/prometheus/latest/configuration/configuration/#kubernetes_sd_config
```

node节点自动发现

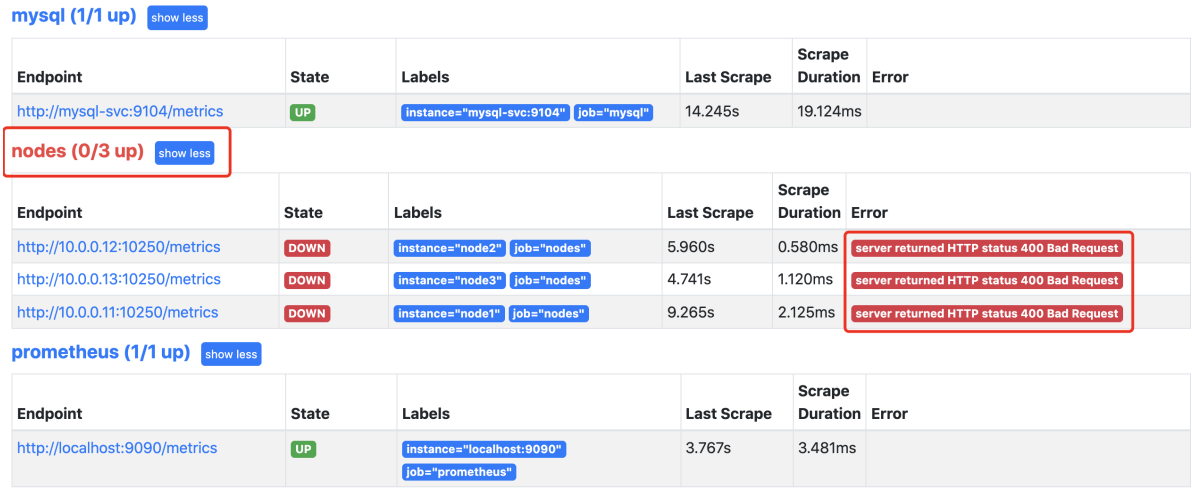
修改prometheus配置

```
cat > prom-cm.yml << 'EOF'
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: prom
data:
  prometheus.yml: |
    global:
      scrape_interval: 15s
      scrape_timeout: 15s
    scrape_configs:
    - job_name: 'prometheus'
      static_configs:
      - targets: ['localhost:9090']

    - job_name: 'coredns'
      static_configs:
      - targets: ['10.2.0.16:9153','10.2.0.17:9153']

    - job_name: 'mysql'
      static_configs:
      - targets: ['mysql-svc:9104']

    - job_name: 'nodes'
      kubernetes_sd_configs:        #k8s自动服务发现
      - role: node                  #自动发现类型为node
EOF
```

生效配置

```
[root@node1 prom]# kubectl apply -f prom-cm.yml
configmap/prometheus-config configured

[root@node1 prom]# curl -X POST "http://10.2.2.86:9090/-/reload"
```

查看prometheus



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：

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

修改prometheus配置：

```
cat > prom-cm.yml << 'EOF'
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: prom
data:
  prometheus.yml: |
    global:
      scrape_interval: 15s
      scrape_timeout: 15s
    scrape_configs:
    - job_name: 'prometheus'
      static_configs:
      - targets: ['localhost:9090']

    - job_name: 'coredns'
      static_configs:
      - targets: ['10.2.0.16:9153','10.2.0.17:9153']

    - job_name: 'mysql'
```

```
      static_configs:
      - targets: ['mysql-svc:9104']

    - job_name: 'nodes'
      kubernetes_sd_configs:
      - role: node
      relabel_configs:                                          #配置重写
      - action: replace                                         #基于正则表达式匹配执行的操作
        source_labels: ['__address__']      #从源标签里选择值
        regex: '(.*):10250'                                     #提取的值与之匹配的正则表达式
        replacement: '${1}:9100'                        #执行正则表达式替换的值
        target_label: __address__                       #结果值在替换操作中写入的标签
EOF
```

生效配置：

```
[root@node1 prom]# kubectl apply -f prom-cm.yml
configmap/prometheus-config configured

等待一会再更新
[root@node1 prom]# curl -X POST "http://10.2.2.86:9090/-/reload"
```

prometheus查看：

mysql (1/1 up) show less

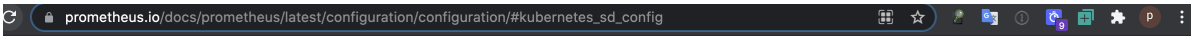
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://mysql-svc:9104/metrics	UP	instance="mysql-svc:9104" job="mysql"	5.366s	15.673ms	

nodes (3/3 up) show less

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：

```
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:

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

修改配置:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: prom
data:
  prometheus.yml: |
    global:
      scrape_interval: 15s
      scrape_timeout: 15s
    scrape_configs:
    - job_name: 'prometheus'
      static_configs:
      - targets: ['localhost:9090']

    - job_name: 'coredns'
      static_configs:
      - targets: ['10.2.0.16:9153','10.2.0.17:9153']

    - job_name: 'mysql'
      static_configs:
      - targets: ['mysql-svc:9104']

    - job_name: 'nodes'
      kubernetes_sd_configs:
      - role: node
      relabel_configs:
      - action: replace
        source_labels: ['__address__']
        regex: '(.*):10250'
```

```
        replacement: '${1}:9100'
        target_label: __address__

      - action: labelmap        #将正则表达式与所有标签名称匹配
        regex: __meta_kubernetes_node_label_(.+)     #提取符合正则匹配的标签，然后天交到Label里
```

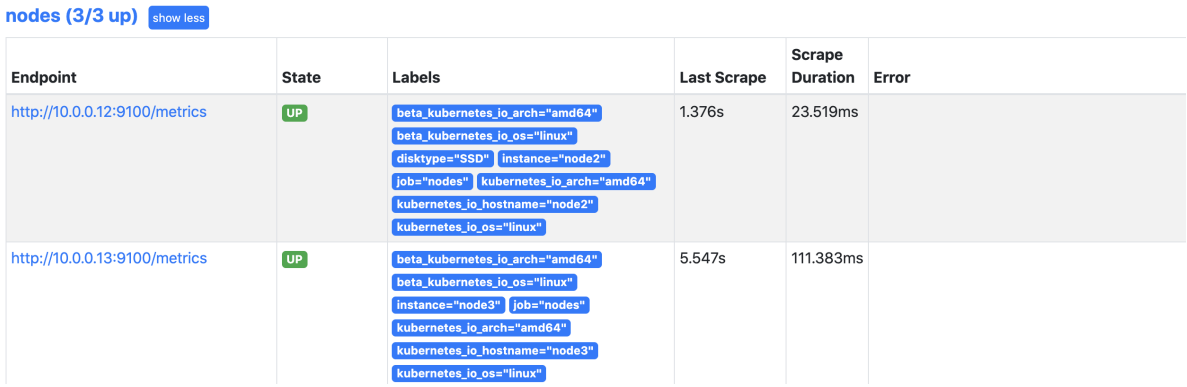
配置解释：

```
官方说明：
https://prometheus.io/docs/prometheus/latest/configuration/configuration/#relabel_config

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Lable标签里。
```

生效配置：

```
[root@node1 prom]# kubectl apply -f prom-cm.yml
configmap/prometheus-config configured

等一会再更新
[root@node1 prom]# curl -X POST "http://10.2.2.86:9090/-/reload"
```

查看prometheus



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：

```
cat > prom-cm.yml << 'EOF'
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: prom
data:
  prometheus.yml: |
```

```
    global:
      scrape_interval: 15s
      scrape_timeout: 15s
    scrape_configs:
    - job_name: 'prometheus'
      static_configs:
      - targets: ['localhost:9090']

    - job_name: 'coredns'
      static_configs:
      - targets: ['10.2.0.16:9153','10.2.0.17:9153']

    - job_name: 'mysql'
      static_configs:
      - targets: ['mysql-svc:9104']

    - job_name: 'nodes'
      kubernetes_sd_configs:
      - role: node
      relabel_configs:
      - source_labels: ['__address__']
        regex: '(.*):10250'
        replacement: '${1}:9100'
        target_label: __address__
        action: replace
      - action: labelmap
        regex: __meta_kubernetes_node_label_(.+)

    - job_name: 'kubelet'
      kubernetes_sd_configs:
      - role: node
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - action: labelmap
        regex: __meta_kubernetes_node_label_(.+)
EOF
```

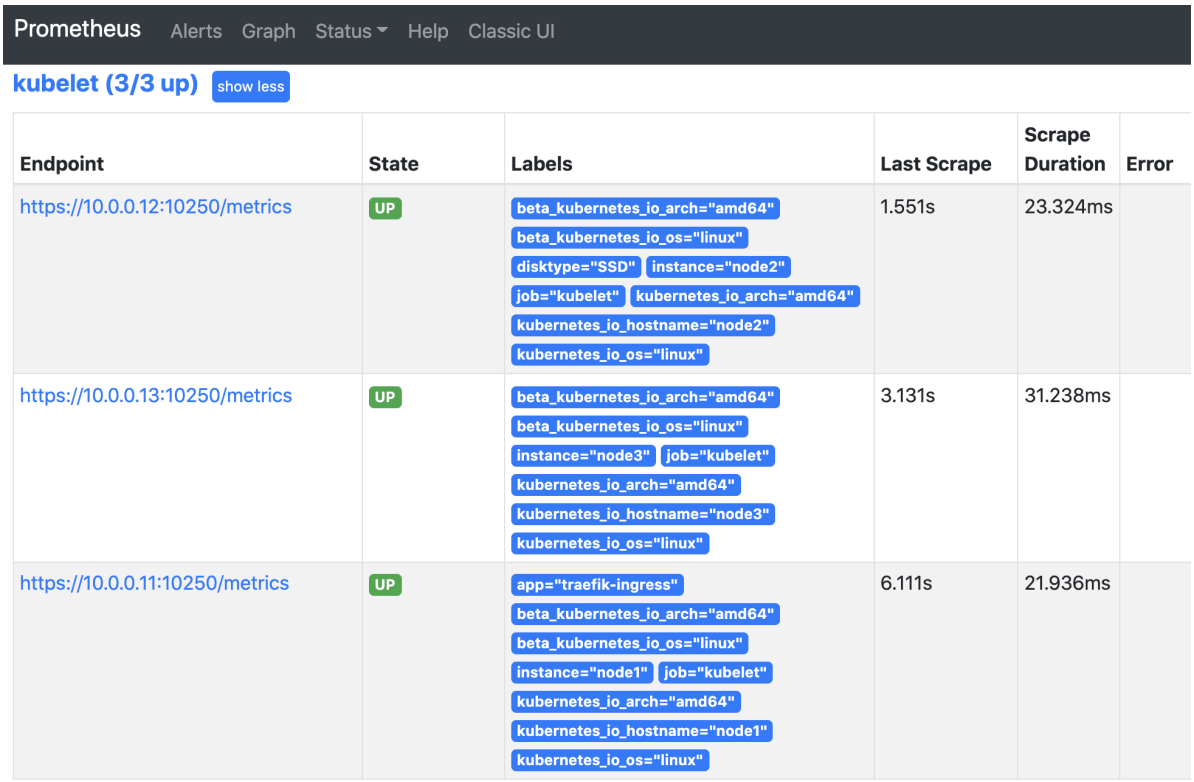
配置解释:

```
- job_name: 'kubelet'
  kubernetes_sd_configs:
  - role: node
  scheme: https     #配置用于请求的协议
  tls_config:           #tls配置
    ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt       #ca证书
    insecure_skip_verify: true      #禁用证书检查
  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
  relabel_configs:
  - action: labelmap
    regex: __meta_kubernetes_node_label_(.+)
```

应用配置：

```
[root@node1 prom]# kubectl apply -f prom-cm.yml
configmap/prometheus-config configured

稍等一会配置
[root@node1 prom]# curl -X POST "http://10.2.2.86:9090/-/reload"
```

查看效果：



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：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: prom
data:
  prometheus.yml: |
    global:
      scrape_interval: 15s
      scrape_timeout: 15s
    scrape_configs:
    - job_name: 'prometheus'
      static_configs:
      - targets: ['localhost:9090']

    - job_name: 'coredns'
```

```
      static_configs:
      - targets: ['10.2.0.16:9153','10.2.0.17:9153']

    - job_name: 'mysql'
      static_configs:
      - targets: ['mysql-svc:9104']

    - job_name: 'nodes'
      kubernetes_sd_configs:
      - role: node
      relabel_configs:
      - source_labels: ['__address__']
        regex: '(.*):10250'
        replacement: '${1}:9100'
        target_label: __address__
        action: replace
      - action: labelmap
        regex: __meta_kubernetes_node_label_(.+)

    - job_name: 'kubelet'
      kubernetes_sd_configs:
      - role: node
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - action: labelmap
        regex: __meta_kubernetes_node_label_(.+)

    - job_name: 'cadvisor'
      kubernetes_sd_configs:
      - role: node
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - action: labelmap
        regex: __meta_kubernetes_node_label_(.+)
        replacement: $1
      - source_labels: [__meta_kubernetes_node_name]
        regex: (.*)
        replacement: /metrics/cadvisor
        target_label: __metrics_path__
```

应用配置:

```
[root@node1 prom]# kubectl apply -f prom-cm.yml
configmap/prometheus-config configured

[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	UP	app="traefik-ingress" beta_kubernetes_io_arch="amd64" beta_kubernetes_io_os="linux" instance="node1" job="cadvisor" kubernetes_io_arch="amd64" kubernetes_io_hostname="node1" kubernetes_io_os="linux"	24.830s	88.742ms
https://10.0.0.12:10250/metrics/cadvisor	UP	beta_kubernetes_io_arch="amd64" beta_kubernetes_io_os="linux" disktype="SSD" instance="node2" job="cadvisor" kubernetes_io_arch="amd64" kubernetes_io_hostname="node2" kubernetes_io_os="linux"	23.430s	48.214ms
https://10.0.0.13:10250/metrics/cadvisor	UP	beta_kubernetes_io_arch="amd64" beta_kubernetes_io_os="linux" instance="node3" job="cadvisor" kubernetes_io_arch="amd64" kubernetes_io_hostname="node3" kubernetes_io_os="linux"	21.402s	36.157ms

关于查询的数据可以查看官网的说明：

```
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：

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

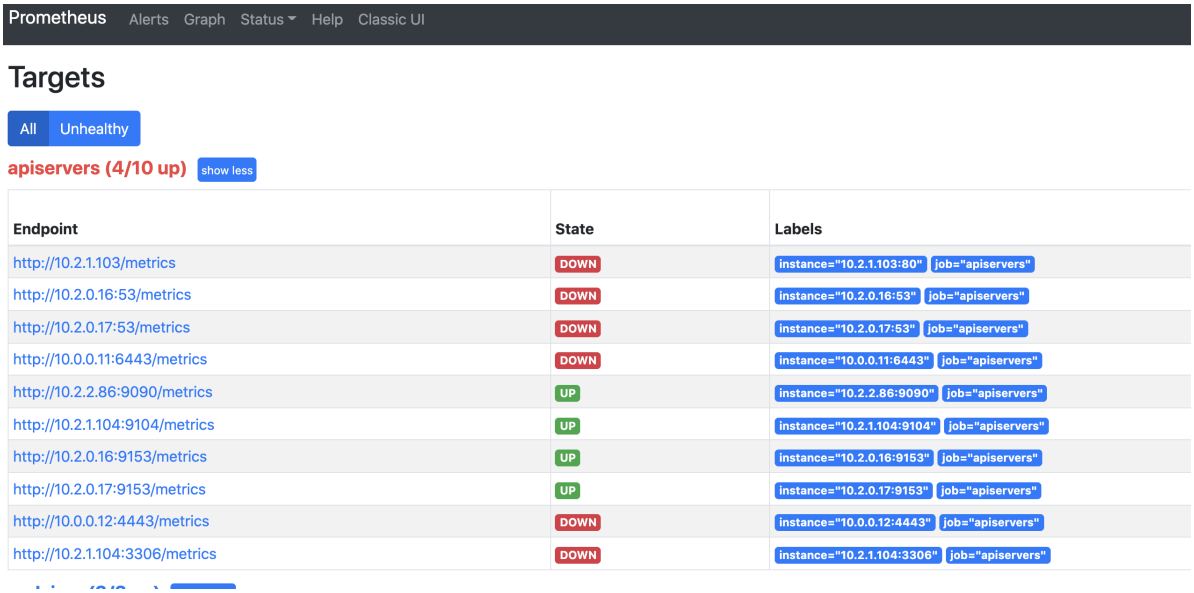
修改配置：

```
    - job_name: 'apiservers'
      kubernetes_sd_configs:
      - role: endpoints
```

应用配置：

```
kubectl apply -f prom-cm.yml
curl -X POST "http://10.2.2.86:9090/-/reload"
```

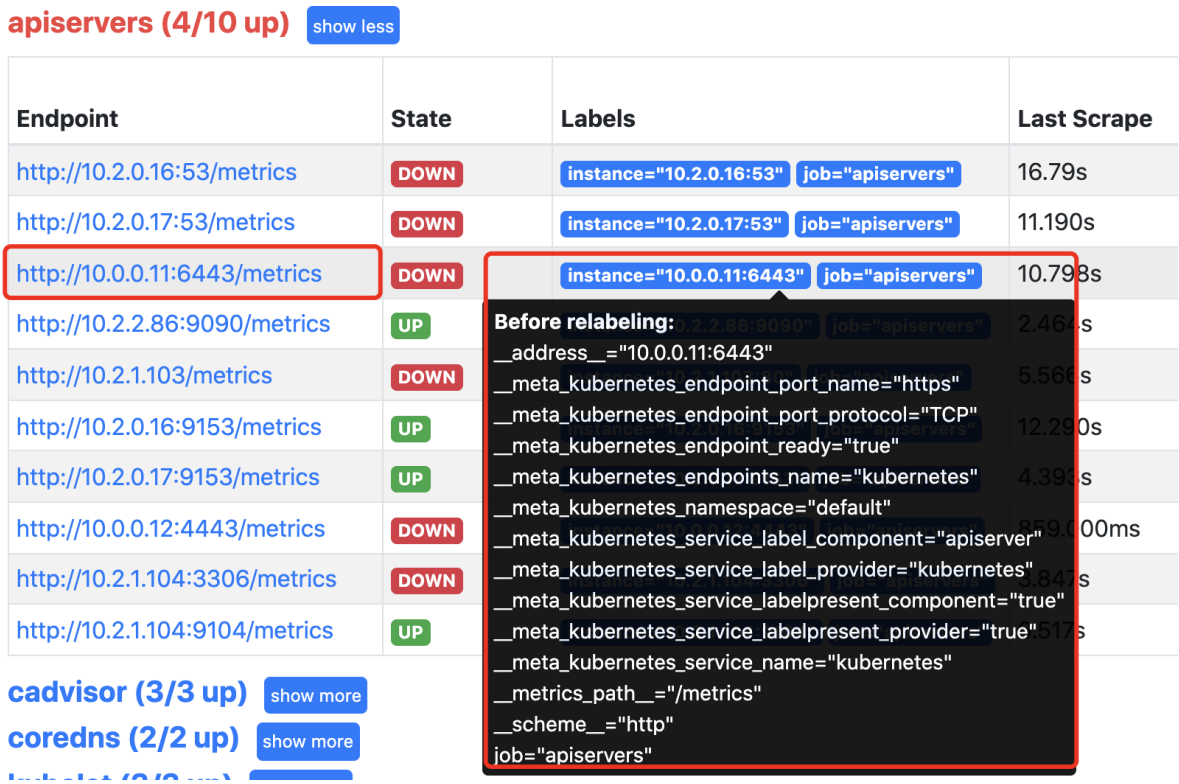
查看结果：



这时我们发现promtheus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。

```
[root@node1 prom]# kubectl describe svc kubernetes
Name:              kubernetes
Namespace:         default
Labels:            component=apiserver
                   provider=kubernetes
Annotations:       <none>
Selector:          <none>
Type:              ClusterIP
IP:                10.1.0.1
Port:              https  443/TCP
TargetPort:        6443/TCP
Endpoints:         10.0.0.11:6443
Session Affinity:  None
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。



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：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: prom
data:
  prometheus.yml: |
    global:
      scrape_interval: 15s
      scrape_timeout: 15s
    scrape_configs:
    - job_name: 'prometheus'
      static_configs:
      - targets: ['localhost:9090']

    - job_name: 'coredns'
      static_configs:
      - targets: ['10.2.0.16:9153','10.2.0.17:9153']

    - job_name: 'mysql'
      static_configs:
      - targets: ['mysql-svc:9104']

    - job_name: 'nodes'
      kubernetes_sd_configs:
      - role: node
      relabel_configs:
      - source_labels: ['__address__']
        regex: '(.*):10250'
```

```
        replacement: '${1}:9100'
        target_label: __address__
        action: replace
      - action: labelmap
        regex: __meta_kubernetes_node_label_(.+)

    - job_name: 'kubelet'
      kubernetes_sd_configs:
      - role: node
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - action: labelmap
        regex: __meta_kubernetes_node_label_(.+)

    - job_name: 'cadvisor'
      kubernetes_sd_configs:
      - role: node
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - action: labelmap
        regex: __meta_kubernetes_node_label_(.+)
        replacement: $1
      - source_labels: [__meta_kubernetes_node_name]
        regex: (.*)
        replacement: /metrics/cadvisor
        target_label: __metrics_path__

    - job_name: 'apiservers'
      kubernetes_sd_configs:
      - role: endpoints
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - source_labels: [__meta_kubernetes_service_label_component]
        action: keep
        regex: apiserver
```

应用修改：

```
kubectl apply -f prom-cm.yml
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	instance="10.0.0.11:6443" job="apiservers"	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:

```
    - job_name: 'pods'
      kubernetes_sd_configs:
      - role: endpoints
      relabel_configs:
      - source_labels:
[__meta_kubernetes_service_annotation_prometheus_io_scrape]
        action: keep
        regex: true
      - action: labelmap
        regex: __meta_kubernetes_service_label_(.+)
      - source_labels: [__address__,
__meta_kubernetes_service_annotation_prometheus_io_port]
        action: replace
        target_label: __address__
        regex: ([^:]+)(?::\d+)?;(\d+)
        replacement: $1:$2
      - source_labels: [__meta_kubernetes_namespace]
        action: replace
        target_label: kubernetes_namespace
      - source_labels: [__meta_kubernetes_service_name]
        action: replace
        target_label: kubernetes_name
      - source_labels: [__meta_kubernetes_pod_name]
        action: replace
        target_label: kubernetes_pod_name
```

生效配置:

```
kubectl apply -f prom-cm.yml
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	instance="10.2.0.17:9153" job="pods" 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"	1.977s	11.130ms
http://10.2.0.16:9153/metrics	UP	instance="10.2.0.16:9153" job="pods" 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"	5.823s	4.332ms

匹配参数解释：

```
__meta_kubernetes_service_annotation_prometheus_io_scrape 为 true
__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

```
[root@node1 prom]# kubectl -n kube-system get svc kube-dns -o yaml
apiVersion: v1
kind: Service
metadata:
  annotations:
    prometheus.io/port: "9153"
    prometheus.io/scrape: "true"
  labels:
    k8s-app: kube-dns
    kubernetes.io/cluster-service: "true"
    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。

```
---
kind: Service
apiVersion: v1
metadata:
  name: mysql-svc
  namespace: prom
```

```
  annotations:
    prometheus.io/scrape: "true"
    prometheus.io/port: "9104"
spec:
  selector:
    app: mysql
  ports:
  - name: mysql
    port: 3306
    targetPort: 3306
  - name: mysql-prom
    port: 9104
    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	instance="10.2.1.104:9104" job="pods" kubernetes_name="mysql-svc" kubernetes_namespace="prom" kubernetes_pod_name="mysql-dp-79b48cff96-m96bz"	2.228s	19.243ms
http://10.2.0.16:9153/metrics	UP	instance="10.2.0.16:9153" job="pods" 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"	10.617s	5.563ms
http://10.2.0.17:9153/metrics	UP	instance="10.2.0.17:9153" job="pods" 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"	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：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: prom
data:
  prometheus.yml: |
    global:
      scrape_interval: 15s
      scrape_timeout: 15s
    scrape_configs:
    - job_name: 'prometheus'
      static_configs:
      - targets: ['localhost:9090']

    #- job_name: 'coredns'
    #  static_configs:
    #  - 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:
  /var/run/secrets/kubernetes.io/serviceaccount/token
45        relabel_configs:
46        - action: labelmap
47          regex: __meta_kubernetes_node_label_(.+)
48
49      - job_name: 'cadvisor'
50        kubernetes_sd_configs:
51        - role: node
52        scheme: https
53        tls_config:
54          ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
55          insecure_skip_verify: true
56        bearer_token_file:
  /var/run/secrets/kubernetes.io/serviceaccount/token
57        relabel_configs:
58        - action: labelmap
59          regex: __meta_kubernetes_node_label_(.+)
60          replacement: $1
61        - source_labels: [__meta_kubernetes_node_name]
62          regex: (.*)
63          replacement: /metrics/cadvisor
64          target_label: __metrics_path__
65
66      - job_name: 'apiservers'
67        kubernetes_sd_configs:
68        - role: endpoints
69        scheme: https
70        tls_config:
71          ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
72          insecure_skip_verify: true
73        bearer_token_file:
  /var/run/secrets/kubernetes.io/serviceaccount/token
```

```
      relabel_configs:
      - source_labels: [__meta_kubernetes_service_label_component]
        action: keep
        regex: apiserver

    - job_name: 'pods'
      kubernetes_sd_configs:
      - role: endpoints
      relabel_configs:
      - source_labels:
[__meta_kubernetes_service_annotation_prometheus_io_scrape]
        action: keep
        regex: true
      - action: labelmap
        regex: __meta_kubernetes_service_label_(.+)
      - source_labels: [__address__,
__meta_kubernetes_service_annotation_prometheus_io_port]
        action: replace
        target_label: __address__
        regex: ([^:]+)(?::\d+)?;(\d+)
        replacement: $1:$2
      - source_labels: [__meta_kubernetes_namespace]
        action: replace
        target_label: kubernetes_namespace
      - source_labels: [__meta_kubernetes_service_name]
        action: replace
        target_label: kubernetes_name
      - source_labels: [__meta_kubernetes_pod_name]
        action: replace
        target_label: kubernetes_pod_name

    - job_name: 'coredns'
      kubernetes_sd_configs:
      - role: endpoints
      relabel_configs:
      - source_labels:
[__meta_kubernetes_service_annotation_prometheus_io_scrape]
        action: keep
        regex: true
      - action: labelmap
        regex: __meta_kubernetes_service_label_(.+)
      - source_labels: [__address__,
__meta_kubernetes_service_annotation_prometheus_io_port]
        action: replace
        target_label: __address__
        regex: ([^:]+)(?::\d+)?;(\d+)
        replacement: $1:$2
      - source_labels: [__meta_kubernetes_namespace]
        action: replace
        target_label: kubernetes_namespace
      - source_labels: [__meta_kubernetes_service_name]
        action: replace
        target_label: kubernetes_name
      - source_labels: [__meta_kubernetes_pod_name]
        action: replace
        target_label: kubernetes_pod_name
      - source_labels: [__meta_kubernetes_endpoints_name]
        action: keep
```

```
128           regex: kube-dns
```

应用配置：

```
1 kubectl apply -f prom-cm.yml
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
```

```
vim service.yaml
------------------
apiVersion: v1
kind: Service
metadata:
  labels:
    app.kubernetes.io/name: kube-state-metrics
    app.kubernetes.io/version: 2.1.1
  annotations:
    prometheus.io/scrape: "true"
    prometheus.io/port: "8080"
  name: kube-state-metrics
  namespace: kube-system
  ......................
------------------
```

创建资源配置：

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

查看promtheus：

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：

```
存在执行失败的Job: kube_job_status_failed
集群节点状态错误: kube_node_status_condition{condition="Ready",
status!="true"}==1
集群中存在启动失败的 Pod: kube_pod_status_phase{phase=~"Failed|Unknown"}==1
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：

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

第x章 grafana

grafana介绍

```

```

安装部署

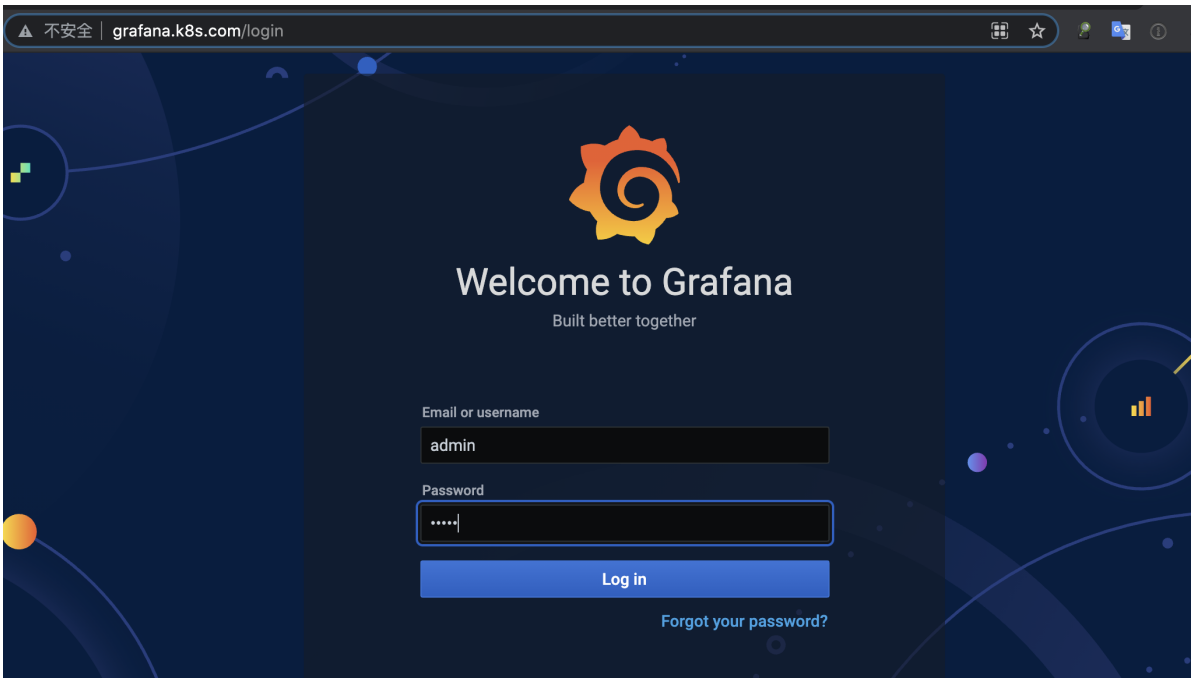
```
cat > grafana.yml << 'EOF'
apiVersion: apps/v1
kind: Deployment
metadata:
  name: grafana
  namespace: prom
spec:
  selector:
    matchLabels:
      app: grafana
  template:
    metadata:
      labels:
        app: grafana
    spec:
      volumes:
      - name: storage
        hostPath:
          path: /data/k8s/grafana/
      nodeSelector:
        kubernetes.io/hostname: node2
      securityContext:
        runAsUser: 0
      containers:
      - name: grafana
        image: grafana/grafana:7.4.3
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 3000
          name: grafana
        env:
        - name: GF_SECURITY_ADMIN_USER
          value: admin
        - name: GF_SECURITY_ADMIN_PASSWORD
          value: admin
        readinessProbe:
          failureThreshold: 10
          httpGet:
            path: /api/health
            port: 3000
            scheme: HTTP
          initialDelaySeconds: 60
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 30
        livenessProbe:
          failureThreshold: 3
          httpGet:
            path: /api/health
            port: 3000
            scheme: HTTP
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 3
        resources:
          limits:
            cpu: 150m
```

```
            memory: 512Mi
          requests:
            cpu: 150m
            memory: 512Mi
        volumeMounts:
        - mountPath: /var/lib/grafana
          name: storage
---
apiVersion: v1
kind: Service
metadata:
  name: grafana
  namespace: prom
spec:
  ports:
    - port: 3000
  selector:
    app: grafana
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: grafana
  namespace: prom
  labels:
    app: grafana
spec:
  rules:
  - host: grafana.k8s.com
    http:
      paths:
      - path: /
        pathType: ImplementationSpecific
        backend:
          service:
            name: grafana
            port:
              number: 3000
EOF
```

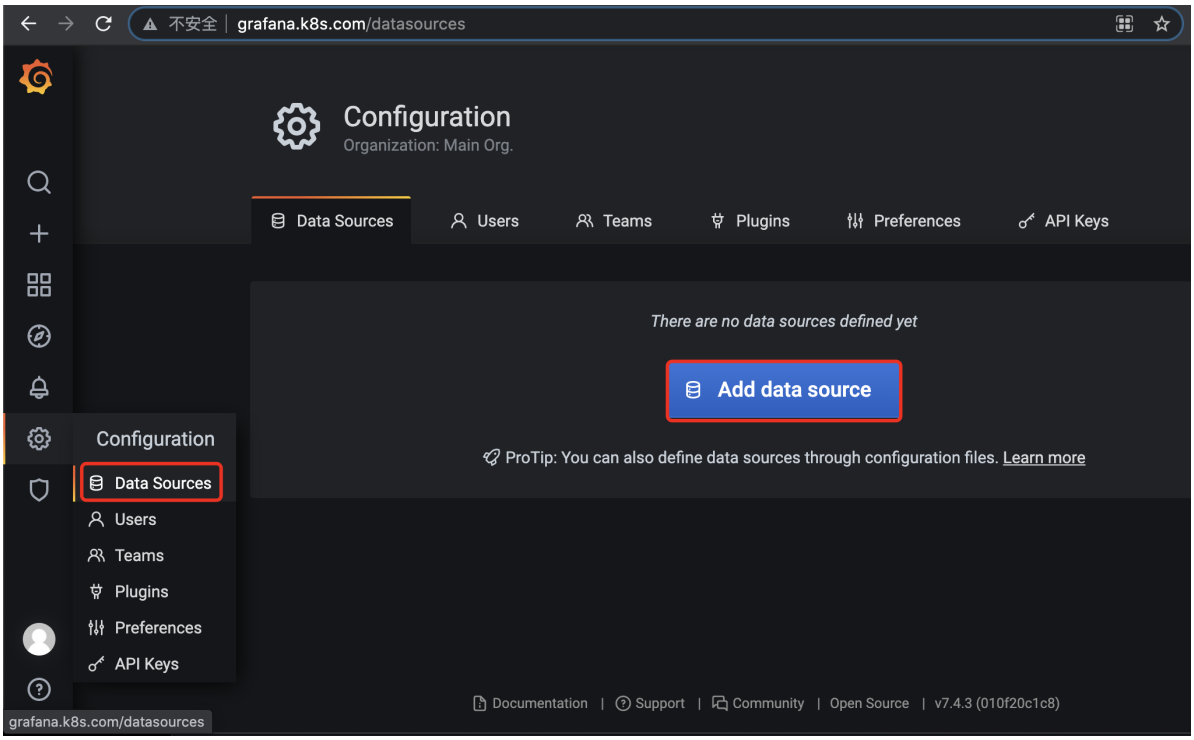
应用资源配置:

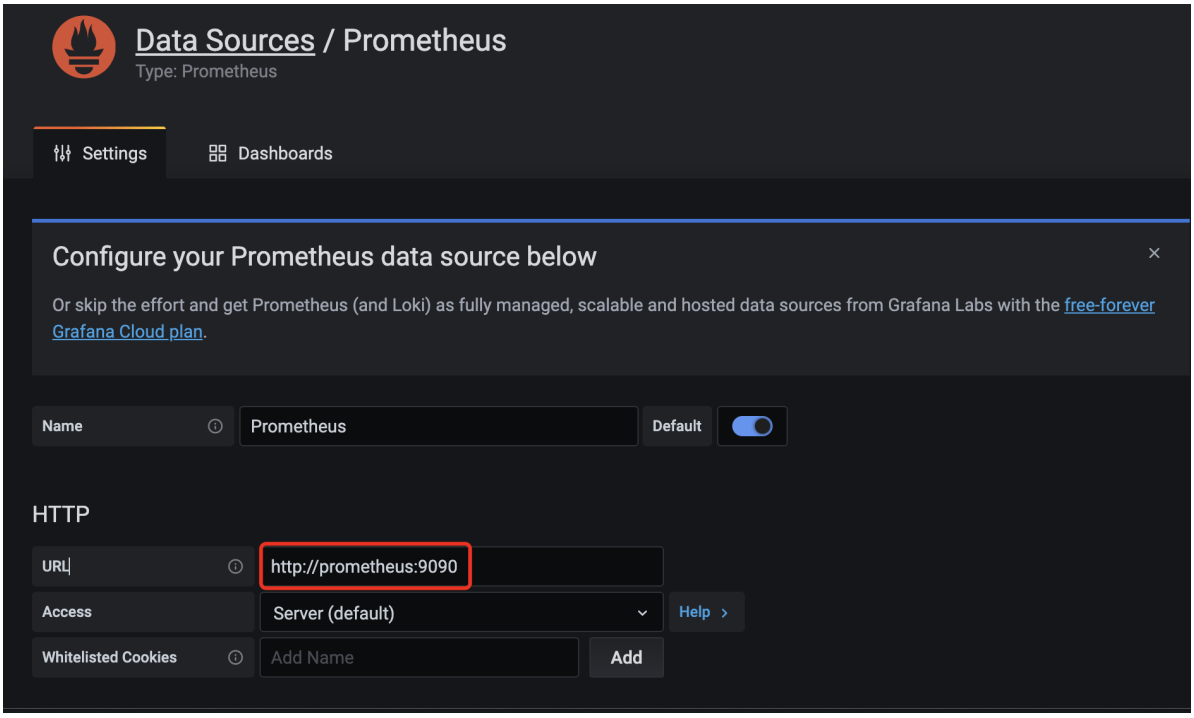
```
[root@node1 prom]# kubectl apply -f grafana.yml
deployment.apps/grafana created
service/grafana created
ingress.networking.k8s.io/prometheus configured
```

访问测试：


不安全 | grafana.k8s.com/login
Welcome to Grafana
Built better together
Email or username
admin
Password
Log in
Forgot your password?

添加数据源


不安全 | grafana.k8s.com/datasources
Configuration
Organization: Main Org.
Data Sources
Users
Teams
Plugins
Preferences
API Keys
There are no data sources defined yet
Add data source
ProTip: You can also define data sources through configuration files. Learn more
Configuration
Data Sources
Users
Teams
Plugins
Preferences
API Keys
Documentation | Support | Community | Open Source | v7.4.3 (010f20c1c8)
grafana.k8s.com/datasources



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：

```
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：

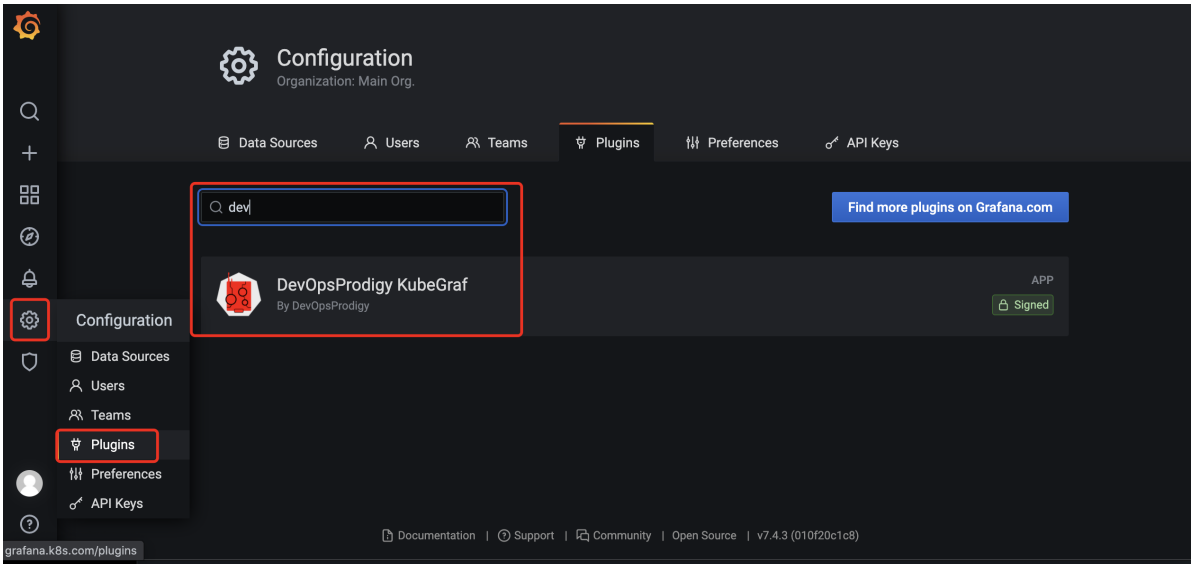
```
[root@node1 prom]# kubectl -n prom exec -it grafana-7f5b7455fc-z6ctx -- /bin/bash
bash-5.0# grafana-cli plugins install devopsprodigy-kubegraf-app
installing devopsprodigy-kubegraf-app @ 1.5.2
from: https://grafana.com/api/plugins/devopsprodigy-kubegraf-app/versions/1.5.2/download
into: /var/lib/grafana/plugins

✓ Installed devopsprodigy-kubegraf-app successfully
installing grafana-piechart-panel @ 1.6.2
from: https://grafana.com/api/plugins/grafana-piechart-panel/versions/1.6.2/download
into: /var/lib/grafana/plugins

✓ Installed grafana-piechart-panel successfully
Installed dependency: grafana-piechart-panel ✓

```

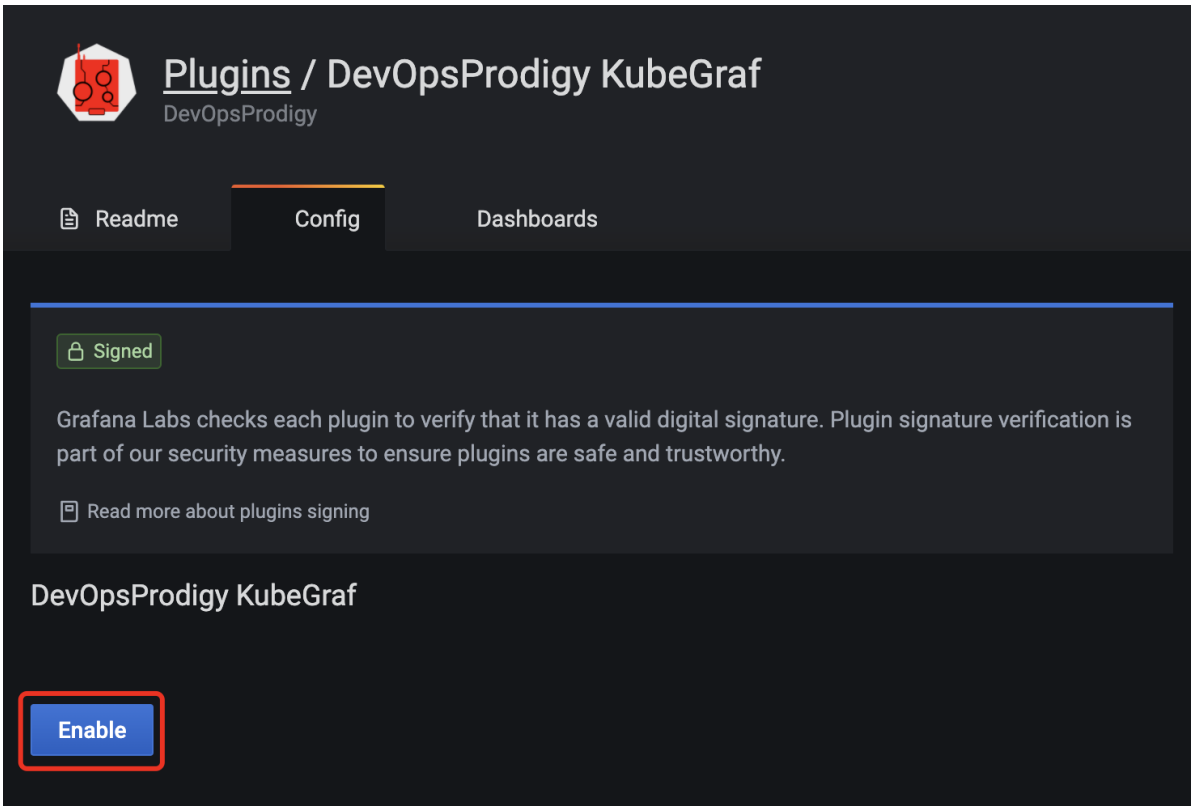
```
Restart grafana after installing plugins . <service grafana-server restart>

bash-5.0#
```

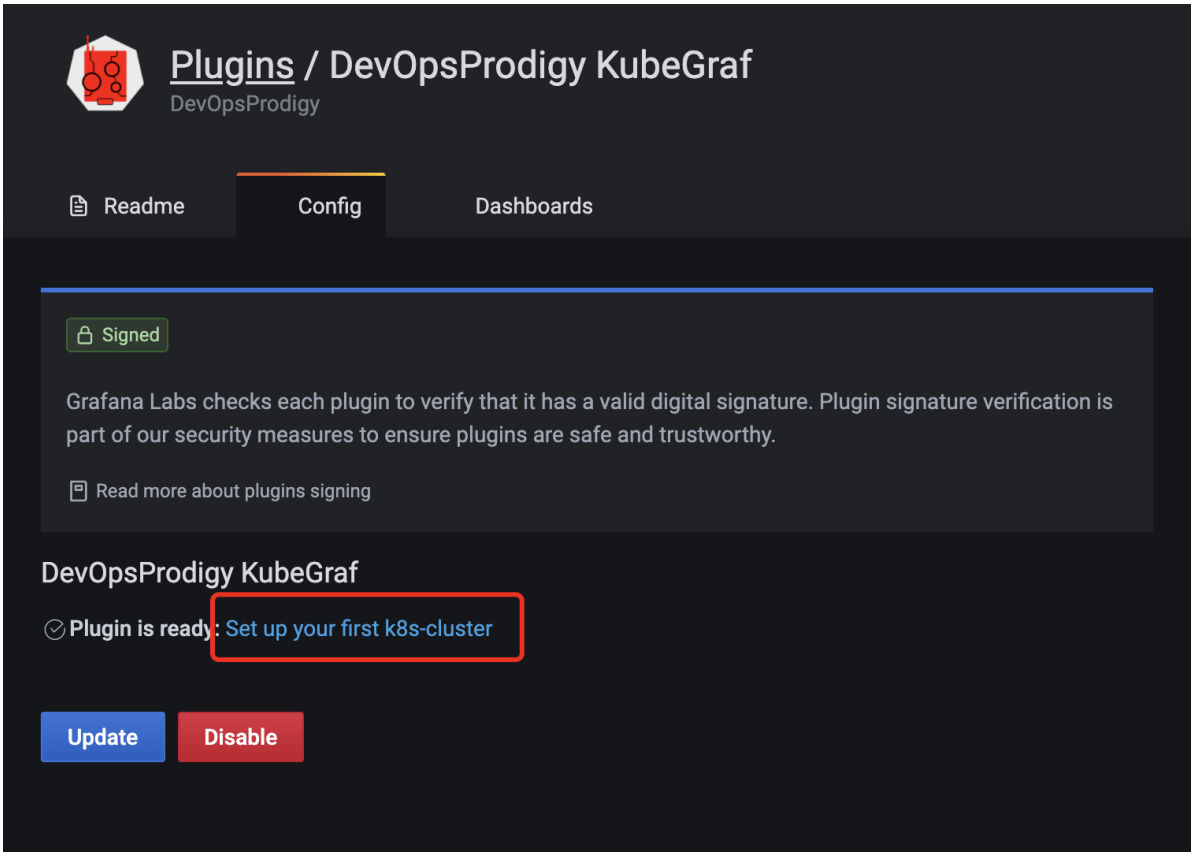
安装完成后我们还需要重启一下grafana才能生效，因为我们做了数据持久化，所以直接删除pod重新创建即可。

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

重启之后我们在grafana页面激活插件



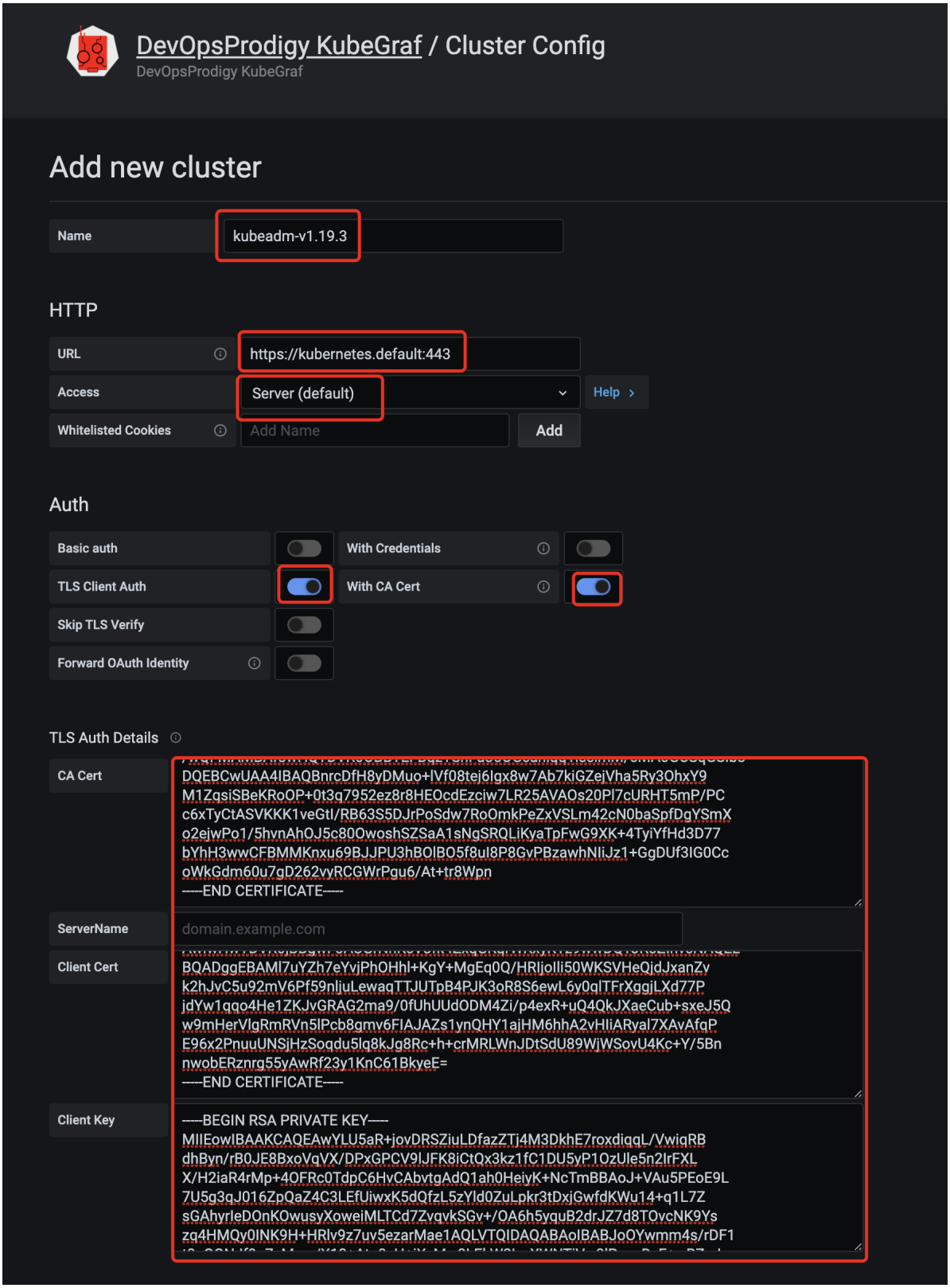


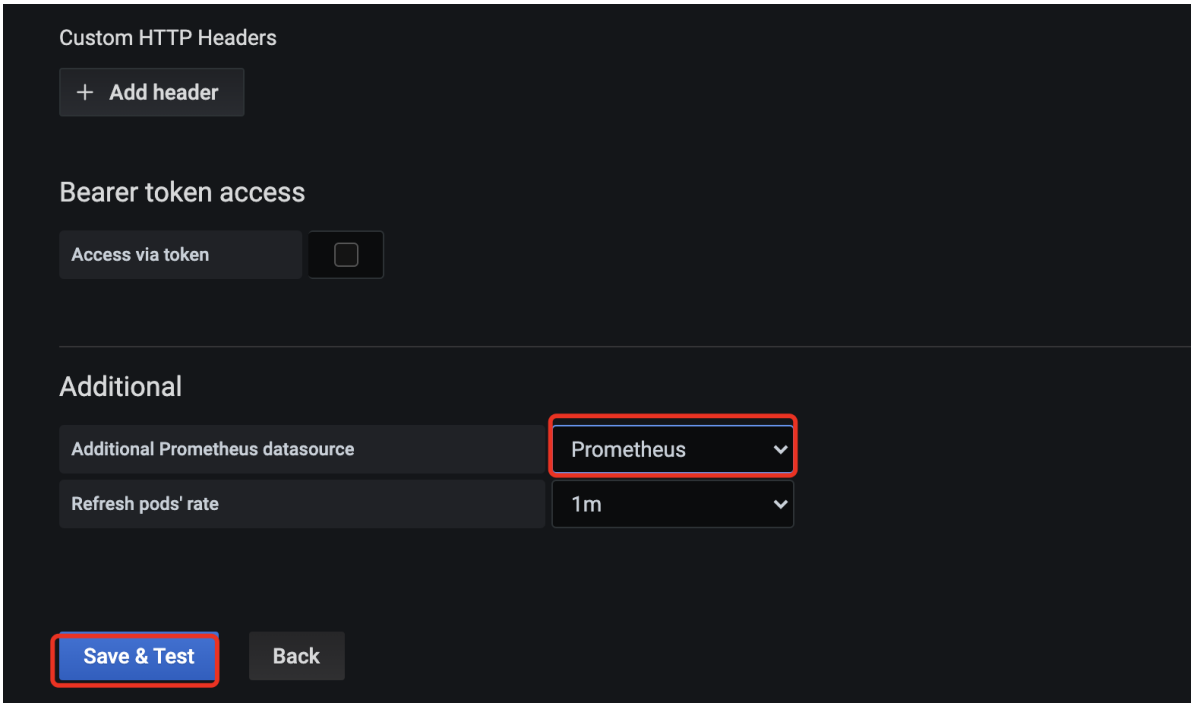


这里需要对验证，我们使用kubectl的kubeconfig配置文件的内容来进行配置：

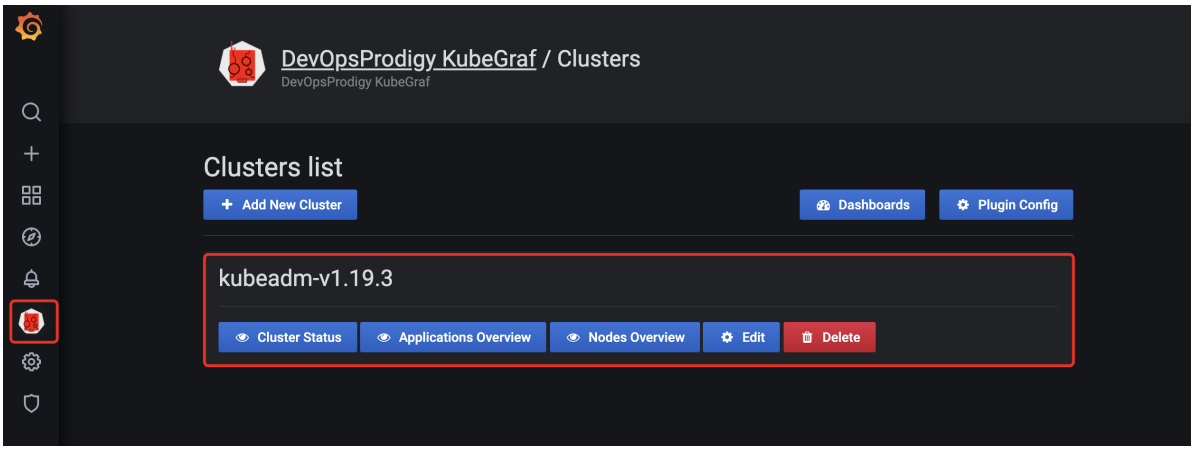
```
[root@node1 prom]# cat ~/.kube/config
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data:     #CA Cert的值
..............
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data:            #Client Cert的值
    client-key-data:                            #Client Key的值
..............
```

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：

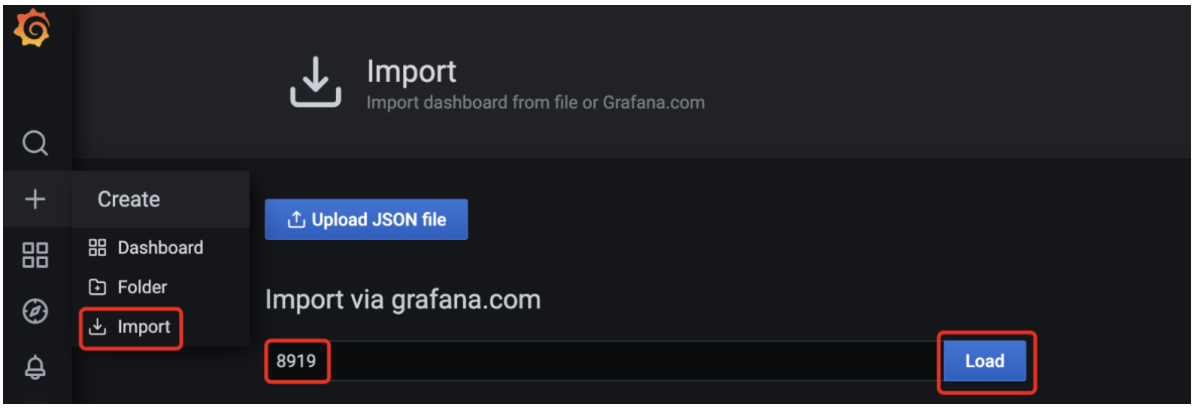

DevOpsProdigy KubeGraf / Cluster Config
DevOpsProdigy KubeGraf
Add new cluster
Name
kubeadm-v1.19.3
HTTP
URL
https://kubernetes.default:443
Access
Server (default)
Help
Whitelisted Cookies
Add Name
Add
Auth
Basic auth
With Credentials
TLS Client Auth
With CA Cert
Skip TLS Verify
Forward OAuth Identity
TLS Auth Details
CA Cert
DQEBCwUAA4IBAQBnrcDfH8yDMuo+lVf08tej6lgx8w7Ab7kiGZejVha5Ry3OhxY9
M1ZqsiSBeKRoOP+0t3q7952ez8r8HEOcdEzciw7LR25AVAOs20Pl7cURHT5mP/PC
c6xTyCtASVKKK1veGtI/RB63S5DJrPoSdw7RoOmkPeZxVSLm42cN0baSpfDgYSmX
o2ejwPo1/5hvnAhOJ5c80OwoshSZSaA1sNgSRQLiKyaTpFwG9XK+4TyiYfHd3D77
bYhH3wwCFBMMKnxu69BJJPU3hBOlBO5f8ul8P8GvPBzawhNIiJz1+GgDUf3IG0Cc
oWkGdm60u7gD262vyRCGWrPgu6/At+tr8Wpn
-----END CERTIFICATE-----
ServerName
domain.example.com
Client Cert
BQADggEBAMI7uYZh7eYvjPhOHhl+KgY+MgEq0Q/HRIjoIli50WKSVHeQjdJxanZv
k2hJvC5u92mV6Pf59nIjuLewaqTTJUTpB4PJK3oR8S6ewL6y0qlTFrXggjLXd77P
jdYw1qqo4He1ZKJvGRAG2ma9/0fUhUUdODM4Zi/p4exR+uQ4QkJXaeCub+sxeJ5Q
w9mHerVlgRmRVn5lPcb8gmv6FIAJAZs1ynQHY1ajHM6hhA2vHliARyal7XAvAfqP
E96x2PnuuUNSjHzSoqdu5lq8kJq8Rc+h+crMRLWnJDtSdU89WjWSovU4Kc+Y/5Bn
nwobERznrg55yAwRf23y1KnC61BkyeE=
-----END CERTIFICATE-----
Client Key
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAwYLU5aR+jovDRSZiuLDfazZTj4M3DkhE7roxdiqqL/VwiqRB
dhByn/rB0JE8BxoVqVX/DPxGPCV9lJFK8iCtQx3kz1fC1DU5yP1OzUle5n2IrFXL
X/H2iaR4rMp+4OFRc0TdpC6HvCAbvtgAdQ1ah0HeiyK+NcTmBBAoJ+VAu5PEoE9L
7U5g3qJ016ZpQaZ4C3LEfUiwxK5dQfzL5zYld0ZuLpkr3tDxjGwfdKWu14+q1L7Z
sGAhyrIeDOnKOwusyXoweiMLTCd7ZvqvkSGv+/OA6h5yquB2drJZ7d8TOvcNK9Ys
zq4HMQv0INK9H+HRlv9z7uv5ezarMae1AQLVTQIDAQABAoIBABJoOYwmm4s/rDF1

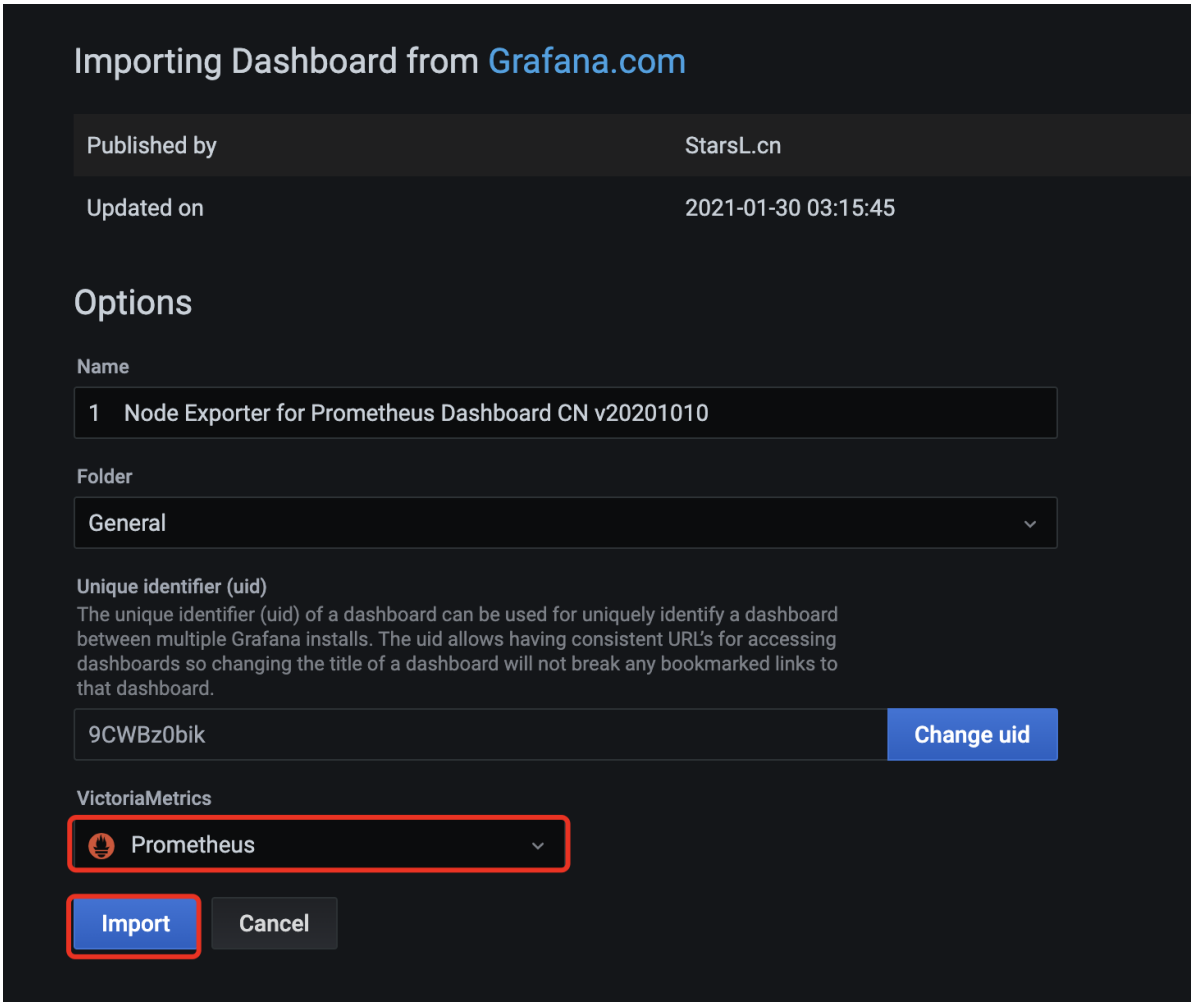


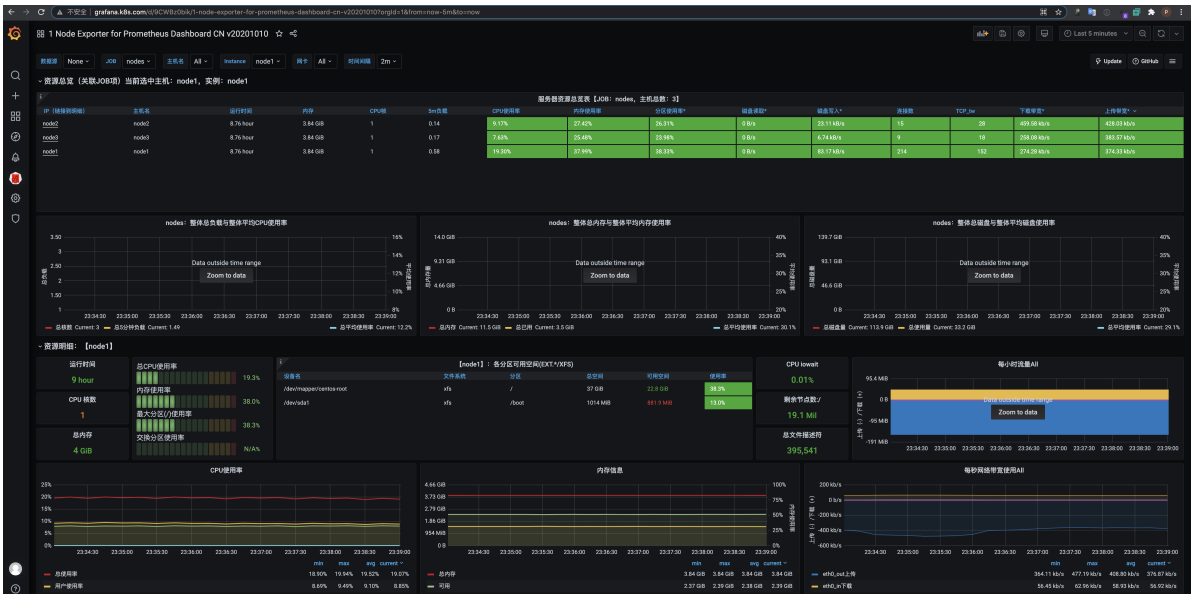
保存之后左边就会出现插件的图标，点击就可以查看了



导入dashboard







第x章 Alertmanager报警

安装

```
cat > alertmanager.yml << 'EOF'
apiVersion: v1
kind: ConfigMap
metadata:
  name: alert-config
  namespace: prom
data:
```

```
  config.yml: |-
    global:
      # 当alertmanager持续多长时间未接收到告警后标记告警状态为 resolved
      resolve_timeout: 5m
      # 配置邮件发送信息
      smtp_smarthost: 'smtp.qq.com:465'
      smtp_from: '526195417@qq.com'
      smtp_auth_username: '526195417@qq.com'
      smtp_auth_password: 'mtbxhvaqonvrbgia'
      smtp_hello: 'qq.com'
      smtp_require_tls: false
    # 所有报警信息进入后的根路由，用来设置报警的分发策略
    route:
      # 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=LatncyHigh 这样的标签的报警信息将会批量被聚合到一个分组里面
      group_by: ['alertname', 'cluster']
      # 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。
      group_wait: 30s

      # 相同的group之间发送告警通知的时间间隔
      group_interval: 30s

      # 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
      repeat_interval: 1h

      # 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
      receiver: default

      # 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。
      routes:
      - receiver: email
        group_wait: 10s
        match:
          team: node
    receivers:
    - name: 'default'
      email_configs:
      - to: '526195417@qq.com'
        send_resolved: true  # 接受告警恢复的通知
    - name: 'email'
      email_configs:
      - to: '526195417@qq.com'
        send_resolved: true
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: alertmanager
  namespace: prom
  labels:
    app: alertmanager
spec:
  selector:
    matchLabels:
      app: alertmanager
```

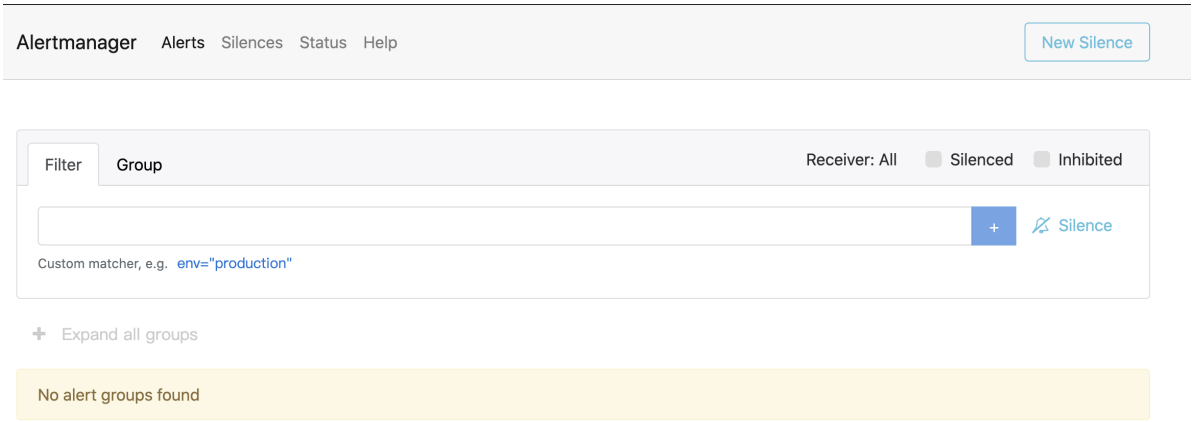
```yaml
  template:
    metadata:
      labels:
        app: alertmanager
    spec:
      volumes:
      - name: alertcfg
        configMap:
          name: alert-config
      containers:
      - name: alertmanager
        image: prom/alertmanager:v0.21.0
        imagePullPolicy: IfNotPresent
        args:
        - "--config.file=/etc/alertmanager/config.yml"
        ports:
        - containerPort: 9093
          name: http
        volumeMounts:
        - mountPath: "/etc/alertmanager"
          name: alertcfg
        resources:
          requests:
            cpu: 100m
            memory: 256Mi
          limits:
            cpu: 100m
            memory: 256Mi
---
apiVersion: v1
kind: Service
metadata:
  name: alertmanager
  namespace: prom
  labels:
    app: alertmanager
spec:
  selector:
    app: alertmanager
  ports:
    - name: web
      port: 9093
      targetPort: http
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: alertmanager
  namespace: prom
  labels:
    app: alertmanager
spec:
  rules:
  - host: alertmanager.k8s.com
    http:
      paths:
      - path: /
        pathType: ImplementationSpecific
```

```
        backend:
          service:
            name: alertmanager
            port:
              number: 9093
EOF
```

应用配置：

```
[root@node1 prom]# kubectl apply -f alertmanager.yml
configmap/alert-config created
deployment.apps/alertmanager created
service/alertmanager created
ingress.networking.k8s.io/alertmanager created
```

访问查看



配置报警规则

编写报警规则

```
cat > alert-rules.yml << 'EOF'
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: kube-mon
data:
  prometheus.yml: |
    global:
      scrape_interval: 15s
      scrape_timeout: 15s
      evaluation_interval: 30s  # 默认情况下每分钟对告警规则进行计算
    alerting:
      alertmanagers:
      - static_configs:
        - targets: ["alertmanager:9093"]
    rule_files:
    - /etc/prometheus/rules.yml
  ...... # 省略prometheus其他部分
  rules.yml: |
    groups:
    - name: test-node-mem
```

```
      rules:
      - alert: NodeMemoryUsage
        expr: (node_memory_MemTotal_bytes - (node_memory_MemFree_bytes + node_memory_Buffers_bytes + node_memory_Cached_bytes)) / node_memory_MemTotal_bytes * 100 > 20
        for: 2m
        labels:
          team: node
        annotations:
          summary: "{{$labels.instance}}: High Memory usage detected"
          description: "{{$labels.instance}}: Memory usage is above 20% (current value is: {{ $value }}"
EOF
```