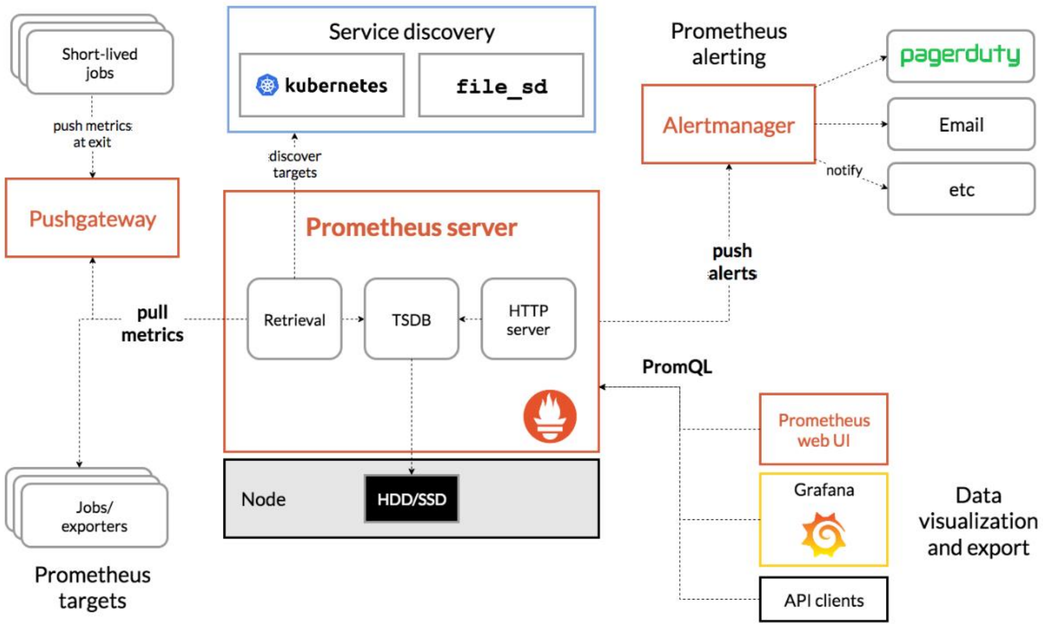
第一章 Kubernetes 监控告警

1.1 Prometheus 架构

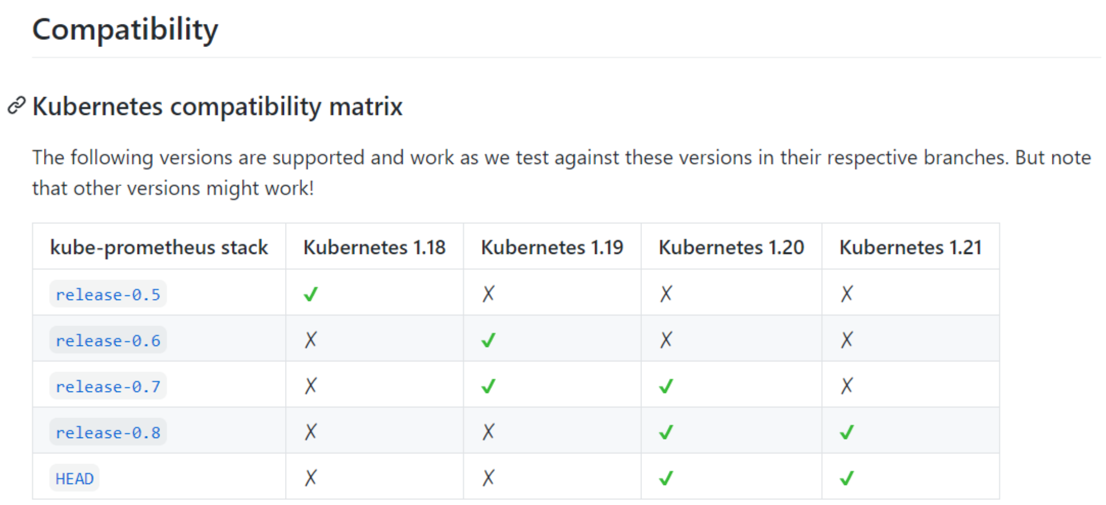


- Prometheus Server：Prometheus 生态最重要的组件，主要用于抓取和存储时间序列数据，同时提供数据的查询和告警策略的配置管理；
- Alertmanager：Prometheus 生态用于告警的组件，Prometheus Server 会将告警发送给 Alertmanager，Alertmanager 根据路由配置，将告警信息发送给指定的人或组。Alertmanager 支持邮件、Webhook、微信、钉钉、短信等媒介进行告警通知；
- Grafana：用于展示数据，便于数据的查询和观测；
- Push Gateway：Prometheus 本身是通过 Pull 的方式拉取数据，但是有些监控数据可能是短期的，如果没有采集数据可能会出现丢失。Push Gateway 可以用来解决此类问题，它可以用来接收数据，也就是客户端可以通过 Push 的方式将数据推送到 Push Gateway，之后 Prometheus 可以通过 Pull 拉取该数据；
- Exporter：主要用来采集监控数据，比如主机的监控数据可以通过 node_exporter 采集，MySQL 的监控数据可以通过 mysql_exporter 采集，之后 Exporter 暴露一个接口，比如/metrics，Prometheus 可以通过该接口采集到数据；
- PromQL：PromQL 其实不算 Prometheus 的组件，它是用来查询数据的一种语法，比如查询数据库的数据，可以通过 SQL 语句，查询 Loki 的数据，可以通过 LogQL，查询 Prometheus 数据的叫做 PromQL；
- Service Discovery：用来发现监控目标的自动发现，常用的有基于 Kubernetes、Consul、Eureka、文件的自动发现等。

1.2 Prometheus 安装

Kube-Prometheus 项目地址：https://github.com/prometheus-operator/kube-prometheus/

首先需要通过该项目地址，找到和自己 Kubernetes 版本对应的 Kube Prometheus Stack 的版本：



Compatibility

Kubernetes compatibility matrix

The following versions are supported and work as we test against these versions in their respective branches. But note that other versions might work!

kube-prometheus stack	Kubernetes 1.18	Kubernetes 1.19	Kubernetes 1.20	Kubernetes 1.21
`release-0.5`	✓	✗	✗	✗
`release-0.6`	✗	✓	✗	✗
`release-0.7`	✗	✓	✓	✗
`release-0.8`	✗	✗	✓	✓
`HEAD`	✗	✗	✓	✓

```
    # git clone -b release-0.8 https://github.com/prometheus-operator/kube-prometheus.git
    # cd kube-prometheus/manifests
```

安装 Prometheus Operator：

```
    # kubectl create -f setup/
    namespace/monitoring created
    ...
    deployment.apps/prometheus-operator created
    service/prometheus-operator created
    serviceaccount/prometheus-operator created
```

查看 Operator 容器的状态：

```
    # kubectl get po -n monitoring
    NAME                                READY   STATUS    RESTARTS   AGE
    prometheus-operator-bb5c5b6c8-xtkdn  2/2     Running   0          25s
```

Operator 容器启动后，安装 Prometheus Stack：

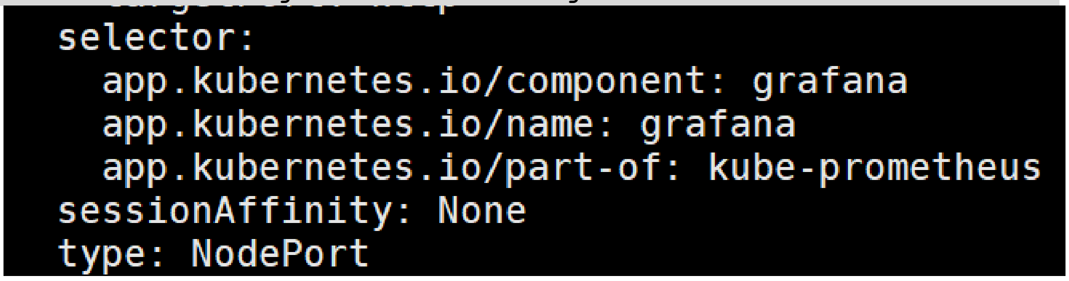
```
    # kubectl create -f .
    alertmanager.monitoring.coreos.com/main created
    ...
    service/prometheus-k8s created
    serviceaccount/prometheus-k8s created
    servicemonitor.monitoring.coreos.com/prometheus-k8s created
```

查看 Prometheus 容器状态：

```
    # kubectl get po -n monitoring
    NAME                                READY   STATUS    RESTARTS   AGE
    alertmanager-main-0                   2/2     Running   0          59s
    alertmanager-main-1                   2/2     Running   0          59s
    alertmanager-main-2                   2/2     Running   0          59s
    blackbox-exporter-7f88596689-fl2v8     3/3     Running   0          59s
    grafana-766bfd54b9-cchqm               1/1     Running   0          58s
    kube-state-metrics-5fd8b545b-hrzxc     3/3     Running   0          58s
    node-exporter-265df                   2/2     Running   0          58s
    node-exporter-5qj7b                   2/2     Running   0          58s
    node-exporter-lxngk                   2/2     Running   0          58s
    node-exporter-n8p7w                   2/2     Running   0          58s
    node-exporter-xjjf2                   2/2     Running   0          58s
    prometheus-adapter-5b849bbc57-tlwvd    1/1     Running   0          58s
    prometheus-adapter-5b849bbc57-xjznh    1/1     Running   0          58s
    prometheus-k8s-0                     2/2     Running   1          57s
    prometheus-k8s-1                     2/2     Running   1          57s
    prometheus-operator-bb5c5b6c8-xtkdn    2/2     Running   0          18m
```

将 Grafana 的 Service 改成 NodePort 类型：

```
# kubectl edit svc grafana -n monitoring
```

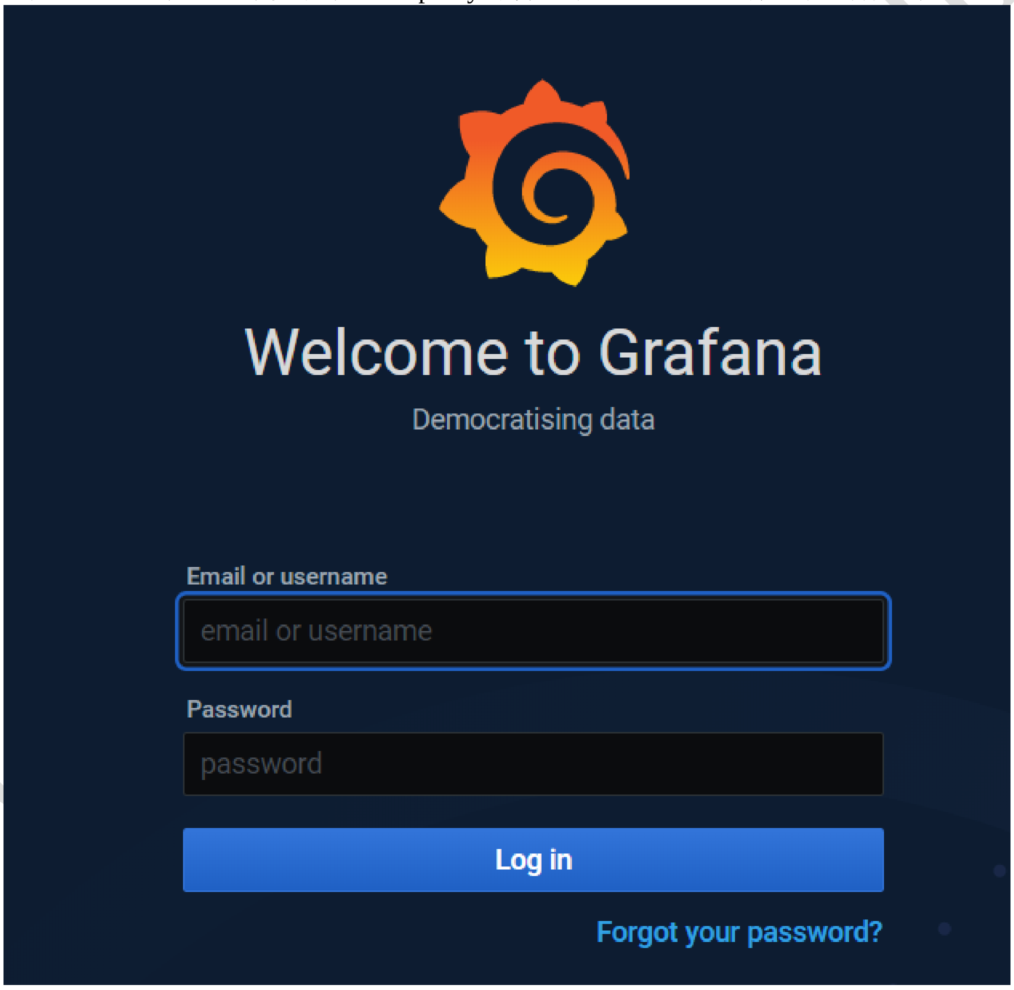


```
  selector:
    app.kubernetes.io/component: grafana
    app.kubernetes.io/name: grafana
    app.kubernetes.io/part-of: kube-prometheus
  sessionAffinity: None
  type: NodePort
```

查看 Grafana Service 的 NodePort：

```
# kubectl get svc grafana -n monitoring
NAME      TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
grafana   NodePort   192.168.183.25   <none>        3000:31266/TCP   4m56s
```

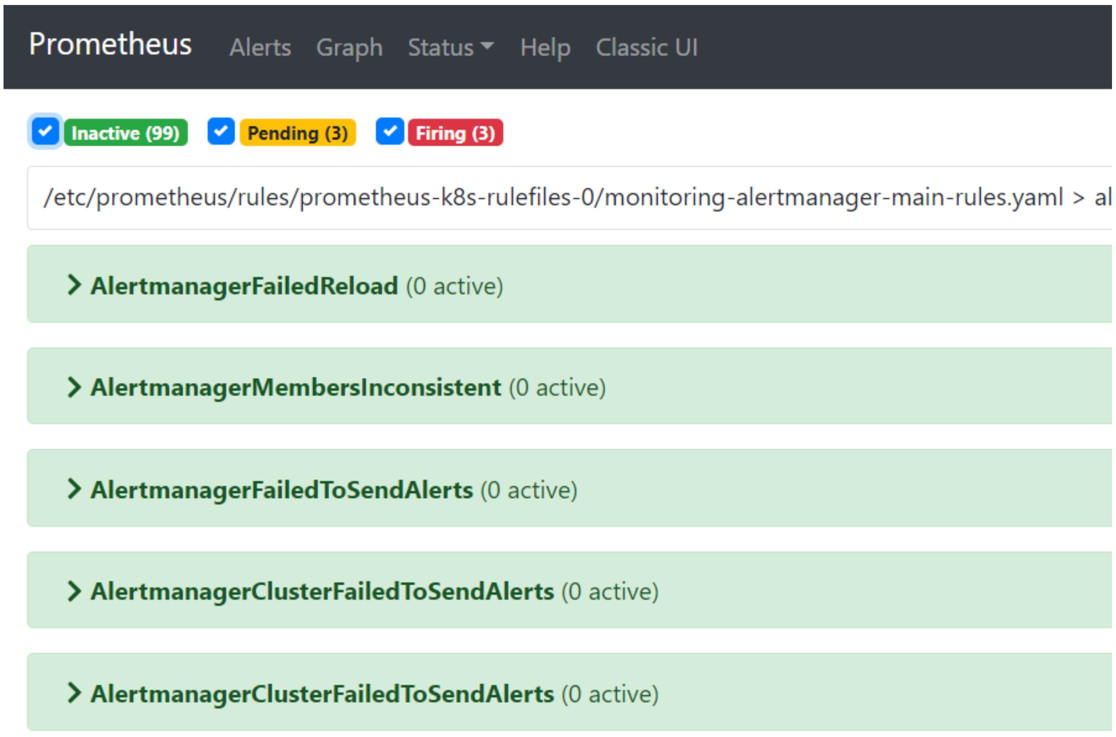
之后可以通过任意一个安装了 kube-proxy 服务的节点 IP+**31266** 端口即可访问到 Grafana：



Grafana 默认登录的账号密码为 admin/admin。然后相同的方式更改 Prometheus 的 Service 为 NodePort：

```
# kubectl  edit svc prometheus-k8s  -n monitoring
# kubectl get svc -n monitoring prometheus-k8s
NAME             TYPE       CLUSTER-IP        EXTERNAL-IP   PORT(S)
AGE
prometheus-k8s   NodePort   192.168.135.107   <none>        9090:31922/TCP
9m52s
```

通过 **31922** 端口即可访问到 Prometheus 的 Web UI：



✓ Inactive (99) ✓ Pending (3) ✓ Firing (3)

/etc/prometheus/rules/prometheus-k8s-rulefiles-0/monitoring-alertmanager-main-rules.yaml > al

> **AlertmanagerFailedReload** (0 active)

> **AlertmanagerMembersInconsistent** (0 active)

> **AlertmanagerFailedToSendAlerts** (0 active)

> **AlertmanagerClusterFailedToSendAlerts** (0 active)

> **AlertmanagerClusterFailedToSendAlerts** (0 active)

提示
默认安装完成后，会有几个告警，先忽略。

1.3 云原生和非云原生应用的监控流程

1.3.1 监控数据来源

目前比较常用的 Exporter 工具如下：

类型	Exporter
数据库	MySQL Exporter, Redis Exporter, MongoDB Exporter, MSSQL Exporter
硬件	Apcupsd Exporter，IoT Edison Exporter， IPMI Exporter, Node Exporter
消息队列	Beanstalkd Exporter, Kafka Exporter, NSQ Exporter, RabbitMQ Exporter
存储	Ceph Exporter, Gluster Exporter, HDFS Exporter, ScaleIO Exporter
HTTP 服务	Apache Exporter, HAProxy Exporter, Nginx Exporter
API 服务	AWS ECS Exporter， Docker Cloud Exporter, Docker Hub Exporter, GitHub Exporter
日志	Fluentd Exporter, Grok Exporter
监控系统	Collectd Exporter, Graphite Exporter, InfluxDB Exporter, Nagios Exporter, SNMP Exporter
其它	Blackbox Exporter, JIRA Exporter, Jenkins Exporter， Confluence Exporter

1.3.2 云原生应用 Etcd 监控

测试访问 Etcd Metrics 接口：

```
    # curl -s --cert /etc/kubernetes/pki/etcd/etcd.pem --key
/etc/kubernetes/pki/etcd/etcd-key.pem  https://YOUR_ETCD_IP:2379/metrics -k
| tail -1
    promhttp_metric_handler_requests_total{code="503"} 0
```

证书的位置可以在 Etcd 配置文件中获得（注意配置文件的位置，不同的集群位置可能不同，Kubeadm 安装方式可能会在/etc/kubernetes/manifests/etcd.yml 中）：

```
    # grep -E "key-file|cert-file" /etc/etcd/etcd.config.yml
      cert-file: '/etc/kubernetes/pki/etcd/etcd.pem'
      key-file: '/etc/kubernetes/pki/etcd/etcd-key.pem'
```

1.3.2.1 Etcd Service 创建

首先需要配置 Etcd 的 Service 和 Endpoint：

```
    # vim etcd-svc.yaml
    apiVersion: v1
    kind: Endpoints
    metadata:
      labels:
        app: etcd-prom
      name: etcd-prom
      namespace: kube-system
    subsets:
    - addresses:
      - ip: YOUR_ETCD_IP01
      - ip: YOUR_ETCD_IP02
      - ip: YOUR_ETCD_IP03
      ports:
      - name: https-metrics
        port: 2379   # etcd端口
        protocol: TCP
    ---
    apiVersion: v1
    kind: Service
    metadata:
      labels:
        app: etcd-prom
      name: etcd-prom
      namespace: kube-system
    spec:
      ports:
      - name: https-metrics
        port: 2379
        protocol: TCP
        targetPort: 2379
      type: ClusterIP
```

需要注意将 **YOUR_ETCD_IP** 改成自己的 Etcd 主机 IP，另外需要注意 port 的名称为 **https-metrics**，需要和后面的 ServiceMonitor 保持一致。之后创建该资源并查看 Service 的 ClusterIP：

```
    # kubectl create -f etcd-svc.yaml
    # kubectl get svc -n kube-system etcd-prom
    NAME            TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
    etcd-prom       ClusterIP   192.168.2.188    <none>        2379/TCP
8s
```

通过 ClusterIP 访问测试：

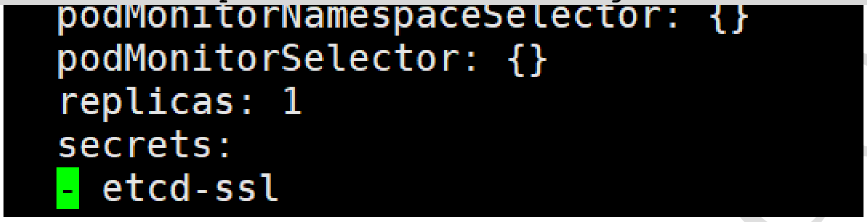
```
    # curl -s --cert /etc/kubernetes/pki/etcd/etcd.pem --key
/etc/kubernetes/pki/etcd/etcd-key.pem  https://192.168.2.188:2379/metrics -k
| tail -1
    promhttp_metric_handler_requests_total{code="503"} 0
```

创建 Etcd 证书的 Secret（证书路径根据实际环境进行更改）：

```
    # kubectl create secret generic etcd-ssl --from-
file=/etc/kubernetes/pki/etcd/etcd-ca.pem --from-
file=/etc/kubernetes/pki/etcd/etcd.pem --from-
file=/etc/kubernetes/pki/etcd/etcd-key.pem -n monitoring
    secret/etcd-ssl created
```

将证书挂载至 Prometheus 容器（由于 Prometheus 是 Operator 部署的，所以只需要修改 Prometheus 资源即可）：

```
    # kubectl edit prometheus k8s -n monitoring
```



```
  podMonitorNamespaceSelector: {}
  podMonitorSelector: {}
  replicas: 1
  secrets:
  - etcd-ssl
```

保存退出后，Prometheus 的 Pod 会自动重启，重启完成后，查看证书是否挂载（任意一个 Prometheus 的 Pod 均可）：

```
    # kubectl get po -n monitoring -l app=prometheus
    NAME               READY   STATUS    RESTARTS   AGE
    prometheus-k8s-0   4/4     Running   1          29s
    #  kubectl exec  -n monitoring prometheus-k8s-0 -c prometheus -- ls
/etc/prometheus/secrets/etcd-ssl/
    etcd-ca.pem
    etcd-key.pem
    etcd.pem
```

1.3.2.2 Etcd ServiceMonitor 创建

之后创建 Etcd 的 ServiceMonitor：

```
    # cat servicemonitor.yaml
    apiVersion: monitoring.coreos.com/v1
    kind: ServiceMonitor
    metadata:
      name: etcd
      namespace: monitoring
      labels:
        app: etcd
    spec:
      jobLabel: k8s-app
      endpoints:
        - interval: 30s
          port: https-metrics  # 这个 port 对应 Service.spec.ports.name
          scheme: https
          tlsConfig:
            caFile: /etc/prometheus/secrets/etcd-ssl/etcd-ca.pem #证书路径
            certFile: /etc/prometheus/secrets/etcd-ssl/etcd.pem
            keyFile: /etc/prometheus/secrets/etcd-ssl/etcd-key.pem
            insecureSkipVerify: true  # 关闭证书校验
      selector:
        matchLabels:
```

```
      app: etcd-prom  # 跟 svc 的 lables 保持一致
  namespaceSelector:
    matchNames:
    - kube-system
```

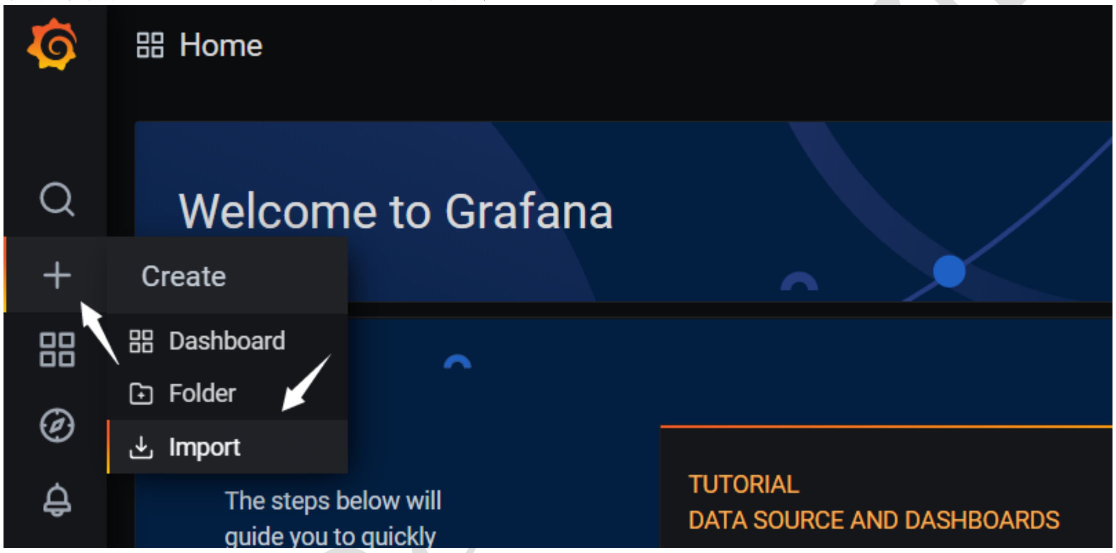
和之前的 ServiceMonitor 相比，多了 tlsConfig 的配置，http 协议的 Metrics 无需该配置。创建该 ServiceMonitor：

```
# kubectl create -f servicemonitor.yaml
servicemonitor.monitoring.coreos.com/etcd created
```

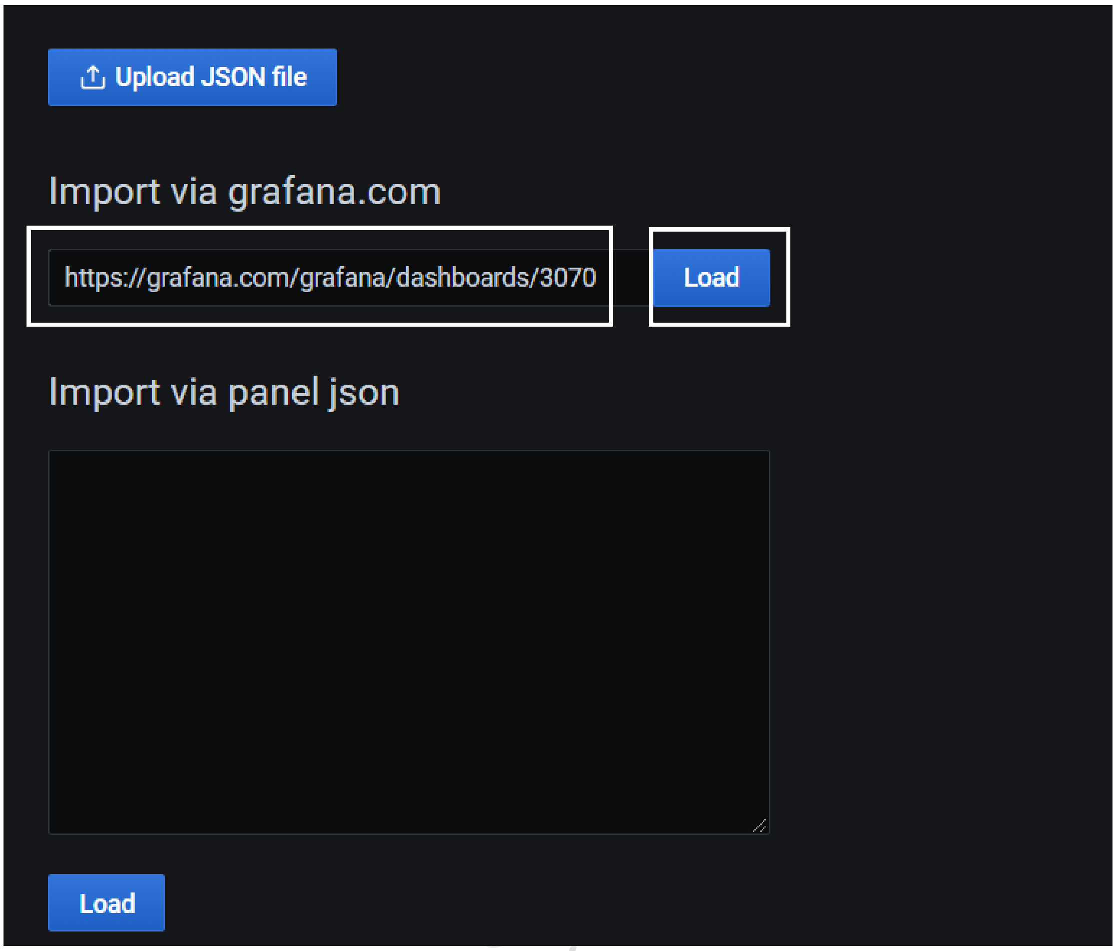
创建完成后，在 Prometheus 的 Web UI 即可看到相关配置，在此不再演示。

1.3.2.3 Grafana 配置

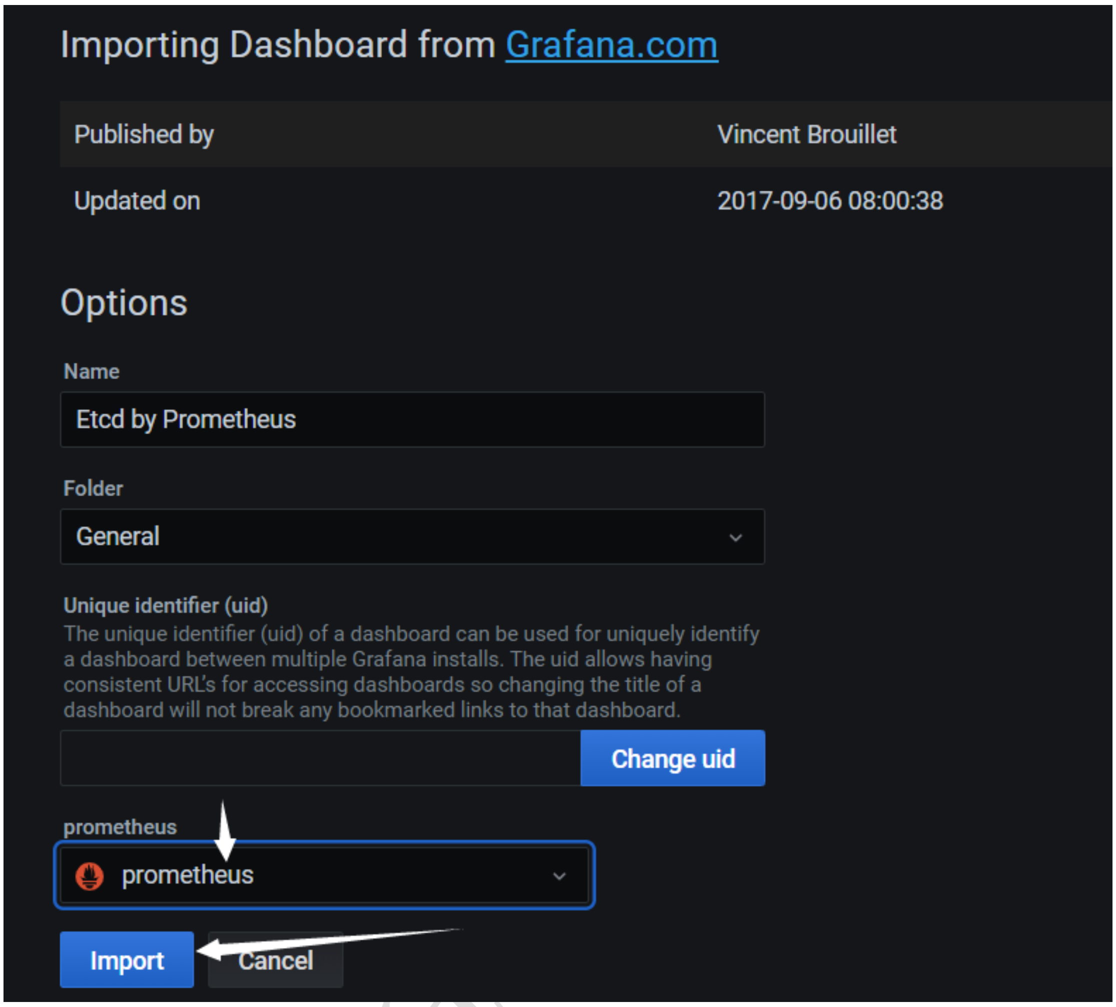
接下来打开 Grafana，添加 Etcd 的监控面板：


Home
Welcome to Grafana
Create
Dashboard
Folder
Import
The steps below will guide you to quickly
TUTORIAL
DATA SOURCE AND DASHBOARDS

依次点击"+"号 → Import，之后输入 Etcd 的 Grafana Dashboard 地址 https://grafana.com/grafana/dashboards/3070，如下图所示：



点击 Load，然后选择 Prometheus，点击 Import 即可：


Importing Dashboard from Grafana.com

Published by	Vincent Brouillet
Updated on	2017-09-06 08:00:38

Options

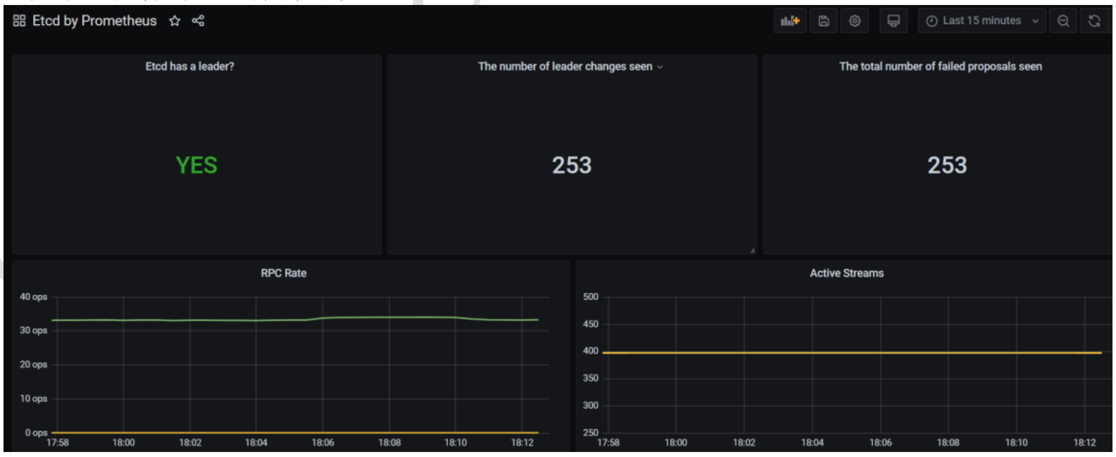
Name: Etcd by Prometheus

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 URL's for accessing dashboards so changing the title of a dashboard will not break any bookmarked links to that dashboard.

prometheus: prometheus

之后就可以看到 Etcd 集群的状态：


Etcd by Prometheus

Etcd has a leader? YES

The number of leader changes seen: 253

The total number of failed proposals seen: 253

1.3.3 非云原生监控 Exporter

本节将使用 MySQL 作为一个测试用例，演示如何使用 Exporter 监控非云原生应用。

1.3.3.1 部署测试用例

首先部署 MySQL 至 Kubernetes 集群中，直接配置 MySQL 的权限即可：

```
    # kubectl create deploy mysql --image=registry.cn-
beijing.aliyuncs.com/dotbalo/mysql:5.7.23
    deployment.apps/mysql created
    # 设置密码
    # kubectl  set env deploy/mysql  MYSQL_ROOT_PASSWORD=mysql
    deployment.apps/mysql env updated
    # 查看 Pod 是否正常
    # kubectl get po  -l app=mysql
    NAME                     READY   STATUS    RESTARTS   AGE
    mysql-69d6f69557-5vxvg   1/1     Running   0          47s
```

创建 Service 暴露 MySQL：

```
    # kubectl expose deploy mysql --port 3306
    service/mysql exposed
    # kubectl get svc -l app=mysql
    NAME    TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
    mysql   ClusterIP   192.168.140.81   <none>        3306/TCP   29s
```

检查 Service 是否可用：

```
    # telnet 192.168.140.81 3306
    Trying 192.168.140.81...
    Connected to 192.168.140.81.
    Escape character is '^]'.
    J
    ;FuNunhZmysql_native_password^CConnection closed by foreign host.
```

登录 MySQL，创建 Exporter 所需的用户和权限（如果已经有需要监控的 MySQL，可以直接执行此步骤即可）：

```
    # kubectl exec -ti mysql-69d6f69557-5vnvg  -- bash
    root@mysql-69d6f69557-5vnvg:/# mysql -uroot -pmysql
    mysql: [Warning] Using a password on the command line interface can be
insecure.
    Welcome to the MySQL monitor.  Commands end with ; or \g.
    Your MySQL connection id is 3
    Server version: 5.7.23 MySQL Community Server (GPL)

    Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights
reserved.

    Oracle is a registered trademark of Oracle Corporation and/or its
    affiliates. Other names may be trademarks of their respective
    owners.

    Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

    mysql> CREATE USER 'exporter'@'%' IDENTIFIED BY 'exporter' WITH
MAX_USER_CONNECTIONS 3;
    Query OK, 0 rows affected (0.01 sec)

    mysql> GRANT PROCESS, REPLICATION CLIENT, SELECT ON *.* TO
'exporter'@'%';
    Query OK, 0 rows affected (0.00 sec)

    mysql> quit
    Bye
    root@mysql-69d6f69557-5vnvg:/# exit
    exit
```

配置 MySQL Exporter 采集 MySQL 监控数据：

```
# cat mysql-exporter.yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-exporter
  namespace: monitoring
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: mysql-exporter
  template:
    metadata:
      labels:
        k8s-app: mysql-exporter
    spec:
      containers:
      - name: mysql-exporter
        image: registry.cn-beijing.aliyuncs.com/dotbalo/mysqld-exporter
        env:
         - name: DATA_SOURCE_NAME
           value: "exporter:exporter@(mysql.default:3306)/"
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9104
---
apiVersion: v1
kind: Service
metadata:
  name: mysql-exporter
  namespace: monitoring
  labels:
    k8s-app: mysql-exporter
spec:
  type: ClusterIP
  selector:
    k8s-app: mysql-exporter
  ports:
  - name: api
    port: 9104
    protocol: TCP
```

注意 DATA_SOURCE_NAME 的配置，需要将 **exporter:exporter@(mysql.default:3306)/**改成自己的实际配置，格式如下 USERNAME:PASSWORD@MYSQL_HOST_ADDRESS:MYSQL_PORT。

创建 Exporter：

```
# kubectl create -f mysql-exporter.yaml
deployment.apps/mysql-exporter created
service/mysql-exporter created
```

```
# kubectl get -f mysql-exporter.yaml
NAME                             READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mysql-exporter   1/1     1            1           39s

NAME                     TYPE        CLUSTER-IP        EXTERNAL-IP   PORT(S)
AGE
service/mysql-exporter   ClusterIP   192.168.150.122   <none>
9104/TCP   39s
```

通过该 Service 地址，检查是否能正常获取 Metrics 数据：

```
# curl 192.168.150.122:9104/metrics | tail -1
promhttp_metric_handler_requests_total{code="503"} 0
```

1.3.3.2 ServiceMonitor 和 Grafana 配置

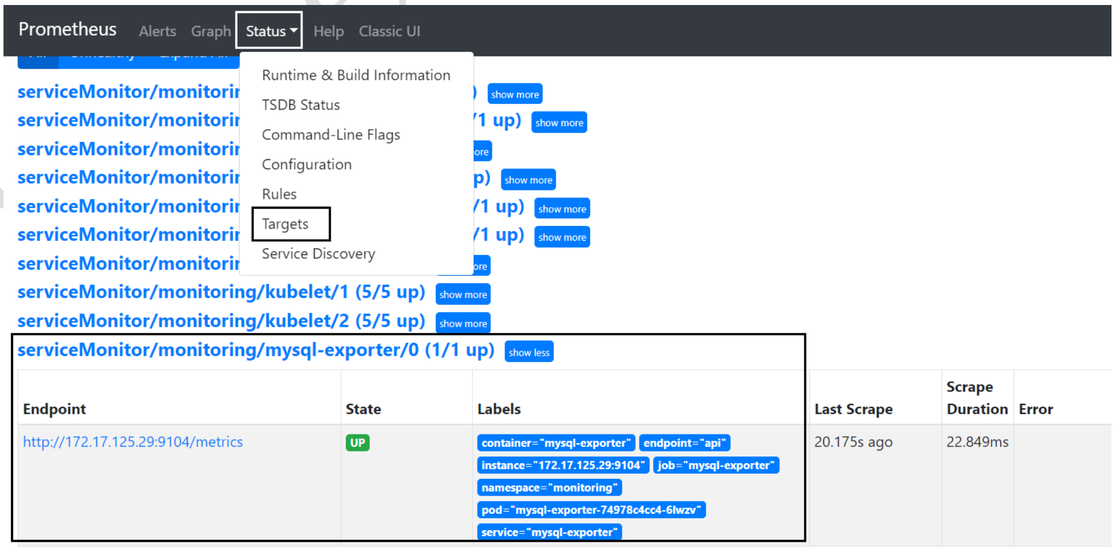
配置 ServiceMonitor：

```
# cat mysql-sm.yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: mysql-exporter
  namespace: monitoring
  labels:
    k8s-app: mysql-exporter
    namespace: monitoring
spec:
  jobLabel: k8s-app
  endpoints:
  - port: api
    interval: 30s
    scheme: http
  selector:
    matchLabels:
      k8s-app: mysql-exporter
  namespaceSelector:
    matchNames:
    - monitoring
```

需要注意 matchLabels 和 endpoints 的配置，要和 MySQL 的 Service 一致。之后创建该 ServiceMonitor：

```
# kubectl create -f mysql-sm.yaml
servicemonitor.monitoring.coreos.com/mysql-exporter created
```

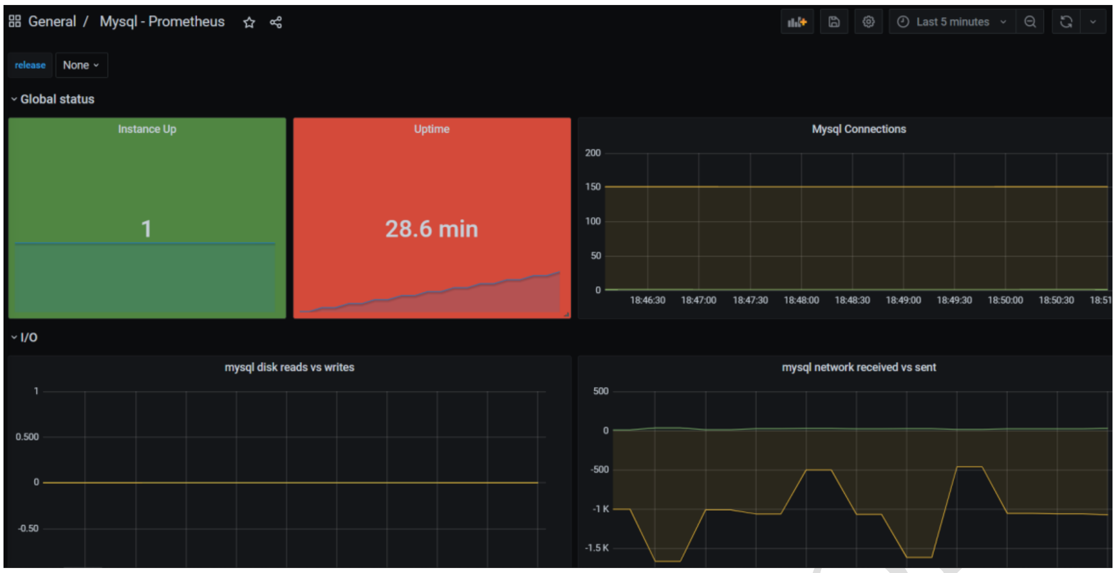
接下来即可在 Prometheus Web UI 看到该监控：



serviceMonitor/monitoring/kubelet/1 (5/5 up)
serviceMonitor/monitoring/kubelet/2 (5/5 up)
serviceMonitor/monitoring/mysql-exporter/0 (1/1 up)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://172.17.125.29:9104/metrics	UP	container="mysql-exporter" endpoint="api" instance="172.17.125.29:9104" job="mysql-exporter" namespace="monitoring" pod="mysql-exporter-74978c4cc4-6lwzv" service="mysql-exporter"	20.175s ago	22.849ms	

导入 Grafana Dashboard，地址：https://grafana.com/grafana/dashboards/6239，导入步骤和之前类似，在此不再演示。导入完成后，即可在 Grafana 看到监控数据：


General / Mysql - Prometheus
Last 5 minutes
release
None
Global status
Instance Up
1
Uptime
28.6 min
Mysql Connections
I/O
mysql disk reads vs writes
mysql network received vs sent

1.3.4 Service Monitor 找不到监控主机排查

```
    # kubectl get servicemonitor -n monitoring kube-controller-manager kube-
scheduler
    NAME                      AGE
    kube-controller-manager   39h
    kube-scheduler            39h


    # kubectl get servicemonitor -n monitoring kube-controller-manager -
oyaml
    apiVersion: monitoring.coreos.com/v1
    kind: ServiceMonitor
    ...
        port: https-metrics
        scheme: https
        tlsConfig:
          insecureSkipVerify: true
      jobLabel: app.kubernetes.io/name
      namespaceSelector:
        matchNames:
        - kube-system
      selector:
        matchLabels:
          app.kubernetes.io/name: kube-controller-manager
```

该 Service Monitor 匹配的是 kube-system 命名空间下，具有 app.kubernetes.io/name=kube-controller-manager 标签，接下来通过该标签查看是否有该 Service：

```
    # kubectl get svc -n kube-system -l app.kubernetes.io/name=kube-
controller-manager
    No resources found in kube-system namespace.
```

可以看到并没有此标签的 Service，所以导致了找不到需要监控的目标，此时可以手动创建该 Service 和 Endpoint 指向自己的 Controller Manager：

```
    apiVersion: v1
    kind: Endpoints
    metadata:
      labels:
        app.kubernetes.io/name: kube-controller-manager
      name: kube-controller-manager-prom
      namespace: kube-system
    subsets:
    - addresses:
      - ip: YOUR_CONTROLLER_IP01
      - ip: YOUR_CONTROLLER_IP02
      - ip: YOUR_CONTROLLER_IP03
      ports:
      - name: http-metrics
        port: 10252
        protocol: TCP
    ---
    apiVersion: v1
    kind: Service
    metadata:
      labels:
        app.kubernetes.io/name: kube-controller-manager
      name: kube-controller-manager-prom
      namespace: kube-system
    spec:
      ports:
      - name: http-metrics
        port: 10252
        protocol: TCP
        targetPort: 10252
      sessionAffinity: None
      type: ClusterIP
```

注意需要更改 Endpoint 配置的 **YOUR_CONTROLLER_IP** 为自己的 Controller Manager 的 IP，接下来创建该 Service 和 Endpoint：

```
    # kubectl create -f controller.yaml
    endpoints/kube-controller-manager-prom created
    service/kube-controller-manager-prom created
```

查看创建的 Service 和 Endpoint：

```
    # kubectl get svc -n kube-system kube-controller-manager-prom
    NAME                         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
    kube-controller-manager-prom   ClusterIP   192.168.213.1   <none>
10252/TCP   34s
```

此时该 Service 可能是不通的，因为在集群搭建时，可能 Controller Manager 和 Scheduler 是监听的 127.0.0.1 就导致无法被外部访问，此时需要更改它的监听地址为 0.0.0.0：

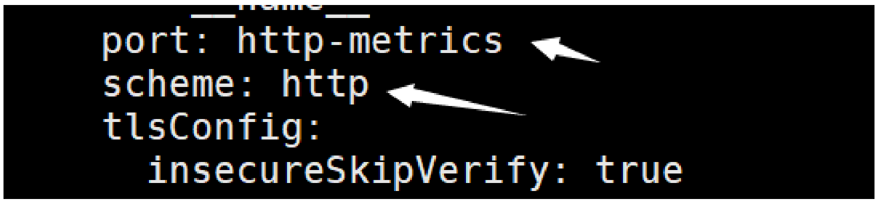
```
    # sed -i "s#address=127.0.0.1#address=0.0.0.0#g"
/usr/lib/systemd/system/kube-controller-manager.service
    # systemctl daemon-reload
    # systemctl restart kube-controller-manager
```

通过该 Service 的 ClusterIP 访问 Controller Manager 的 Metrics 接口：

```
    # kubectl get svc -n kube-system kube-controller-manager-prom
    NAME                         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
    kube-controller-manager-prom   ClusterIP   192.168.213.1   <none>
10252/TCP   5m59s
    # curl -s 192.168.213.1:10252/metrics | tail -1
    workqueue_work_duration_seconds_count{name="DynamicServingCertificateCon
troller"} 3
```

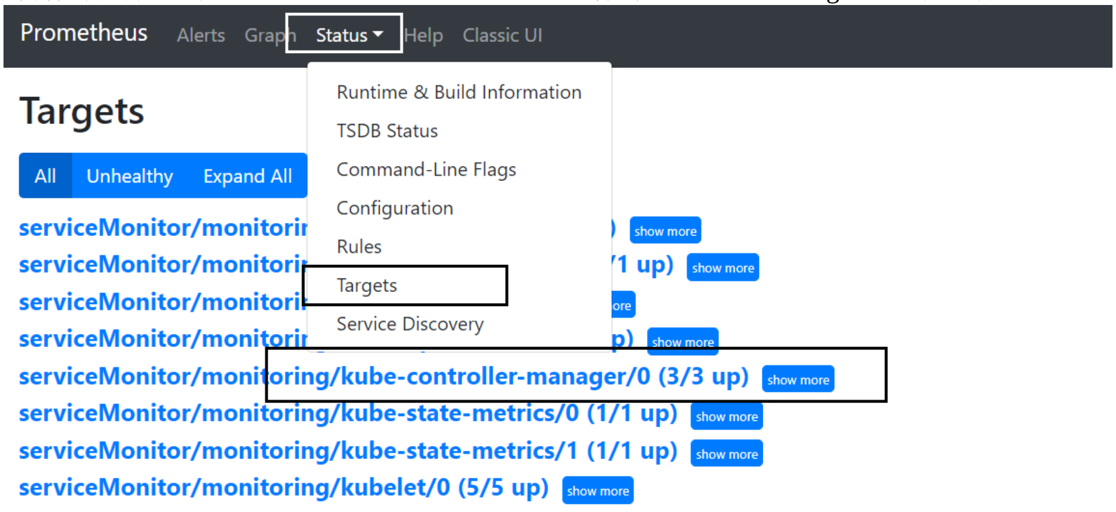
更改 ServiceMonitor 的配置和 Service 一致：

```
    # kubectl edit servicemonitor kube-controller-manager -n monitoring
```



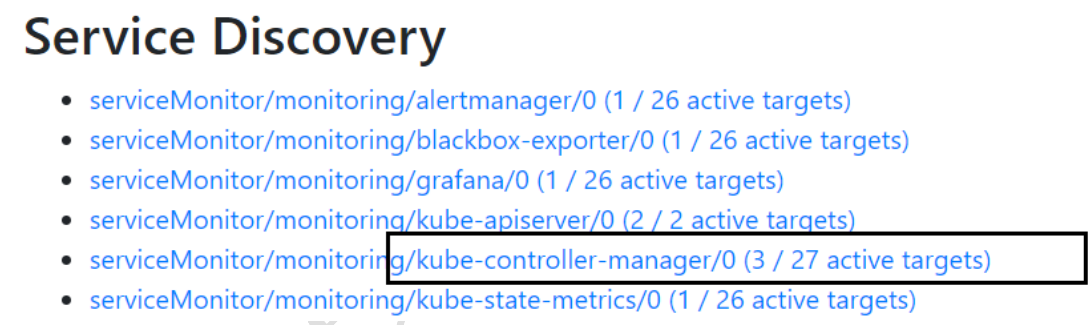
```
    port: http-metrics
    scheme: http
    tlsConfig:
      insecureSkipVerify: true
```

等待几分钟后，就可以在 Prometheus 的 Web UI 上看到 Controller Manager 的监控目标：



Targets

serviceMonitor/monitoring/kube-controller-manager/0 (3/3 up)
serviceMonitor/monitoring/kube-state-metrics/0 (1/1 up)
serviceMonitor/monitoring/kube-state-metrics/1 (1/1 up)
serviceMonitor/monitoring/kubelet/0 (5/5 up)



Service Discovery

- serviceMonitor/monitoring/alertmanager/0 (1 / 26 active targets)
- serviceMonitor/monitoring/blackbox-exporter/0 (1 / 26 active targets)
- serviceMonitor/monitoring/grafana/0 (1 / 26 active targets)
- serviceMonitor/monitoring/kube-apiserver/0 (2 / 2 active targets)
- serviceMonitor/monitoring/kube-controller-manager/0 (3 / 27 active targets)
- serviceMonitor/monitoring/kube-state-metrics/0 (1 / 26 active targets)

通过 Service Monitor 监控应用时，如果监控没有找到目标主机的排查步骤时，排查步骤大致如下：

- ➢ 确认 Service Monitor 是否成功创建
- ➢ 确认 Prometheus 是否生成了相关配置
- ➢ 确认存在 Service Monitor 匹配的 Service
- ➢ 确认通过 Service 能够访问程序的 Metrics 接口
- ➢ 确认 Service 的端口和 Scheme 和 Service Monitor 一致

1.4 黑盒监控

新版 Prometheus Stack 已经默认安装了 BlackboxExporter，可以通过以下命令查看：

```
    # kubectl get po -n monitoring -l app.kubernetes.io/name=blackbox-
exporter
    NAME                                READY   STATUS    RESTARTS   AGE
    blackbox-exporter-7f88596689-fl2v8   3/3     Running   0          8d
```

同时也会创建一个 Service，可以通过该 Service 访问 Blackbox Exporter 并传递一些参数：

```
    # kubectl get svc -n monitoring -l app.kubernetes.io/name=blackbox-
exporter
    NAME              TYPE         CLUSTER-IP        EXTERNAL-IP    PORT(S)
```

```
AGE
    blackbox-exporter   ClusterIP   192.168.204.117   <none>
9115/TCP,19115/TCP   8d
```

比如检测下 **gaoxin.kubeasy.com**（使用任何一个公网域名或者公司内的域名探测即可）网站的状态，可以通过如下命令进行检查：

```
    # curl -s
"http://192.168.204.117:19115/probe?target=gaoxin.kubeasy.com&module=http_2x
x" | tail -1
    probe_tls_version_info{version="TLS 1.2"} 1
```

probe 是接口地址，target 是检测的目标，module 是使用哪个模块进行探测。

如果集群中没有配置 Blackbox Exporter，可以参考 https://github.com/prometheus/blackbox_exporter 进行安装。

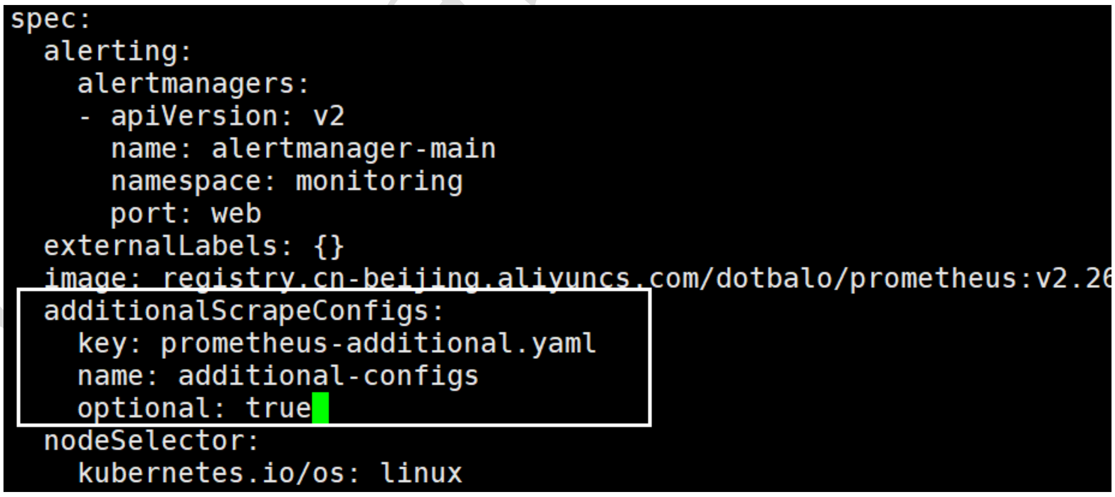
1.5 Prometheus 静态配置

首先创建一个空文件，然后通过该文件创建一个 Secret，那么这个 Secret 即可作为 Prometheus 的静态配置：

```
    # touch prometheus-additional.yaml
    # kubectl create secret generic additional-configs --from-
file=prometheus-additional.yaml -n monitoring
    secret/additional-configs created
```

创建完 Secret 后，需要编辑下 Prometheus 配置：

```
    # kubectl edit prometheus -n monitoring k8s

      additionalScrapeConfigs:
        key: prometheus-additional.yaml
        name: additional-configs
        optional: true
```



```
spec:
  alerting:
    alertmanagers:
    - apiVersion: v2
      name: alertmanager-main
      namespace: monitoring
      port: web
  externalLabels: {}
  image: registry.cn-beijing.aliyuncs.com/dotbalo/prometheus:v2.26
  additionalScrapeConfigs:
    key: prometheus-additional.yaml
    name: additional-configs
    optional: true
  nodeSelector:
    kubernetes.io/os: linux
```

添加上述配置后保存退出，无需重启 Prometheus 的 Pod 即可生效。之后在 prometheus-additional.yaml 文件内编辑一些静态配置，此处用黑盒监控的配置进行演示：

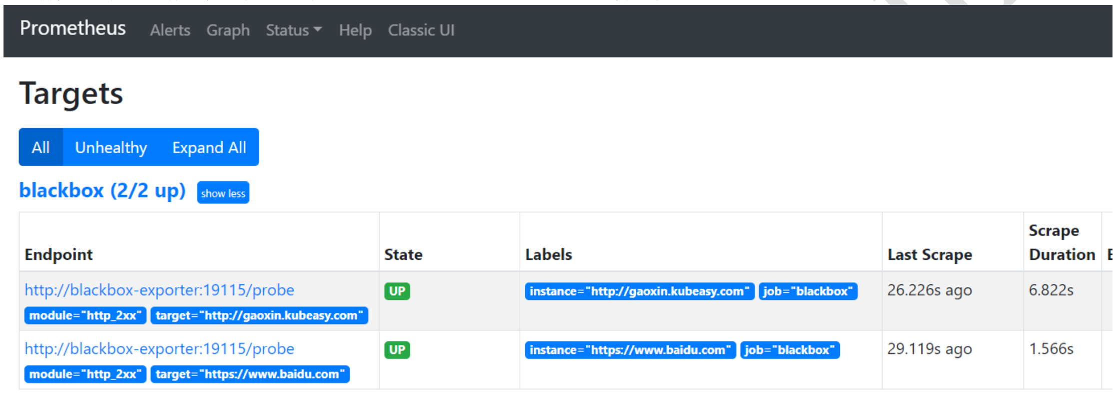
```
    - job_name: 'blackbox'
      metrics_path: /probe
      params:
        module: [http_2xx]  # Look for a HTTP 200 response.
      static_configs:
        - targets:
          - http://gaoxin.kubeasy.com    # Target to probe with http.
          - https://www.baidu.com   # Target to probe with https.
      relabel_configs:
        - source_labels: [__address__]
          target_label: __param_target
        - source_labels: [__param_target]
          target_label: instance
        - target_label: __address__
          replacement: blackbox-exporter:19115  # The blackbox exporter's real
  hostname:port.
```

- targets：探测的目标，根据实际情况进行更改
- params：使用哪个模块进行探测
- replacement：Blackbox Exporter 的地址

可以看到此处的内容，和传统配置的内容一致，只需要添加对应的 job 即可。之后通过该文件更新该 Secret：

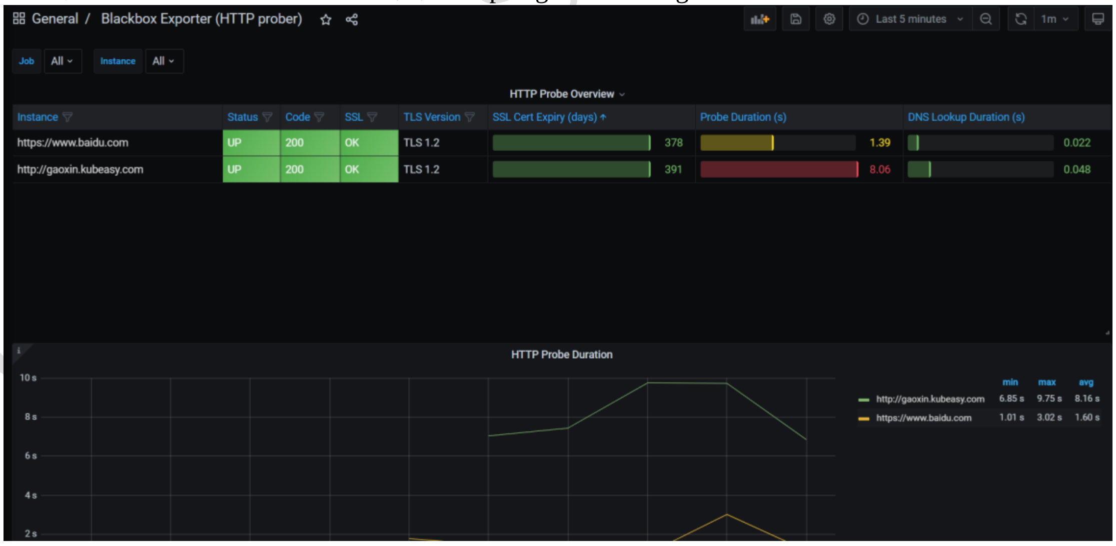
```
    # kubectl create secret generic additional-configs --from-
file=prometheus-additional.yaml --dry-run=client -oyaml | kubectl replace -f
- -n monitoring
```

更新完成后，稍等一分钟即可在 Prometheus Web UI 看到该配置：



Endpoint	State	Labels	Last Scrape	Scrape Duration
http://blackbox-exporter:19115/probe module="http_2xx" target="http://gaoxin.kubeasy.com"	UP	instance="http://gaoxin.kubeasy.com" job="blackbox"	26.226s ago	6.822s
http://blackbox-exporter:19115/probe module="http_2xx" target="https://www.baidu.com"	UP	instance="https://www.baidu.com" job="blackbox"	29.119s ago	1.566s

监控状态 UP 后，导入黑盒监控的模板（https://grafana.com/grafana/dashboards/13659）即可：



Instance	Status	Code	SSL	TLS Version	SSL Cert Expiry (days)	Probe Duration (s)	DNS Lookup Duration (s)
https://www.baidu.com	UP	200	OK	TLS 1.2	378	1.39	0.022
http://gaoxin.kubeasy.com	UP	200	OK	TLS 1.2	391	8.06	0.048

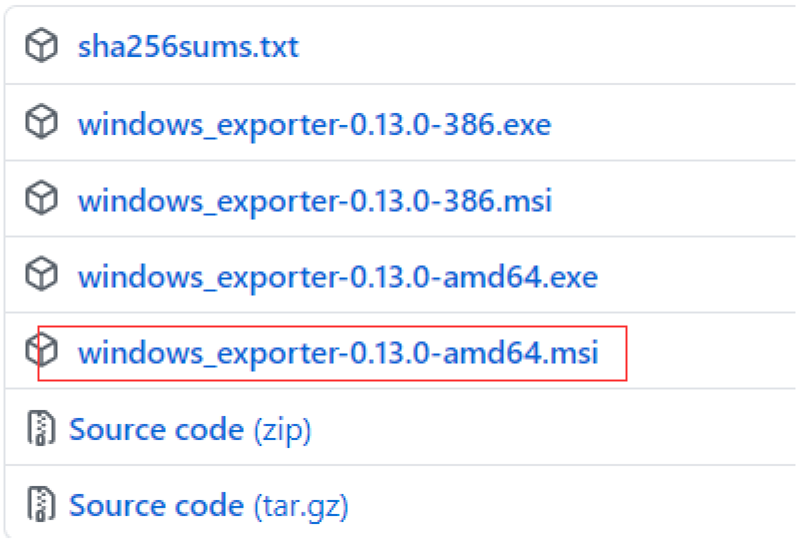
	min	max	avg
http://gaoxin.kubeasy.com	6.85 s	9.75 s	8.16 s
https://www.baidu.com	1.01 s	3.02 s	1.60 s

其他模块使用方法类似，可以参考：https://github.com/prometheus/blackbox_exporter

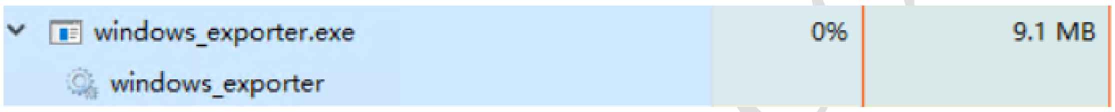
1.6 Prometheus 监控 Windows（外部）主机

监控 Linux 的 Exporter 是：https://github.com/prometheus/node_exporter，监控 Windows 主机的 Exporter 是：https://github.com/prometheus-community/windows_exporter。

首先下载对应的 Exporter 至 Windows 主机（MSI 文件下载地址：https://github.com/prometheus-community/windows_exporter/releases）：


sha256sums.txt
windows_exporter-0.13.0-386.exe
windows_exporter-0.13.0-386.msi
windows_exporter-0.13.0-amd64.exe
windows_exporter-0.13.0-amd64.msi
Source code (zip)
Source code (tar.gz)

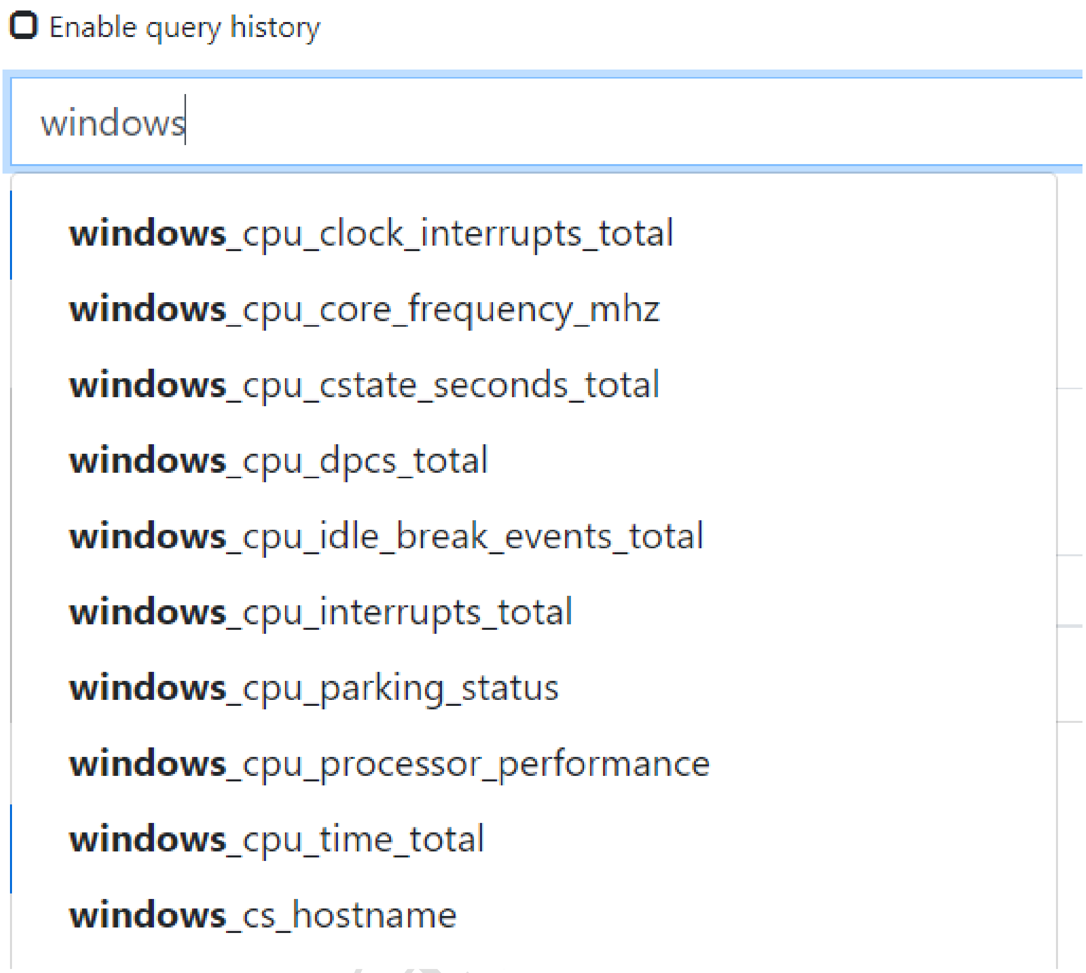
下载完成后，双击打开即可完成安装，之后可以在任务管理器上看到对应的进程：


windows_exporter.exe 0% 9.1 MB
windows_exporter

Windows Exporter 会暴露一个 9182 端口，可以通过该端口访问到 Windows 的监控数据。接下来在静态配置文件中添加以下配置：

```
  - job_name: 'WindowsServerMonitor'
    static_configs:
      - targets:
        - "1.1.1.1:9182"
        labels:
          server_type: 'windows'
    relabel_configs:
      - source_labels: [__address__]
        target_label: instance
```

Targets 配置的是监控主机，如果是多台 Windows 主机，配置多行即可，当然每台主机都需要配置 Exporter。之后可以在 Prometheus Web UI 看到监控数据：

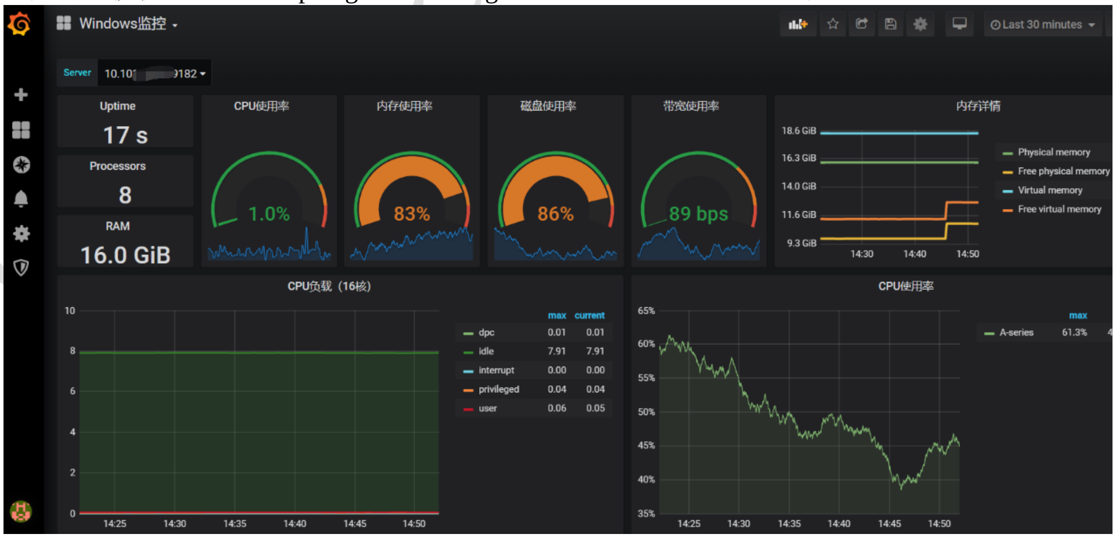


Enable query history

windows

windows_cpu_clock_interrupts_total
windows_cpu_core_frequency_mhz
windows_cpu_cstate_seconds_total
windows_cpu_dpcs_total
windows_cpu_idle_break_events_total
windows_cpu_interrupts_total
windows_cpu_parking_status
windows_cpu_processor_performance
windows_cpu_time_total
windows_cs_hostname

之后导入模板（地址：https://grafana.com/grafana/dashboards/12566）即可：

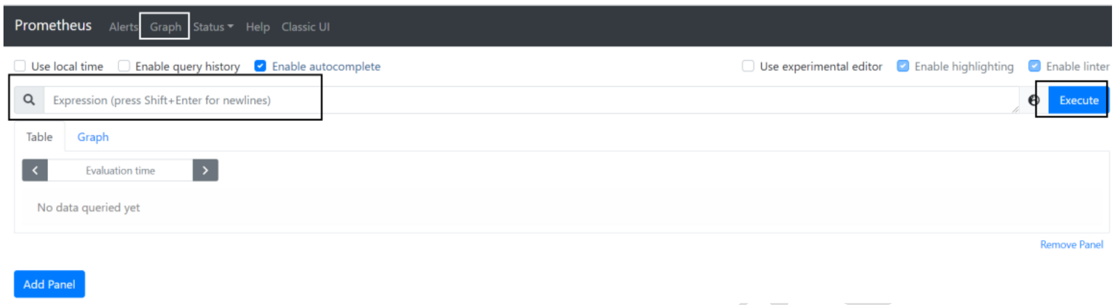


	max	current
dpc	0.01	0.01
idle	7.91	7.91
interrupt	0.00	0.00
privileged	0.04	0.04
user	0.06	0.05

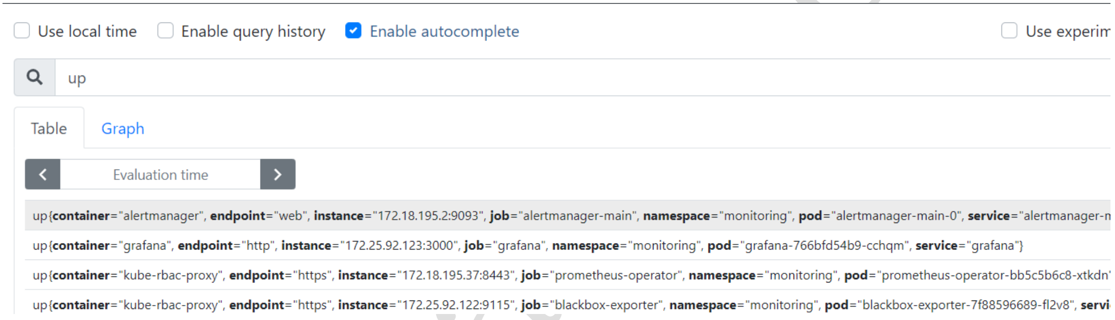
1.7 Prometheus 语法 PromQL 入门

1.7.1 PromQL 语法初体验

PromQL Web UI 的 Graph 选项卡提供了简单的用于查询数据的入口，对于 PromQL 的编写和校验都可以在此位置，如图所示：

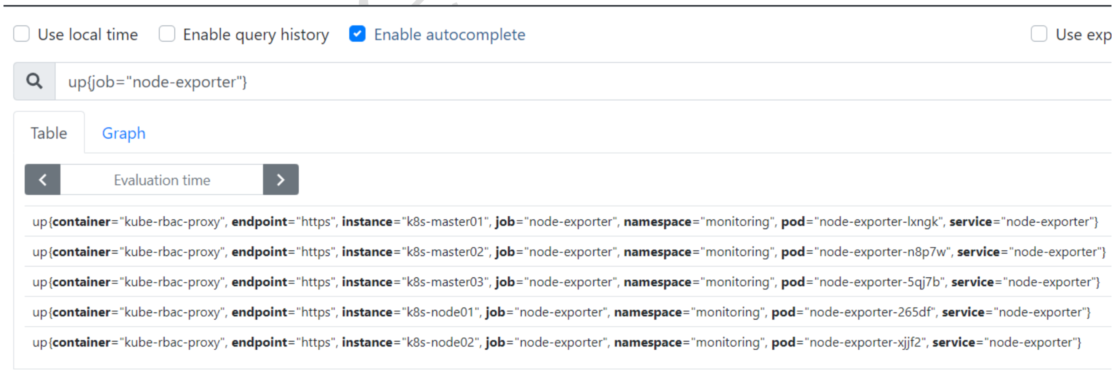


输入 up，然后点击 Execute，就能查到监控正常的 Target：



```
up{container="alertmanager", endpoint="web", instance="172.18.195.2:9093", job="alertmanager-main", namespace="monitoring", pod="alertmanager-main-0", service="alertmanager-n
up{container="grafana", endpoint="http", instance="172.25.92.123:3000", job="grafana", namespace="monitoring", pod="grafana-766bfd54b9-cchqm", service="grafana"}
up{container="kube-rbac-proxy", endpoint="https", instance="172.18.195.37:8443", job="prometheus-operator", namespace="monitoring", pod="prometheus-operator-bb5c5b6c8-xtkdn"
up{container="kube-rbac-proxy", endpoint="https", instance="172.25.92.122:9115", job="blackbox-exporter", namespace="monitoring", pod="blackbox-exporter-7f88596689-fl2v8", servi
```

通过标签选择器过滤出 job 为 node-exporter 的监控，语法为：`up{job="node-exporter"}`：

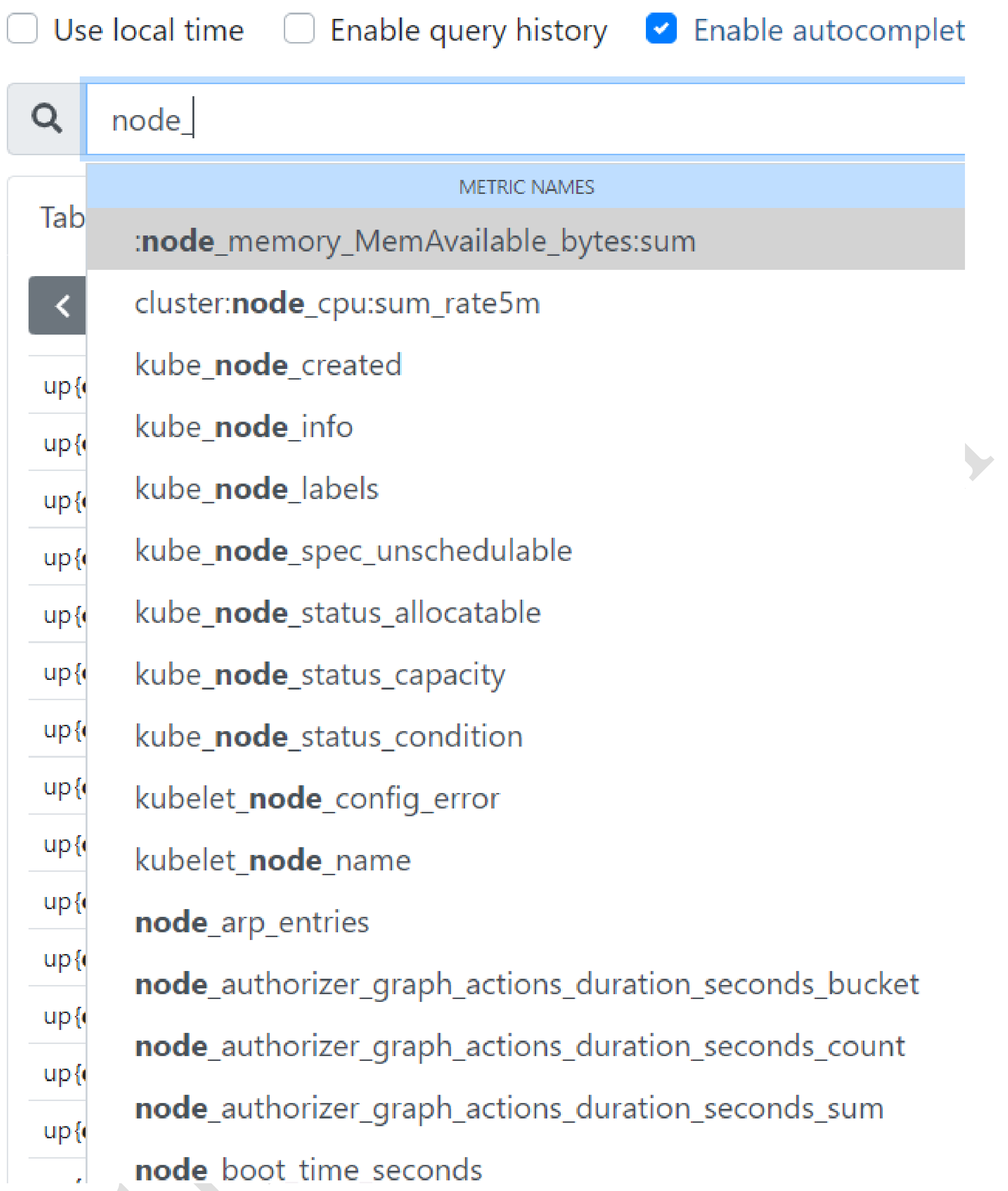


```
up{container="kube-rbac-proxy", endpoint="https", instance="k8s-master01", job="node-exporter", namespace="monitoring", pod="node-exporter-lxngk", service="node-exporter"}
up{container="kube-rbac-proxy", endpoint="https", instance="k8s-master02", job="node-exporter", namespace="monitoring", pod="node-exporter-n8p7w", service="node-exporter"}
up{container="kube-rbac-proxy", endpoint="https", instance="k8s-master03", job="node-exporter", namespace="monitoring", pod="node-exporter-5qj7b", service="node-exporter"}
up{container="kube-rbac-proxy", endpoint="https", instance="k8s-node01", job="node-exporter", namespace="monitoring", pod="node-exporter-265df", service="node-exporter"}
up{container="kube-rbac-proxy", endpoint="https", instance="k8s-node02", job="node-exporter", namespace="monitoring", pod="node-exporter-xjjf2", service="node-exporter"}
```

注意此时是 `up{job="node-exporter"}`属于绝对匹配，PromQL 也支持如下表达式：

- !=：不等于；
- =~：表示等于符合正则表达式的指标；
- !~：和=~类似，=~表示正则匹配，!~表示正则不匹配。

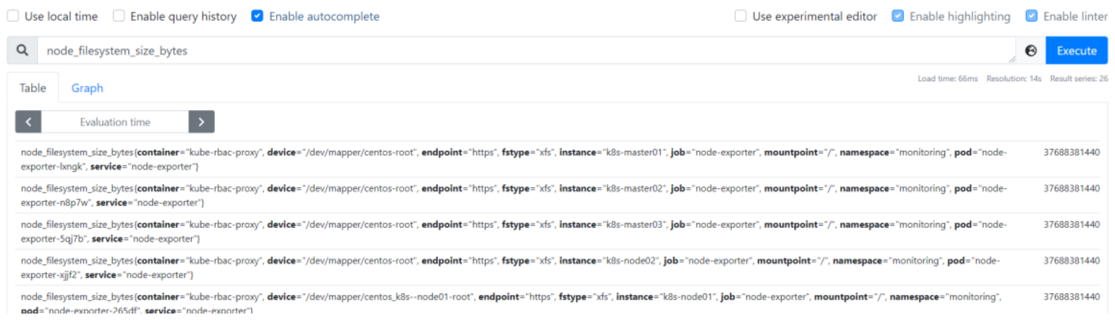
如果想要查看主机监控的指标有哪些，可以输入 node，会提示所有主机监控的指标：



```
node_
```

METRIC NAMES
:**node**_memory_MemAvailable_bytes:sum
cluster:**node**_cpu:sum_rate5m
kube_**node**_created
kube_**node**_info
kube_**node**_labels
kube_**node**_spec_unschedulable
kube_**node**_status_allocatable
kube_**node**_status_capacity
kube_**node**_status_condition
kubelet_**node**_config_error
kubelet_**node**_name
node_arp_entries
node_authorizer_graph_actions_duration_seconds_bucket
node_authorizer_graph_actions_duration_seconds_count
node_authorizer_graph_actions_duration_seconds_sum
node_boot_time_seconds

假 如 想 要 查 询 Kubernetes 集 群 中 每 个 宿 主 机 的 磁 盘 总 量 ， 可 以 使 用 node_filesystem_size_bytes：

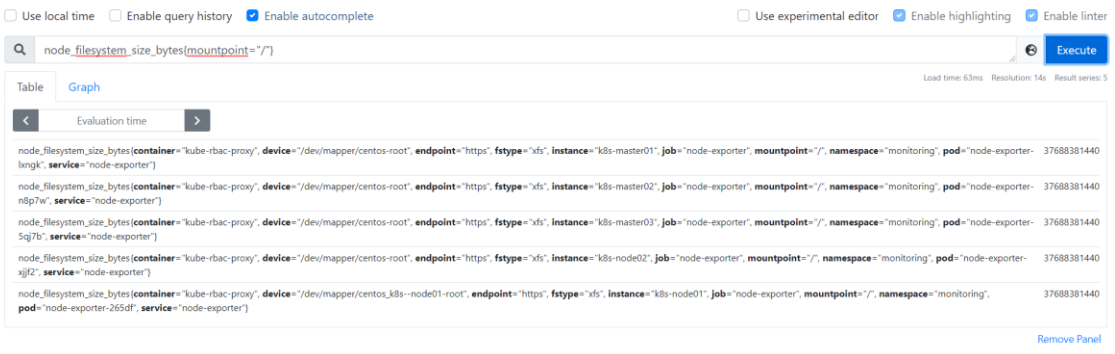


```
node_filesystem_size_bytes
```

Load time: 66ms   Resolution: 14s   Result series: 26

```
node_filesystem_size_bytes{container="kube-rbac-proxy", device="/dev/mapper/centos-root", endpoint="https", fstype="xfs", instance="k8s-master01", job="node-exporter", mountpoint="/", namespace="monitoring", pod="node-exporter-lxngk", service="node-exporter"}    37688381440
node_filesystem_size_bytes{container="kube-rbac-proxy", device="/dev/mapper/centos-root", endpoint="https", fstype="xfs", instance="k8s-master02", job="node-exporter", mountpoint="/", namespace="monitoring", pod="node-exporter-n8p7w", service="node-exporter"}    37688381440
node_filesystem_size_bytes{container="kube-rbac-proxy", device="/dev/mapper/centos-root", endpoint="https", fstype="xfs", instance="k8s-master03", job="node-exporter", mountpoint="/", namespace="monitoring", pod="node-exporter-5qj7b", service="node-exporter"}    37688381440
node_filesystem_size_bytes{container="kube-rbac-proxy", device="/dev/mapper/centos-root", endpoint="https", fstype="xfs", instance="k8s-node02", job="node-exporter", mountpoint="/", namespace="monitoring", pod="node-exporter-xjjf2", service="node-exporter"}    37688381440
node_filesystem_size_bytes{container="kube-rbac-proxy", device="/dev/mapper/centos_k8s--node01-root", endpoint="https", fstype="xfs", instance="k8s-node01", job="node-exporter", mountpoint="/", namespace="monitoring", pod="node-exporter-265df", service="node-exporter"}    37688381440
```

查询指定分区大小 `node_filesystem_size_bytes{mountpoint="/"}`：



Use local time Enable query history [✓ Enable autocomplete] Use experimental editor [✓ Enable highlighting] [✓ Enable linter]

```
node_filesystem_size_bytes{mountpoint="/"}
```

Load time: 63ms Resolution: 14s Result series: 5

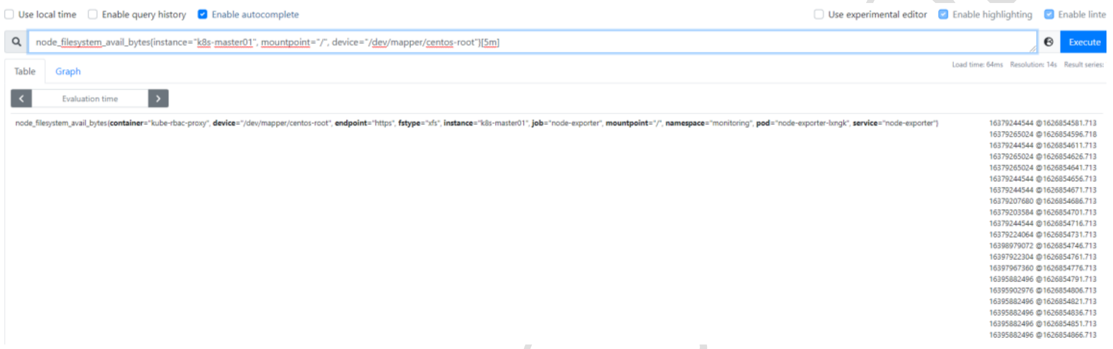
```
node_filesystem_size_bytes{container="kube-rbac-proxy", device="/dev/mapper/centos-root", endpoint="https", fstype="xfs", instance="k8s-master01", job="node-exporter", mountpoint="/", namespace="monitoring", pod="node-exporter-lxngk", service="node-exporter"}    37688381440
node_filesystem_size_bytes{container="kube-rbac-proxy", device="/dev/mapper/centos-root", endpoint="https", fstype="xfs", instance="k8s-master02", job="node-exporter", mountpoint="/", namespace="monitoring", pod="node-exporter-n8p7w", service="node-exporter"}    37688381440
node_filesystem_size_bytes{container="kube-rbac-proxy", device="/dev/mapper/centos-root", endpoint="https", fstype="xfs", instance="k8s-master03", job="node-exporter", mountpoint="/", namespace="monitoring", pod="node-exporter-5qj7b", service="node-exporter"}    37688381440
node_filesystem_size_bytes{container="kube-rbac-proxy", device="/dev/mapper/centos-root", endpoint="https", fstype="xfs", instance="k8s-node02", job="node-exporter", mountpoint="/", namespace="monitoring", pod="node-exporter-xjjf2", service="node-exporter"}    37688381440
node_filesystem_size_bytes{container="kube-rbac-proxy", device="/dev/mapper/centos_k8s--node01-root", endpoint="https", fstype="xfs", instance="k8s-node01", job="node-exporter", mountpoint="/", namespace="monitoring", pod="node-exporter-265df", service="node-exporter"}    37688381440
```

或者是查询分区不是/boot，且磁盘是/dev/开头的分区大小（结果不再展示）：

```
node_filesystem_size_bytes{device=~"/dev/.*", mountpoint!="/boot"}
```

查询主机 k8s-master01 在最近 5 分钟可用的磁盘空间变化：

```
node_filesystem_avail_bytes{instance="k8s-master01", mountpoint="/", 
device="/dev/mapper/centos-root"}[5m]
```



Use local time Enable query history [✓ Enable autocomplete] Use experimental editor [✓ Enable highlighting] [✓ Enable linte]

```
node_filesystem_avail_bytes{instance="k8s-master01", mountpoint="/", device="/dev/mapper/centos-root"}[5m]
```

Load time: 64ms Resolution: 14s Result series:

```
node_filesystem_avail_bytes{container="kube-rbac-proxy", device="/dev/mapper/centos-root", endpoint="https", fstype="xfs", instance="k8s-master01", job="node-exporter", mountpoint="/", namespace="monitoring", pod="node-exporter-lxngk", service="node-exporter"}
16379244544 @1626854581.713
16379265024 @1626854596.718
16379244544 @1626854611.713
16379265024 @1626854626.713
16379265024 @1626854641.713
16379244544 @1626854656.713
16379244544 @1626854671.713
16379207680 @1626854686.713
16379203584 @1626854701.713
16379244544 @1626854716.713
16379224064 @1626854731.713
16398979072 @1626854746.713
16397922304 @1626854761.713
16397967360 @1626854776.713
16395882496 @1626854791.713
16395902976 @1626854806.713
16395882496 @1626854821.713
16395882496 @1626854836.713
16395882496 @1626854851.713
16395882496 @1626854866.713
```

目前支持的范围单位如下：

- s：秒
- m：分钟
- h：小时
- d：天
- w：周
- y：年

查询 10 分钟之前磁盘可用空间，只需要指定 offset 参数即可：

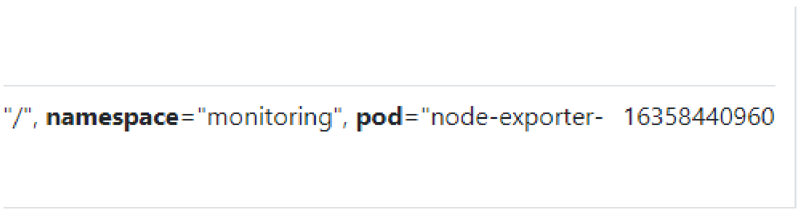
```
node_filesystem_avail_bytes{instance="k8s-master01", mountpoint="/", 
device="/dev/mapper/centos-root"} offset 10m
```

查询 10 分钟之前，5 分钟区间的磁盘可用空间的变化：

```
node_filesystem_avail_bytes{instance="k8s-master01", mountpoint="/", 
device="/dev/mapper/centos-root"}[5m] offset 10m
```

1.7.2 PromQL 操作符

通过 PromQL 的语法查到了主机磁盘的空间数据，查询结果如下：



```
"/", namespace="monitoring", pod="node-exporter-   16358440960
```

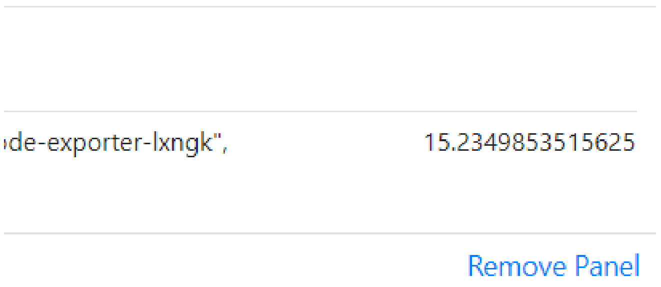
可以通过以下命令将字节转换为 GB 或者 MB：

```
    node_filesystem_avail_bytes{instance="k8s-master01", mountpoint="/",
device="/dev/mapper/centos-root"} / 1024 / 1024 / 1024
```

也可以将 **1024 / 1024 / 1024** 改为**(1024 ^ 3)**：

```
    node_filesystem_avail_bytes{instance="k8s-master01", mountpoint="/",
device="/dev/mapper/centos-root"} / (1024 ^ 3)
```

查询结果如下图所示，此时为 15GB 左右：



```
de-exporter-lxngk",                15.2349853515625
```

此时可以在宿主机上比对数据是否正确：

```
    # df -Th | grep /dev/mapper/centos-root
    /dev/mapper/centos-root xfs       36G   20G   16G  57% /
```

上述使用的 “/” 为数学运算的 “除”，“^” 为幂运算，同时也支持如下运算符：

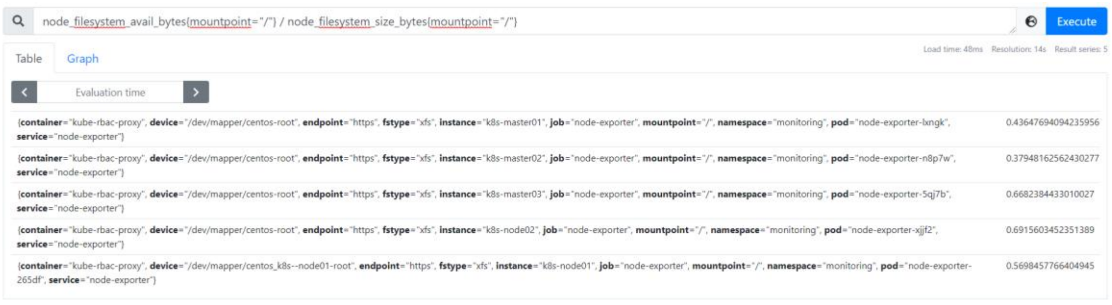
- +： 加
- -： 减
- *： 乘
- /： 除
- ^： 幂运算
- %： 求余

查询 k8s-master01 根区分磁盘可用率，可以通过如下指令进行计算：

```
    node_filesystem_avail_bytes{instance="k8s-master01", mountpoint="/",
device="/dev/mapper/centos-root"} /
node_filesystem_size_bytes{instance="k8s-master01", mountpoint="/",
device="/dev/mapper/centos-root"}
```

查询所有主机根分区的可用率：

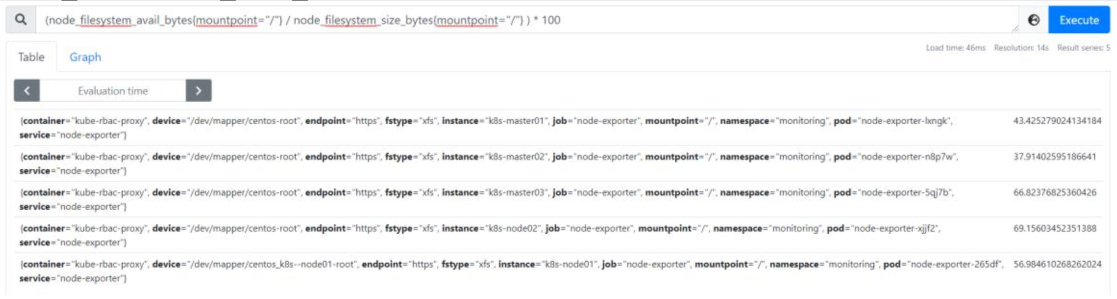
```
    node_filesystem_avail_bytes{mountpoint="/"} /
node_filesystem_size_bytes{mountpoint="/"}
```



```
node_filesystem_avail_bytes{mountpoint="/"} / node_filesystem_size_bytes{mountpoint="/"}

{container="kube-rbac-proxy", device="/dev/mapper/centos-root", endpoint="https", fstype="xfs", instance="k8s-master01", job="node-exporter", mountpoint="/", namespace="monitoring", pod="node-exporter-lxngk", service="node-exporter"}    0.43647694094235956
{container="kube-rbac-proxy", device="/dev/mapper/centos-root", endpoint="https", fstype="xfs", instance="k8s-master02", job="node-exporter", mountpoint="/", namespace="monitoring", pod="node-exporter-n8p7w", service="node-exporter"}    0.37948162562430277
{container="kube-rbac-proxy", device="/dev/mapper/centos-root", endpoint="https", fstype="xfs", instance="k8s-master03", job="node-exporter", mountpoint="/", namespace="monitoring", pod="node-exporter-5qj7b", service="node-exporter"}    0.6682384433010027
{container="kube-rbac-proxy", device="/dev/mapper/centos-root", endpoint="https", fstype="xfs", instance="k8s-node02", job="node-exporter", mountpoint="/", namespace="monitoring", pod="node-exporter-xjjf2", service="node-exporter"}    0.6915603452351389
{container="kube-rbac-proxy", device="/dev/mapper/centos_k8s--node01-root", endpoint="https", fstype="xfs", instance="k8s-node01", job="node-exporter", mountpoint="/", namespace="monitoring", pod="node-exporter-265df", service="node-exporter"}    0.5698457766404945
```

也可以将结果乘以 100 直接得到百分比：

```
    (node_filesystem_avail_bytes{mountpoint="/"} /
node_filesystem_size_bytes{mountpoint="/"} ) * 100
```



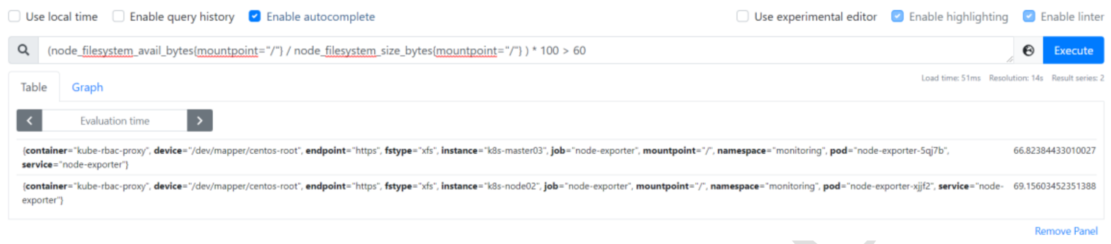
```
(node_filesystem_avail_bytes{mountpoint="/"} / node_filesystem_size_bytes{mountpoint="/"} ) * 100
```

Load time: 46ms   Resolution: 14s   Result series: 5

```
{container="kube-rbac-proxy", device="/dev/mapper/centos-root", endpoint="https", fstype="xfs", instance="k8s-master01", job="node-exporter", mountpoint="/", namespace="monitoring", pod="node-exporter-lxngk", service="node-exporter"}    43.425279024134184
{container="kube-rbac-proxy", device="/dev/mapper/centos-root", endpoint="https", fstype="xfs", instance="k8s-master02", job="node-exporter", mountpoint="/", namespace="monitoring", pod="node-exporter-n8p7w", service="node-exporter"}    37.91402595186641
{container="kube-rbac-proxy", device="/dev/mapper/centos-root", endpoint="https", fstype="xfs", instance="k8s-master03", job="node-exporter", mountpoint="/", namespace="monitoring", pod="node-exporter-5qj7b", service="node-exporter"}    66.82376825360426
{container="kube-rbac-proxy", device="/dev/mapper/centos-root", endpoint="https", fstype="xfs", instance="k8s-node02", job="node-exporter", mountpoint="/", namespace="monitoring", pod="node-exporter-xjjf2", service="node-exporter"}    69.15603452351388
{container="kube-rbac-proxy", device="/dev/mapper/centos_k8s--node01-root", endpoint="https", fstype="xfs", instance="k8s-node01", job="node-exporter", mountpoint="/", namespace="monitoring", pod="node-exporter-265df", service="node-exporter"}    56.984610268262024
```

找到集群中根分区空间可用率大于 60%的主机：

```
    (node_filesystem_avail_bytes{mountpoint="/"} /
node_filesystem_size_bytes{mountpoint="/"} ) * 100 > 60
```



Use local time: No   Enable query history: No   Enable autocomplete: [✓]   Use experimental editor: No   Enable highlighting: [✓]   Enable linter: [✓]

```
(node_filesystem_avail_bytes{mountpoint="/"} / node_filesystem_size_bytes{mountpoint="/"} ) * 100 > 60
```

Load time: 51ms   Resolution: 14s   Result series: 2

```
{container="kube-rbac-proxy", device="/dev/mapper/centos-root", endpoint="https", fstype="xfs", instance="k8s-master03", job="node-exporter", mountpoint="/", namespace="monitoring", pod="node-exporter-5qj7b", service="node-exporter"}    66.82384433010027
{container="kube-rbac-proxy", device="/dev/mapper/centos-root", endpoint="https", fstype="xfs", instance="k8s-node02", job="node-exporter", mountpoint="/", namespace="monitoring", pod="node-exporter-xjjf2", service="node-exporter"}    69.15603452351388
```

PromQL 也支持如下判断：

- ==： (相等)
- != ：(不相等)
- >：(大于)
- < ：(小于)
- >= ：(大于等于)
- <= ：(小于等于)

磁盘可用率大于 30%小于等于 60%的主机：

```
    30 < (node_filesystem_avail_bytes{mountpoint="/"} /
node_filesystem_size_bytes{mountpoint="/"} ) * 100 <= 60
```

也可以用 and 进行联合查询：

```
    (node_filesystem_avail_bytes{mountpoint="/"} /
node_filesystem_size_bytes{mountpoint="/"} ) * 100 > 30 and
(node_filesystem_avail_bytes{mountpoint="/"} /
node_filesystem_size_bytes{mountpoint="/"} ) * 100 <=60
```

除了 and 外，也支持 or 和 unless：

- and： 并且
- or ：或者
- unless ：排除

查询主机磁盘剩余空间，并且排除掉 shm 和 tmpfs 的磁盘：

```
    node_filesystem_free_bytes unless
node_filesystem_free_bytes{device=~"shm|tmpfs"}
```

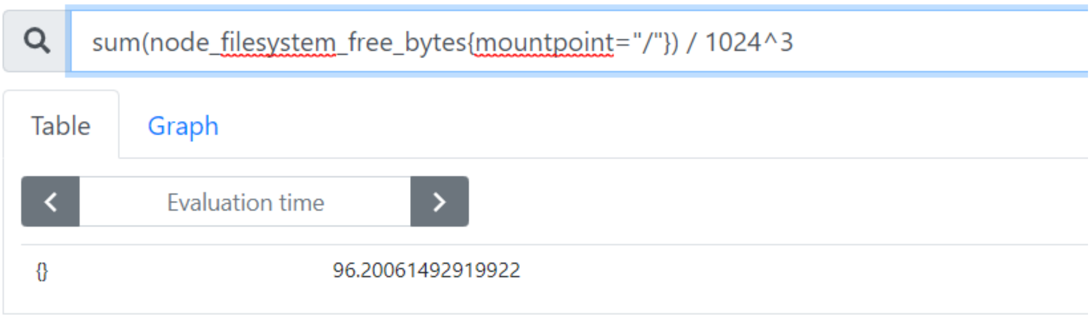
当然，这个语法也可以直接写为：

```
    node_filesystem_free_bytes{device=~"shm|tmpfs"}
```

1.7.3 PromQL 常用函数

使用 sum 函数统计当前监控目标所有主机根分区剩余的空间：

```
    sum(node_filesystem_free_bytes{mountpoint="/"}) / 1024^3
```

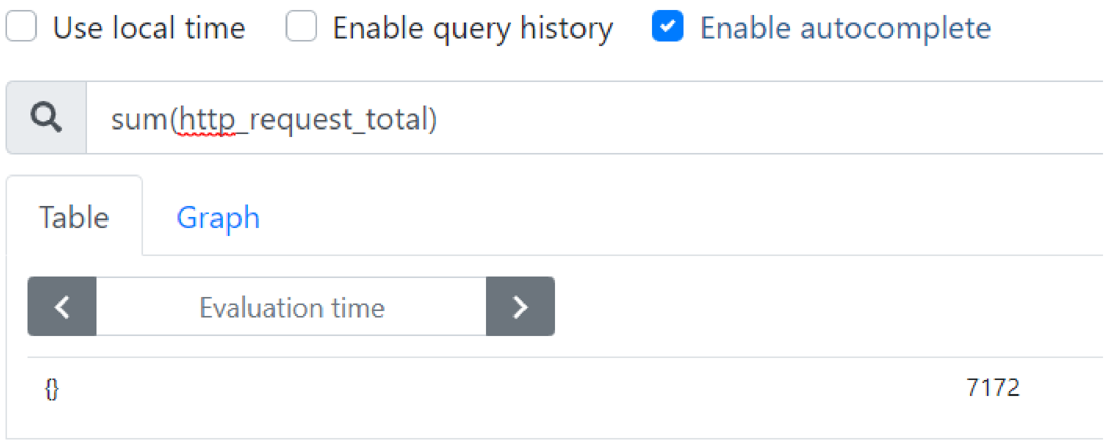


```
sum(node_filesystem_free_bytes{mountpoint="/"}) / 1024^3
```

```
{}                    96.20061492919922
```

也可以用同样方式，计算所有的请求总量：

```
sum(http_request_total)
```



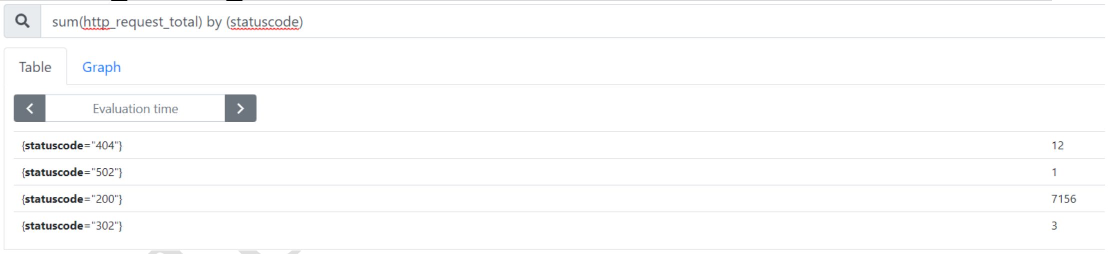
Use local time [✓ Enable autocomplete] Enable query history

```
sum(http_request_total)
```

```
{}                    7172
```

根据 statuscode 字段进行统计请求数据：

```
sum(http_request_total) by (statuscode)
```

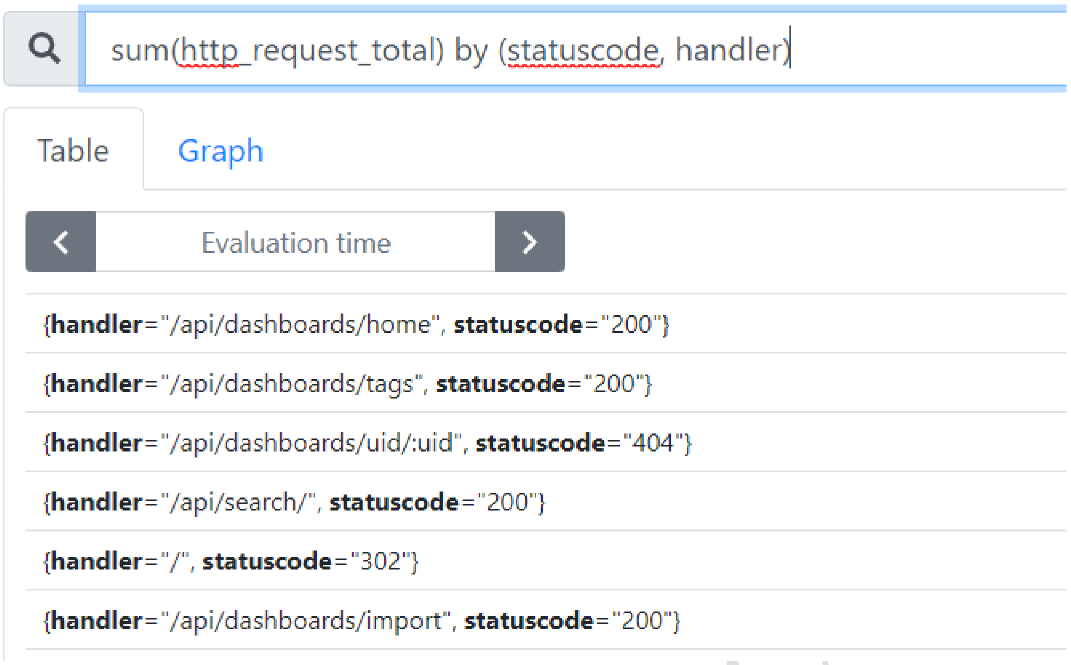


```
sum(http_request_total) by (statuscode)
```

```
{statuscode="404"}    12
{statuscode="502"}    1
{statuscode="200"}    7156
{statuscode="302"}    3
```

根据 statuscode 和 handler 两个指标进一步统计：

```
sum(http_request_total) by (statuscode, handler)
```


```
sum(http_request_total) by (statuscode, handler)
```
Table Graph

Evaluation time

```
{handler="/api/dashboards/home", statuscode="200"}
{handler="/api/dashboards/tags", statuscode="200"}
{handler="/api/dashboards/uid/:uid", statuscode="404"}
{handler="/api/search/", statuscode="200"}
{handler="/", statuscode="302"}
{handler="/api/dashboards/import", statuscode="200"}
```

找到排名前五的数据：

```
topk(5, sum(http_request_total) by (statuscode, handler))
```

取最后三个数据：

```
bottomk(3, sum(http_request_total) by (statuscode, handler))
```

找出统计结果中最小的数据：

```
min(node_filesystem_avail_bytes{mountpoint="/"})
```

最大的数据：

```
max(node_filesystem_avail_bytes{mountpoint="/"})
```

平均值：

```
avg(node_filesystem_avail_bytes{mountpoint="/"})
```

四舍五入，向上取最接近的整数，2.79 → 3：

```
ceil(node_filesystem_files_free{mountpoint="/"} / 1024 / 1024)
```

向下取整数， 2.79 → 2：

```
floor(node_filesystem_files_free{mountpoint="/"} / 1024 / 1024)
```

对结果进行正向排序：

```
sort(sum(http_request_total) by (handler, statuscode))
```

对结果进行逆向排序：

```
sort_desc(sum(http_request_total) by (handler, statuscode))
```

函数 **predict_linear** 可以用于预测分析和预测性告警，比如可以根据一天的数据，预测 4 个小时后，磁盘分区的空间会不会小于 0：

```
    predict_linear(node_filesystem_files_free{mountpoint="/"}[1d], 4*3600) <
0
```

除了上述的函数，还有几个比较重要的函数，比如 increase、rate、irate。其中 increase 是计算在一段时间范围内数据的增长（只能计算 count 类型的数据），rate 和 irate 是计算增长率。比如查询某个请求在 1 小时的时间增长了多少：

```
    increase(http_request_total{handler="/api/datasources/proxy/:id/*",metho
d="get",namespace="monitoring",service="grafana",statuscode="200"}[1h])
```

将 1h 增长的数量处于该时间即为增长率：

```
    increase(http_request_total{handler="/api/datasources/proxy/:id/*",metho
d="get",namespace="monitoring",service="grafana",statuscode="200"}[1h]) /
3600
```

相对于 increase，rate 可以直接计算出某个指标在给定时间范围内的增长率，比如还是计算 1h 的增长率，可以用 rate 函数进行计算：

```
    rate(http_request_total{handler="/api/datasources/proxy/:id/*",method="g
et",namespace="monitoring",service="grafana",statuscode="200"}[1h])
```

长尾效应

1.8 Alertmanager 告警入门

https://github.com/dotbalo/k8s/blob/master/prometheus-operator/alertmanager.yaml
https://prometheus.io/docs/alerting/latest/configuration/
https://github.com/prometheus/alertmanager/blob/main/doc/examples/simple.yml

1.8.1 Alertmanager 配置文件解析

首先看一下一个简单的 Alertmanager 的配置示例：

```
    global:
      resolve_timeout: 5m
      ...
    https://alert.victorops.com/integrations/generic/20131114/alert/
    route:
      receiver: Default
      group_by:
      - namespace
      - job
      - alertname
      routes:
      - receiver: Watchdog
        match:
          alertname: Watchdog
      - receiver: Critical
        match:
          severity: critical
      group_wait: 30s
      group_interval: 5m
      repeat_interval: 10m
    inhibit_rules:
    - source_match:
        severity: critical
      target_match_re:
        severity: warning|info
      equal:
      - namespace
      - alertname
    - source_match:
        severity: warning
      target_match_re:
        severity: info
      equal:
      - namespace
      - alertname
    receivers:
    - name: Default
      email_configs:
      - send_resolved: true
        to: kubernetes_guide@163.com
        from: kubernetes_guide@163.com
        hello: 163.com
        smarthost: smtp.163.com:465
        auth_username: kubernetes_guide@163.com
        auth_password: <secret>
        headers:
          From: kubernetes_guide@163.com
          Subject: '{{ template "email.default.subject" . }}'
          To: kubernetes_guide@163.com
        html: '{{ template "email.default.html" . }}'
        require_tls: false
    - name: Watchdog
    - name: Critical
    templates: []
```

Alertmanager 的配置主要分为五大块：

- Global：全局配置，主要用来配置一些通用的配置，比如邮件通知的账号、密码、SMTP 服务器、微信告警等。Global 块配置下的配置选项在本配置文件内的所有配置项下可见，但是文件内其它位置的子配置可以覆盖 Global 配置；
- Templates：用于放置自定义模板的位置；
- Route：告警路由配置，用于告警信息的分组路由，可以将不同分组的告警发送给不同的收件人。比如将数据库告警发送给 DBA，服务器告警发送给 OPS；
- Inhibit_rules：告警抑制，主要用于减少告警的次数，防止"告警轰炸"。比如某个宿主机宕机，可能会引起容器重建、漂移、服务不可用等一系列问题，如果每个异常均有告警，会一次性发送很多告警，造成告警轰炸，并且也会干扰定位问题的思路，所以可以使用告警抑制，屏蔽由宿主机宕机引来的其他问题，只发送宿主机宕机的消息即可；
- Receivers：告警收件人配置，每个 receiver 都有一个名字，经过 route 分组并且路由后需要指定一个 receiver，就是在此位置配置的。

了解完 Alertmanager 主要的配置块后，接下来需要对 Alertmanager 比较重要的 Route 进行单独讲解，其它配置会在实践中进行补充。

1.8.2 Alertmanager 路由规则

route 配置：

```
route:
  receiver: Default
  group_by:
  - namespace
  - job
  - alertname
  routes:
  - receiver: Watchdog
    match:
      alertname: Watchdog
  - receiver: Critical
    match:
      severity: critical
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 10m
```

- receiver：告警的通知目标，需要和 receivers 配置中 name 进行匹配。需要注意的是 route.routes 下也可以有 receiver 配置，优先级高于 route.receiver 配置的默认接收人，当告警没有匹配到子路由时，会使用 route.receiver 进行通知，比如上述配置中的 Default；

- group_by：分组配置，值类型为列表。比如配置成['job', 'severity']，代表告警信息包含 job 和 severity 标签的会进行分组，且标签的 key 和 value 都相同才会被分到一组；
- continue：决定匹配到第一个路由后，是否继续后续匹配。默认为 false，即匹配到第一个子节点后停止继续匹配；
- match：一对一匹配规则，比如 match 配置的为 job: mysql，那么具有 job=mysql 的告警会进入该路由；
- match_re：和 match 类似，只不过是 match_re 是正则匹配；
- group_wait：告警通知等待，值类型为字符串。若一组新的告警产生，则会等 group_wait 后再发送通知，该功能主要用于当告警在很短时间内接连产生时，在 group_wait 内合并为单一的告警后再发送，防止告警过多，默认值 30s；
- group_interval：同一组告警通知后，如果有新的告警添加到该组中，再次发送告警通知的时间，默认值为 5m；
- repeat_interval：如果一条告警通知已成功发送，且在间隔 repeat_interval 后，该告警仍然未被设置为 resolved，则会再次发送该告警通知，默认值 4h。

1.8.3 Alertmanager 邮件通知

找到 Alertmanager 的配置文件：

```
[root@k8s-master01 kube-prometheus]# cd manifests/
[root@k8s-master01 manifests]# ls alertmanager-secret.yaml
alertmanager-secret.yaml
# cat alertmanager-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  labels:
    alertmanager: main
    app.kubernetes.io/component: alert-router
    app.kubernetes.io/name: alertmanager
    app.kubernetes.io/part-of: kube-prometheus
    app.kubernetes.io/version: 0.21.0
  name: alertmanager-main
  namespace: monitoring
stringData:
  alertmanager.yaml: |-
    "global":
      "resolve_timeout": "5m"
...
```

之后在 alertmanager-secret.yaml 文件的 global 添加配置如下：

```
  alertmanager.yaml: |-
    "global":
      "resolve_timeout": "5m"
      smtp_from: "kubernetesxxx@163.com"
      smtp_smarthost: "smtp.163.com:465"
      smtp_hello: "163.com"
      smtp_auth_username: "kubernetesxxx@163.com"
      smtp_auth_password: "QJUYMWJXXX"
      smtp_require_tls: false
```

之后将名称为 Default 的 receiver 配置更改为邮件通知，修改 alertmanager-secret.yaml 文件的 receivers 配置如下：

```
    "receivers":
    - "name": "Default"
      "email_configs":
      - to: "notification@163.com"
        send_resolved: true
    - "name": "Watchdog"
    - "name": "Critical"
```

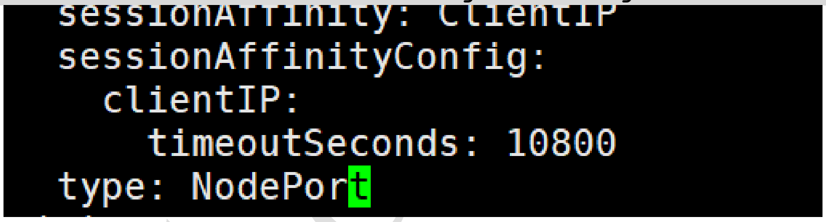
- ➢ email_configs：代表使用邮件通知；
- ➢ to：收件人，此处为 notification@163.com，可以配置多个，逗号隔开；
- ➢ send_resolved：告警如果被解决是否发送解决通知。

接下来分析一下路由规则（默认分组只有 namespace，在此添加上 job 和 alertname，当然不添加也是可以的）：

```
      "route":
        "group_by":
        - "namespace"
        - "job"
        - "alertname"
        "group_interval": "5m"
        "group_wait": "30s"
        "receiver": "Default"
        "repeat_interval": "10m"
        "routes":
        - "match":
            "alertname": "Watchdog"
          "receiver": "Watchdog"
        - "match":
            "severity": "critical"
          "receiver": "Critical"
```

可以通过 Alertmanager 提供的 Web UI 查看分组信息，和 Prometheus 一致，将 Alertmanager 的 Service 更改为 NodePort：

```
# kubectl  edit svc -n monitoring alertmanager-main
```



```
  sessionAffinity: ClientIP
  sessionAffinityConfig:
    clientIP:
      timeoutSeconds: 10800
  type: NodePort
```

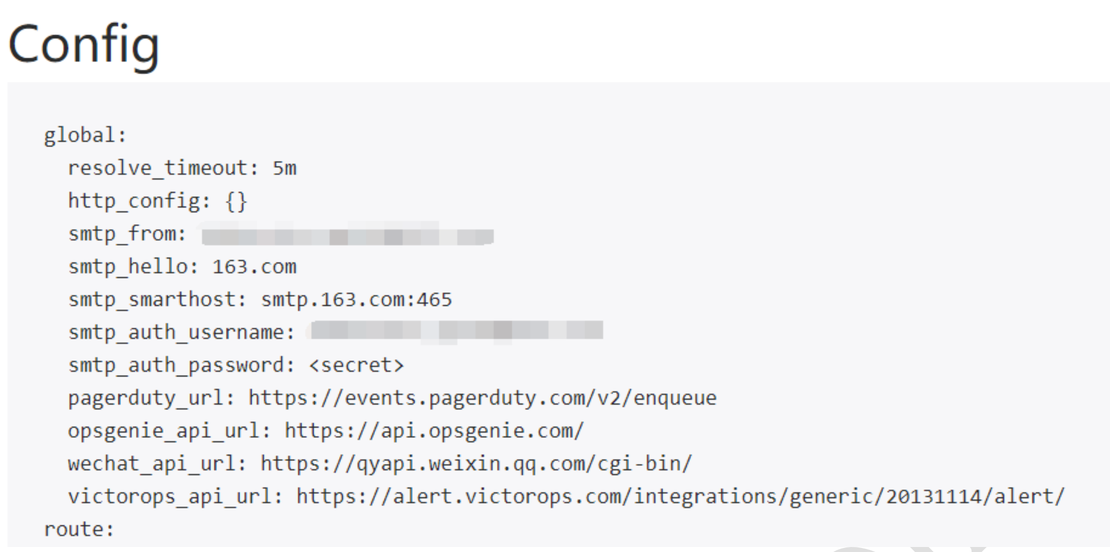
查看监听的端口号：

```
# kubectl get svc -n monitoring alertmanager-main
NAME                TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
alertmanager-main   NodePort   192.168.92.94   <none>        9093:30409/TCP
10d
```

将更改好的 Alertmanager 配置加载到 Alertmanager：

```
# kubectl replace -f alertmanager-secret.yaml
secret/alertmanager-main replaced
```

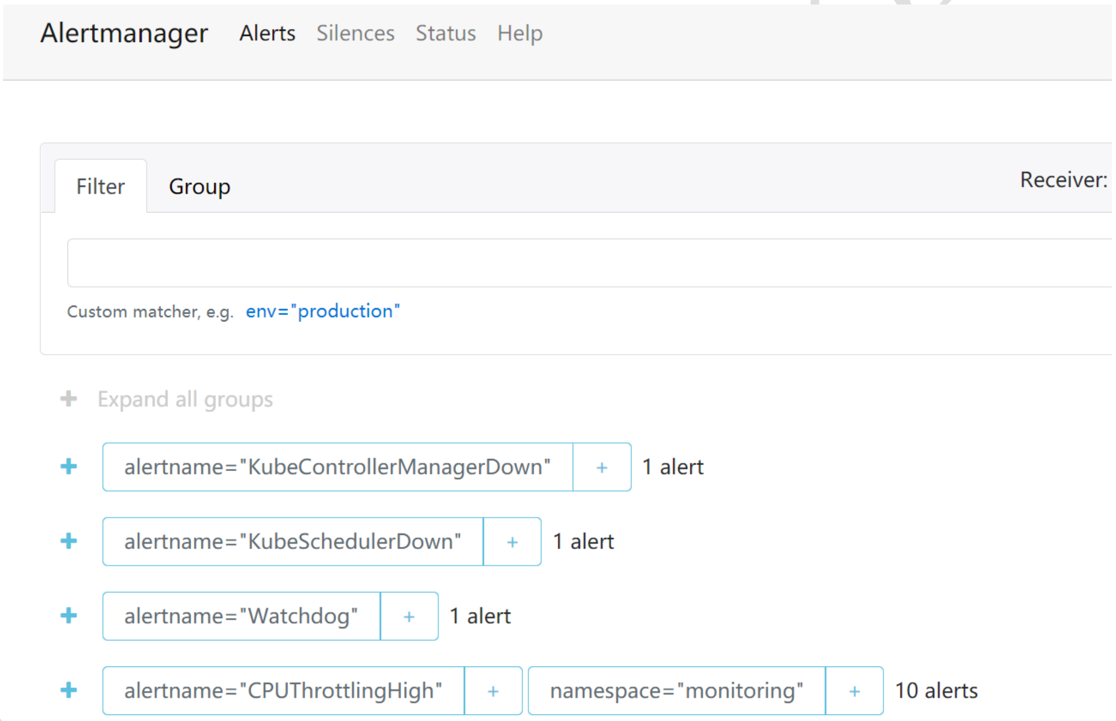
稍等几分钟即可在 Alertmanager 的 Web 界面看到更改的配置（Status）：



Config

```
global:
  resolve_timeout: 5m
  http_config: {}
  smtp_from: [redacted]
  smtp_hello: 163.com
  smtp_smarthost: smtp.163.com:465
  smtp_auth_username: [redacted]
  smtp_auth_password: <secret>
  pagerduty_url: https://events.pagerduty.com/v2/enqueue
  opsgenie_api_url: https://api.opsgenie.com/
  wechat_api_url: https://qyapi.weixin.qq.com/cgi-bin/
  victorops_api_url: https://alert.victorops.com/integrations/generic/20131114/alert/
route:
```

也可以查看分组信息：



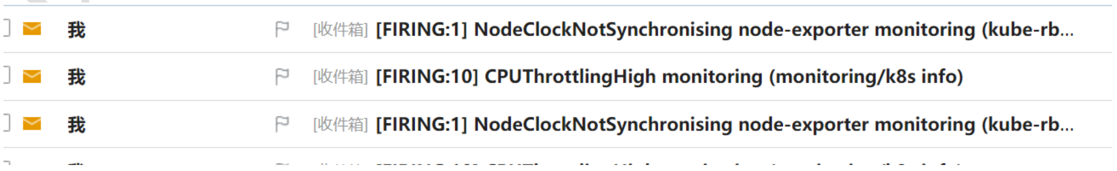
alertname="KubeControllerManagerDown" 1 alert

alertname="KubeSchedulerDown" 1 alert

alertname="Watchdog" 1 alert

alertname="CPUThrottlingHigh" namespace="monitoring" 10 alerts

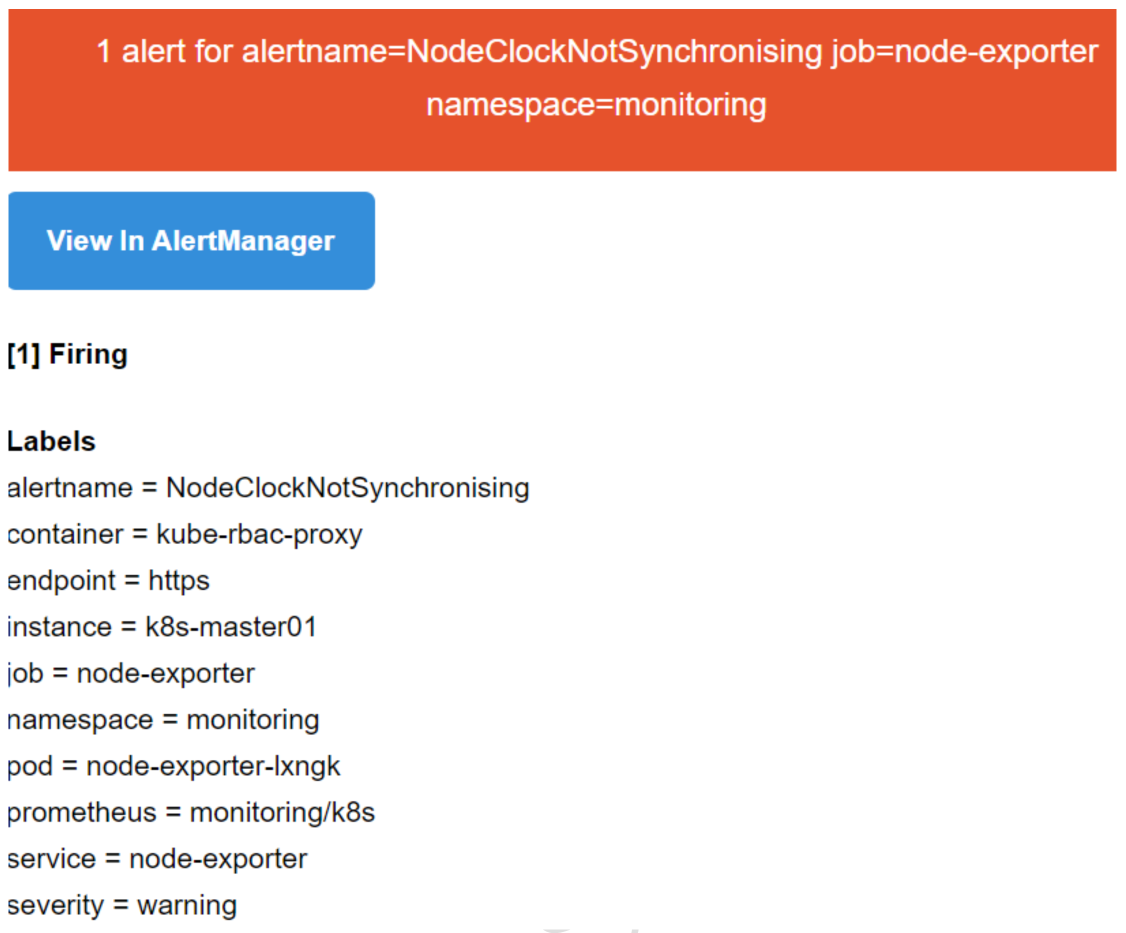
此时 Default receiver 配置的邮箱会收到两者的告警信息，如下所示：



我 [收件箱] **[FIRING:1] NodeClockNotSynchronising node-exporter monitoring (kube-rb...**

我 [收件箱] **[FIRING:10] CPUThrottlingHigh monitoring (monitoring/k8s info)**

我 [收件箱] **[FIRING:1] NodeClockNotSynchronising node-exporter monitoring (kube-rb...**



1 alert for alertname=NodeClockNotSynchronising job=node-exporter namespace=monitoring

View In AlertManager

[1] Firing

Labels

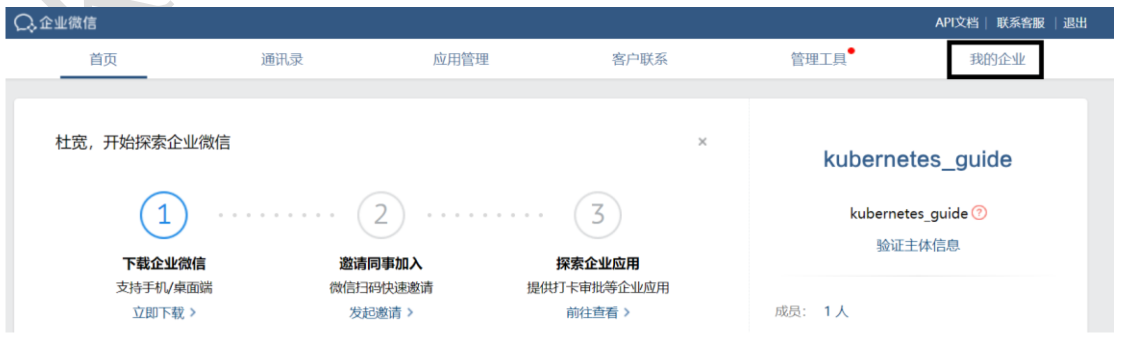
```
alertname = NodeClockNotSynchronising
container = kube-rbac-proxy
endpoint = https
instance = k8s-master01
job = node-exporter
namespace = monitoring
pod = node-exporter-lxngk
prometheus = monitoring/k8s
service = node-exporter
severity = warning
```

1.8.4 Alertmanager 企业微信通知

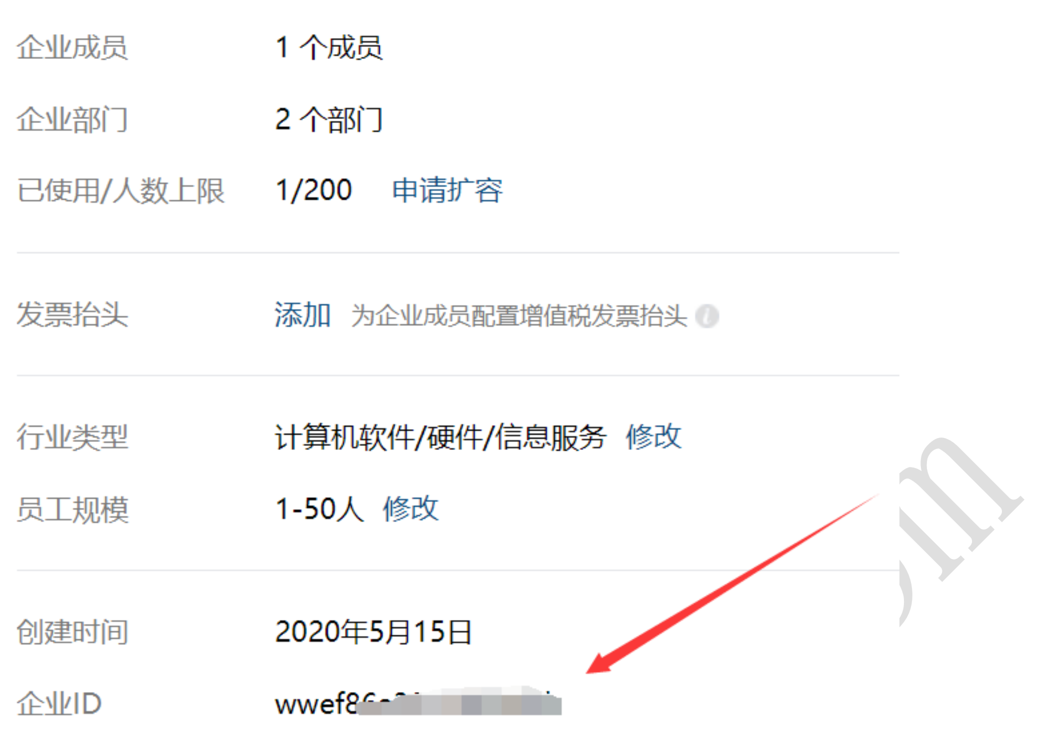
https://work.weixin.qq.com/

1.8.4.1 企业微信配置

注册完成后进行登录，登录后点击我的企业：

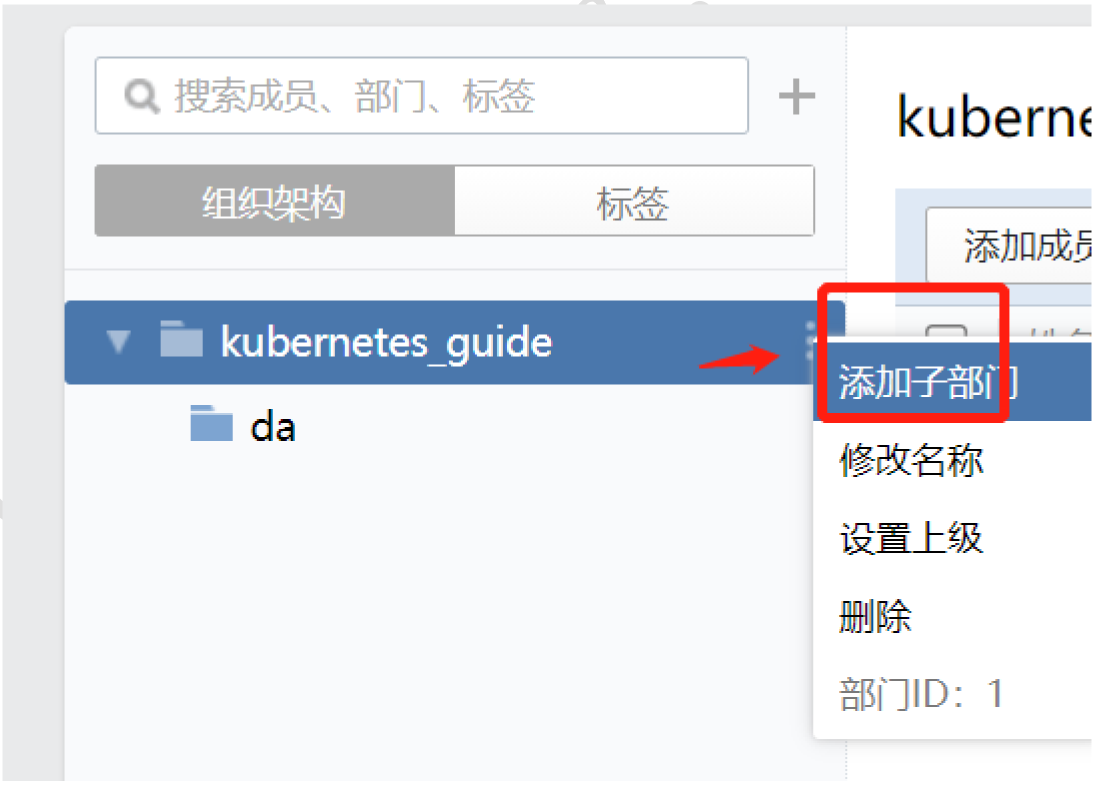


在页面的最下面找到企业 ID（corp_id）并记录，稍后会用到：

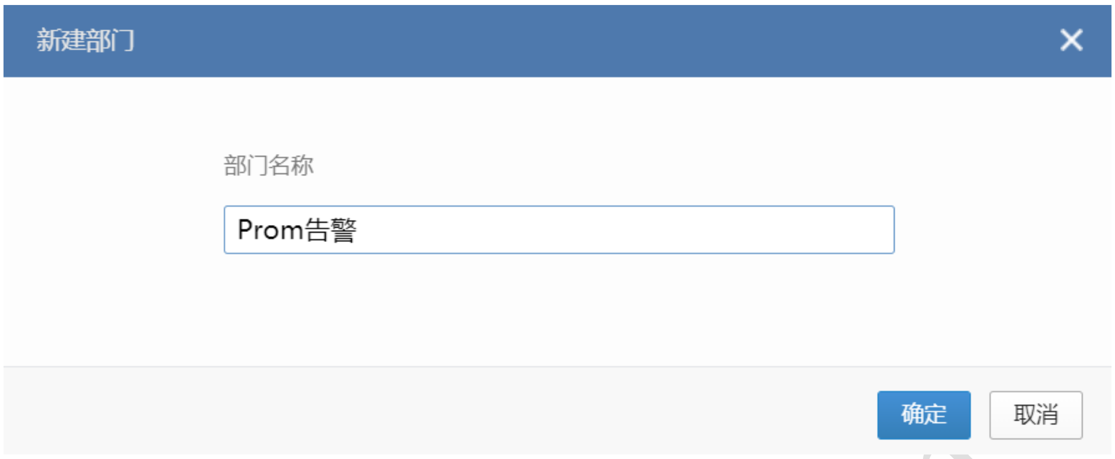


企业成员	1 个成员
企业部门	2 个部门
已使用/人数上限	1/200　申请扩容
发票抬头	添加　为企业成员配置增值税发票抬头
行业类型	计算机软件/硬件/信息服务　修改
员工规模	1-50人　修改
创建时间	2020年5月15日
企业ID	wwef8[redacted]

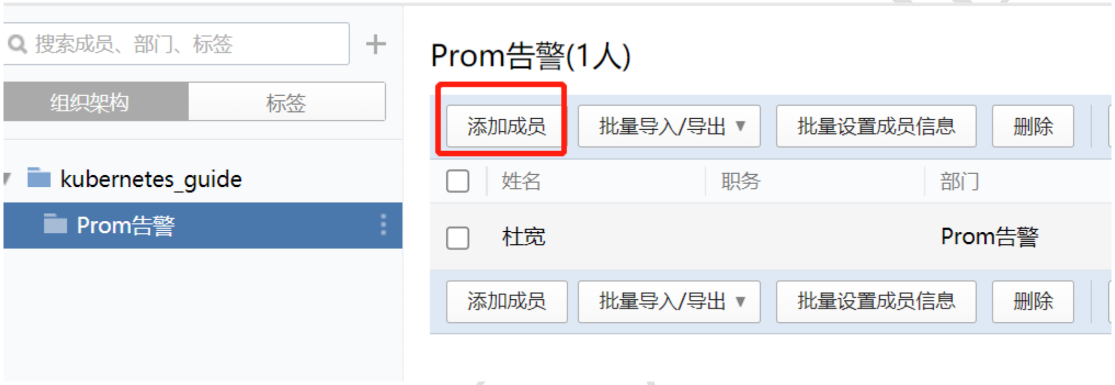
之后创建一个部门，用于接收告警通知：



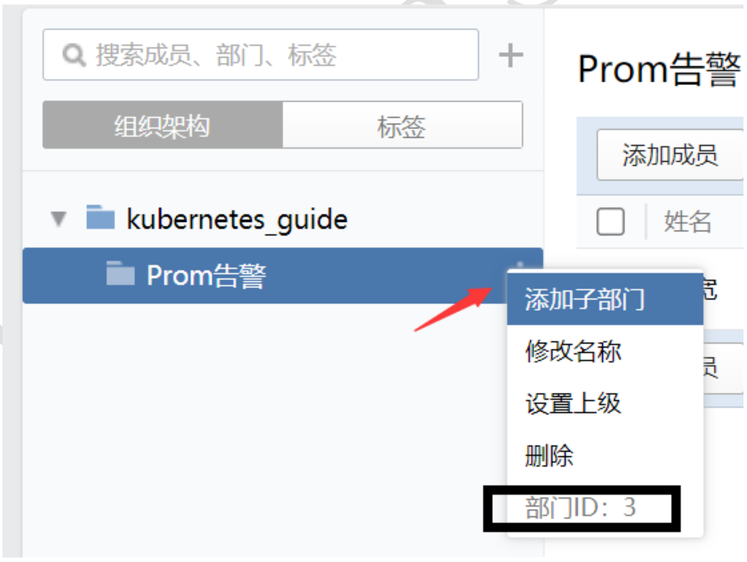
输入 Prom 告警，之后点击确定：



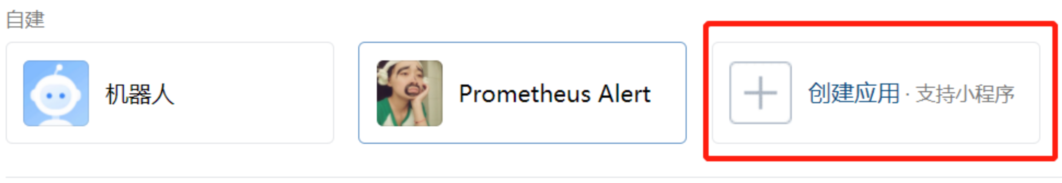
之后在 Prom 告警子部门添加相关的人员即可，在此不再演示：



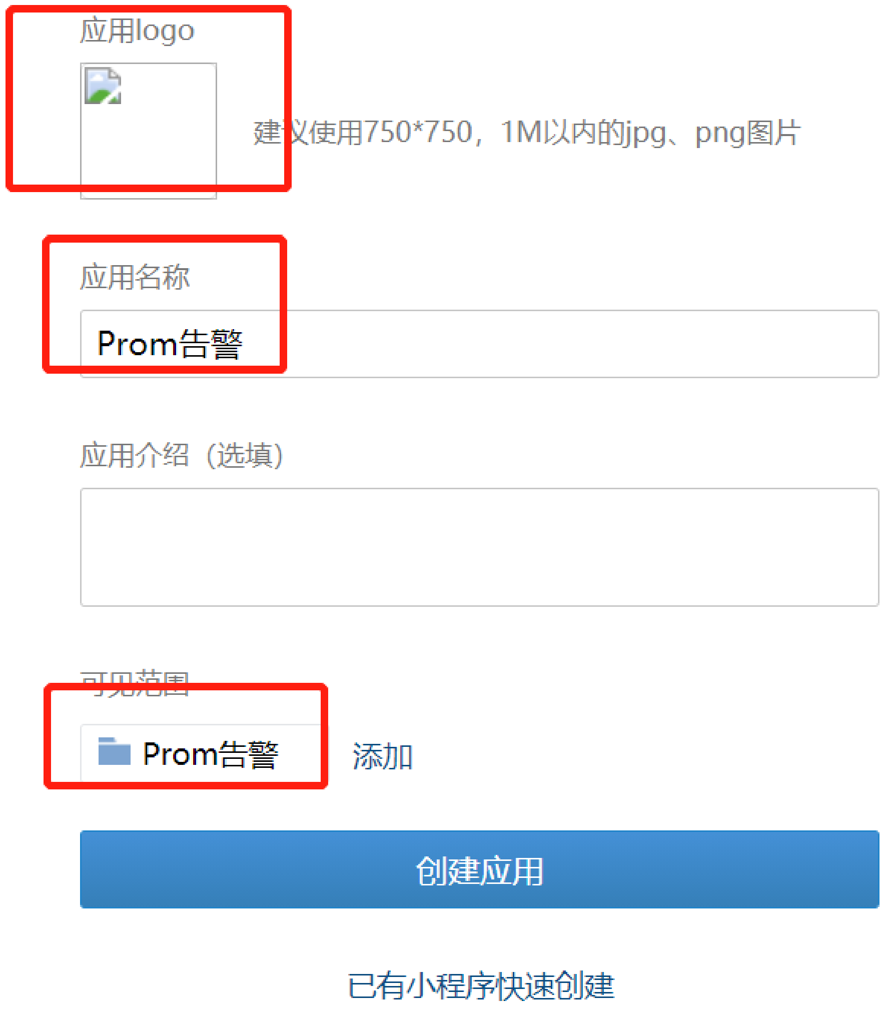
查看该部门 ID（to_party）并记录：



之后创建机器人应用，首先点击应用管理→应用创建：



选择一个 logo，输入应用名称和选择可见范围即可：



应用logo

建议使用750*750，1M以内的jpg、png图片

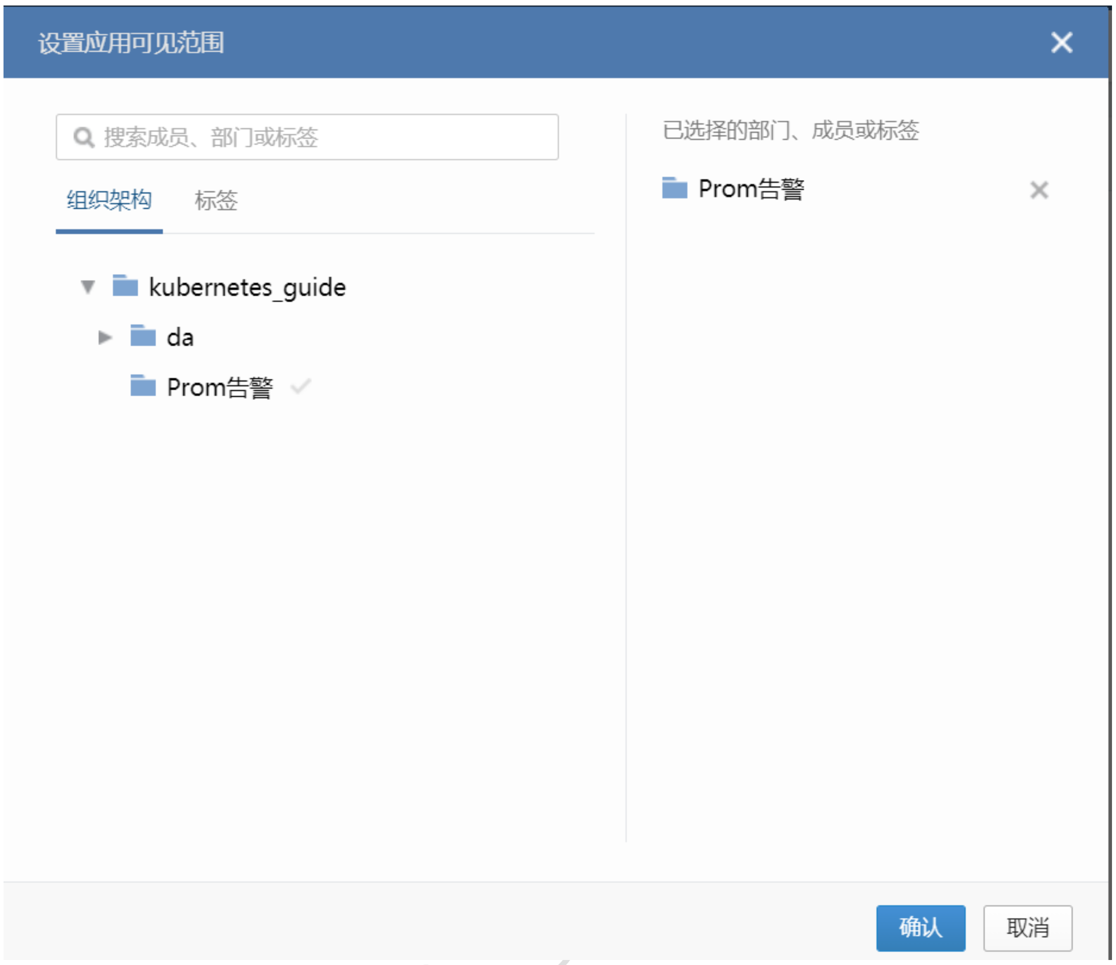
应用名称

Prom告警

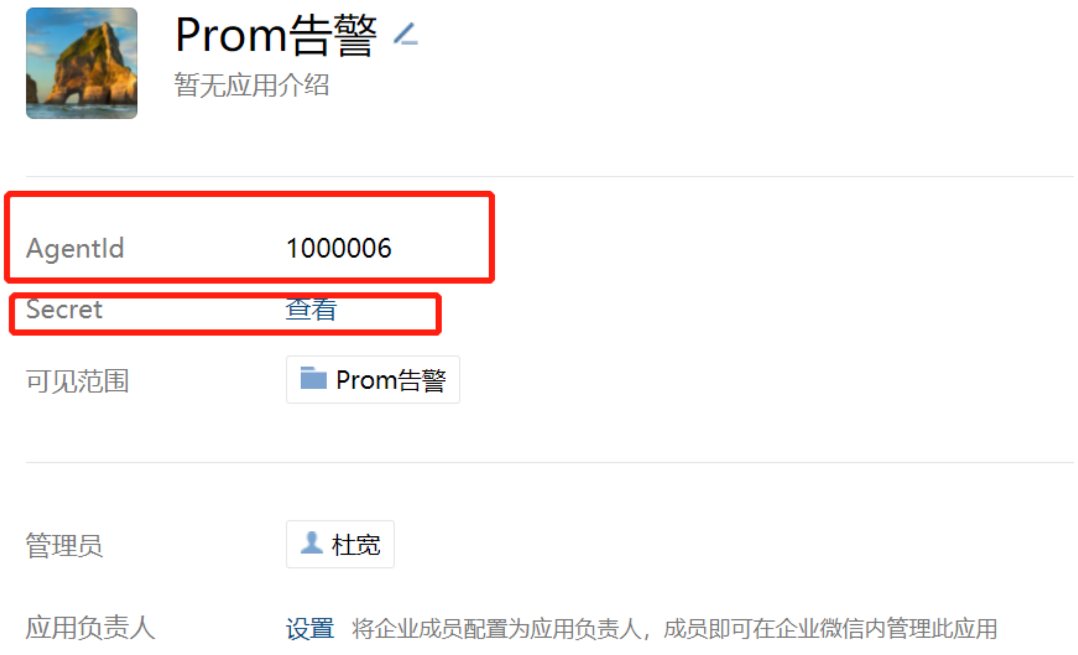
应用介绍（选填）

可见范围

Prom告警 添加



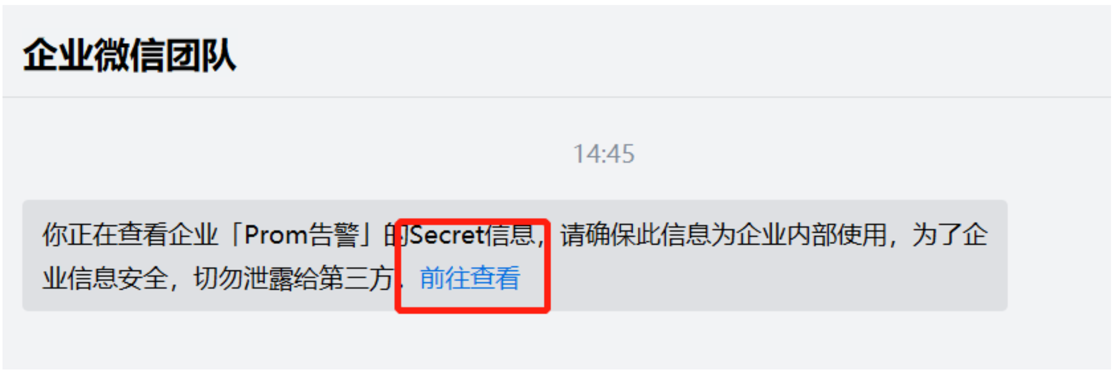
创建完成后，查看 AgentId 和 Secret（api_secret）并记录：


Prom告警

暂无应用介绍

AgentId	1000006
Secret	查看
可见范围	Prom告警
管理员	杜宽
应用负责人	设置 将企业成员配置为应用负责人，成员即可在企业微信内管理此应用

点击查看 Secret，企业微信会将 Secret 发送至企业微信：


企业微信团队

14:45

你正在查看企业「Prom告警」的Secret信息，请确保此信息为企业内部使用，为了企业信息安全，切勿泄露给第三方，前往查看

点击查看并记录即可。

1.8.4.2 Alertmanager 配置

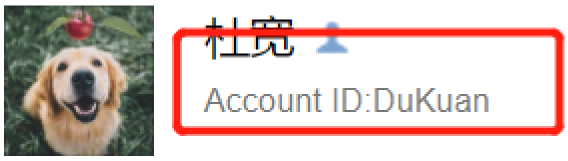
修改 Alertmanager 配置文件，添加企业微信告警。首先修改 Global，添加一些通用配置，wechat_api_url 是固定配置，corp_id 为企业 ID：

```
    "global":
      "resolve_timeout": "5m"
     ...
      wechat_api_url: "https://qyapi.weixin.qq.com/cgi-bin/"
      wechat_api_corp_id: "wwef86a30xxxxxxxxx"
```

Receivers 添加微信通知：

```
    "receivers":
    - name: wechat-ops
      wechat_configs:
      - send_resolved: true
        to_party: 3
        to_user: '@all'
        agent_id: 1000006
        api_secret: "3bB350Sxxxxxxxxxxxxxxxxxxxxxxx"
    - "name": "Default"
      "email_configs":
      - to: "xxxxx@163.com"
        send_resolved: true
```

此处配置的 receiver 名字为 wechat-ops，to_user 为@all，代表发送给所有人，也可以只发送给部门的某一个人，只需要将此处改为 USER_ID 即可：


杜宽

Account ID:DuKuan

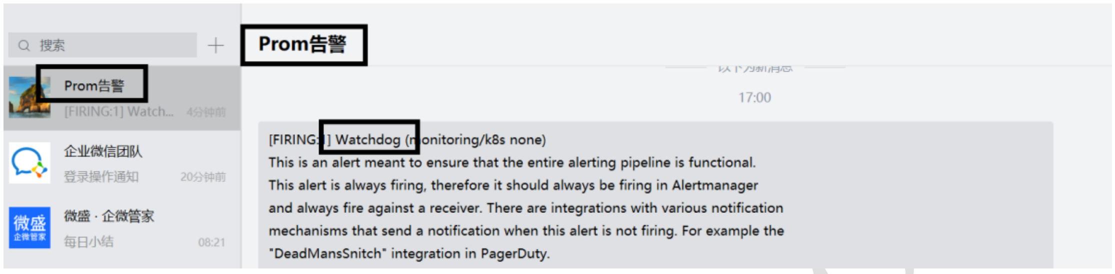
更改路由配置，将 Watchdog 的告警发送给该部门：

```
    "route":
      "group_by":
      - "namespace"
      - "job"
      - "alertname"
      "group_interval": "5m"
      "group_wait": "30s"
      "receiver": "Default"
      "repeat_interval": "2h"
      "routes":
          - "match":
              "alertname": "Watchdog"
            "receiver": "wechat-ops"
            "repeat_interval": "10m"
        ...
```

之后更新 Alertmanager 的配置：

```
    # kubectl replace -f alertmanager-secret.yaml
    secret/alertmanager-main replaced
```

等待几分钟后，可以在 Alertmanager Web UI 看到新配置，并且企业微信可以收到 Watchdog 的告警：



[FIRING:1] Watchdog (monitoring/k8s none)
This is an alert meant to ensure that the entire alerting pipeline is functional.
This alert is always firing, therefore it should always be firing in Alertmanager
and always fire against a receiver. There are integrations with various notification
mechanisms that send a notification when this alert is not firing. For example the
"DeadMansSnitch" integration in PagerDuty.

1.8.5 自定义告警模板

首先修改 alertmanager-secret.yaml 添加自定义模板：

```
        app.kubernetes.io/part-of: kube-prometheus
        app.kubernetes.io/version: 0.21.0
      name: alertmanager-main
      namespace: monitoring
    stringData:
      wechat.tmpl: |-
        {{ define "wechat.default.message" }}
        {{- if gt (len .Alerts.Firing) 0 -}}
        {{- range $index, $alert := .Alerts -}}
        {{- if eq $index 0 }}
        ==========异常告警==========
        告警类型: {{ $alert.Labels.alertname }}
        告警级别: {{ $alert.Labels.severity }}
        告警详情:
{{ $alert.Annotations.message }}{{ $alert.Annotations.description}};{{$alert
.Annotations.summary}}
        故障时间: {{ ($alert.StartsAt.Add 28800e9).Format "2006-01-02
15:04:05" }}
        {{- if gt (len $alert.Labels.instance) 0 }}
        实例信息: {{ $alert.Labels.instance }}
        {{- end }}
        {{- if gt (len $alert.Labels.namespace) 0 }}
        命名空间: {{ $alert.Labels.namespace }}
        {{- end }}
        {{- if gt (len $alert.Labels.node) 0 }}
        节点信息: {{ $alert.Labels.node }}
        {{- end }}
        {{- if gt (len $alert.Labels.pod) 0 }}
        实例名称: {{ $alert.Labels.pod }}
        {{- end }}
        ============END============
        {{- end }}
        {{- end }}
        {{- end }}
        {{- if gt (len .Alerts.Resolved) 0 -}}
        {{- range $index, $alert := .Alerts -}}
        {{- if eq $index 0 }}
        ==========异常恢复==========
        告警类型: {{ $alert.Labels.alertname }}
        告警级别: {{ $alert.Labels.severity }}
        告警详情:
{{ $alert.Annotations.message }}{{ $alert.Annotations.description}};{{$alert
.Annotations.summary}}
        故障时间: {{ ($alert.StartsAt.Add 28800e9).Format "2006-01-02
15:04:05" }}
        恢复时间: {{ ($alert.EndsAt.Add 28800e9).Format "2006-01-02
15:04:05" }}
        {{- if gt (len $alert.Labels.instance) 0 }}
        实例信息: {{ $alert.Labels.instance }}
        {{- end }}
        {{- if gt (len $alert.Labels.namespace) 0 }}
        命名空间: {{ $alert.Labels.namespace }}
        {{- end }}
        {{- if gt (len $alert.Labels.node) 0 }}
        节点信息: {{ $alert.Labels.node }}
        {{- end }}
        {{- if gt (len $alert.Labels.pod) 0 }}
        实例名称: {{ $alert.Labels.pod }}
        {{- end }}
        ============END============
        {{- end }}
        {{- end }}
        {{- end }}
        {{- end }}
      alertmanager.yaml: |-
        "global":
          "resolve_timeout": "5m"
```

在 templates 字段添加模板位置：

```
        templates:
        - '/etc/alertmanager/config/*.tmpl'
        "inhibit_rules":
```

配置 wechat-ops receiver 使用该模板：

```
        "receivers":
        - name: wechat-ops
          wechat_configs:
          - send_resolved: true
            to_party: 3
            ...
            message: '{{ template "wechat.default.message" . }}'
```

注意

{{ template "wechat.default.message" . }}配置的 **wechat.default.message**，是模板文件 define 定义的名称：{{ define "wechat.default.message" }}，并非文件名称。

将配置更新至 Alertmanager：

```
# kubectl replace -f alertmanager-secret.yaml
secret/alertmanager-main replaced
```

更新完成后，可以在 Secret 中查看该配置：

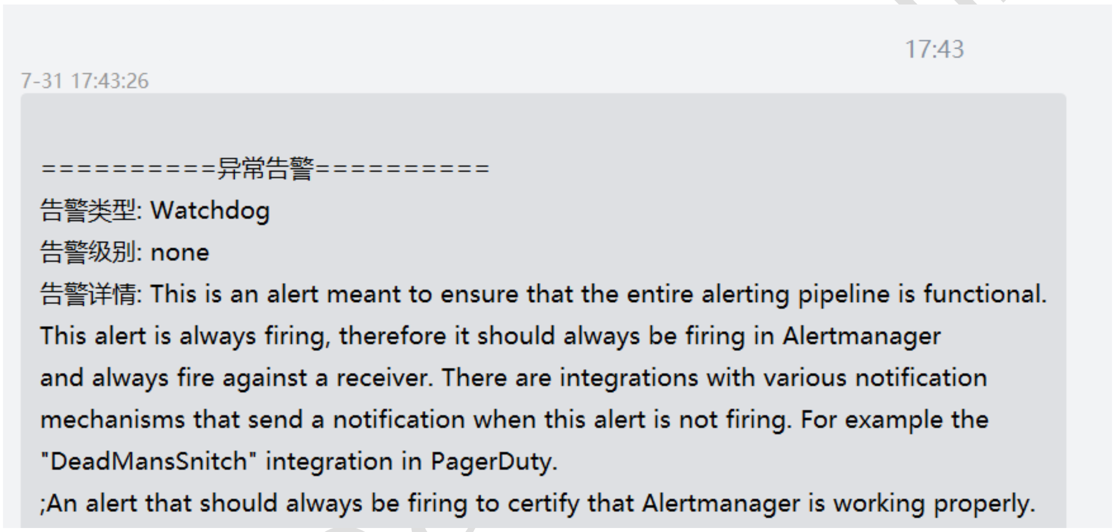
```
# kubectl describe secret alertmanager-main  -n monitoring
Name:         alertmanager-main
Namespace:    monitoring
Labels:       alertmanager=main
              app.kubernetes.io/component=alert-router
```

```
                app.kubernetes.io/name=alertmanager
                app.kubernetes.io/part-of=kube-prometheus
                app.kubernetes.io/version=0.21.0
    Annotations:  <none>

    Type:  Opaque

    Data
    ====
    alertmanager.yaml:  1438 bytes
    wechat.tmpl:        1823 bytes
```

等待几分钟后，可以在 alertmanager 的 Pod 中看到该模板：

```
    # kubectl exec  alertmanager-main-0 -n monitoring -c alertmanager  -- ls /etc/alertmanager/config
    alertmanager.yaml
    wechat.tmpl
```

之后再次收到告警，即为自定义告警模板：


17:43
7-31 17:43:26

==========异常告警==========
告警类型: Watchdog
告警级别: none
告警详情: This is an alert meant to ensure that the entire alerting pipeline is functional.
This alert is always firing, therefore it should always be firing in Alertmanager
and always fire against a receiver. There are integrations with various notification
mechanisms that send a notification when this alert is not firing. For example the
"DeadMansSnitch" integration in PagerDuty.
;An alert that should always be firing to certify that Alertmanager is working properly.

1.9 Prometheus 告警实战

AlertmanagerConfig: https://github.com/prometheus-operator/prometheus-operator/blob/master/example/user-guides/alerting/alertmanager-config-example.yaml
https://github.com/prometheus-operator/prometheus-operator/blob/master/Documentation/api.md#alertmanagerconfig

1.9.1 PrometheusRule

可以通过如下命令查看默认配置的告警策略：

```
    # kubectl get prometheusrule -n monitoring
    NAME                            AGE
    alertmanager-main-rules          19d
    kube-prometheus-rules           19d
    kube-state-metrics-rules         19d
    kubernetes-monitoring-rules       19d
    node-exporter-rules               19d
    prometheus-k8s-prometheus-rules   19d
    prometheus-operator-rules        19d
```

也可以通过-oyaml 查看某个 rules 的详细配置：

```
    # kubectl get prometheusrule -n monitoring node-exporter-rules    -oyaml
    apiVersion: monitoring.coreos.com/v1
    kind: PrometheusRule
    ...
    spec:
      groups:
      - name: node-exporter
        rules:
        - alert: NodeFilesystemSpaceFillingUp
          annotations:
            description: Filesystem on {{ $labels.device }} at
{{ $labels.instance }}
              has only {{ printf "%.2f" $value }}% available space left and is
filling
              up.
            runbook_url: https://github.com/prometheus-operator/kube-
prometheus/wiki/nodefilesystemspacefillingup
            summary: Filesystem is predicted to run out of space within the
next 24 hours.
          expr: |
            (
              node_filesystem_avail_bytes{job="node-exporter",fstype!=""} /
node_filesystem_size_bytes{job="node-exporter",fstype!=""} * 100 < 40
            and
              predict_linear(node_filesystem_avail_bytes{job="node-
exporter",fstype!=""}[6h], 24*60*60) < 0
            and
              node_filesystem_readonly{job="node-exporter",fstype!=""} == 0
            )
          for: 1h
          labels:
            severity: warning
```

- alert：告警策略的名称
- annotations：告警注释信息，一般写为告警信息
- expr：告警表达式
- for：评估等待时间，告警持续多久才会发送告警数据
- labels：告警的标签，用于告警的路由

1.9.2 域名访问延迟告警

假设需要对域名访问延迟进行监控，访问延迟大于 1 秒进行告警，此时可以创建一个 PrometheusRule 如下：

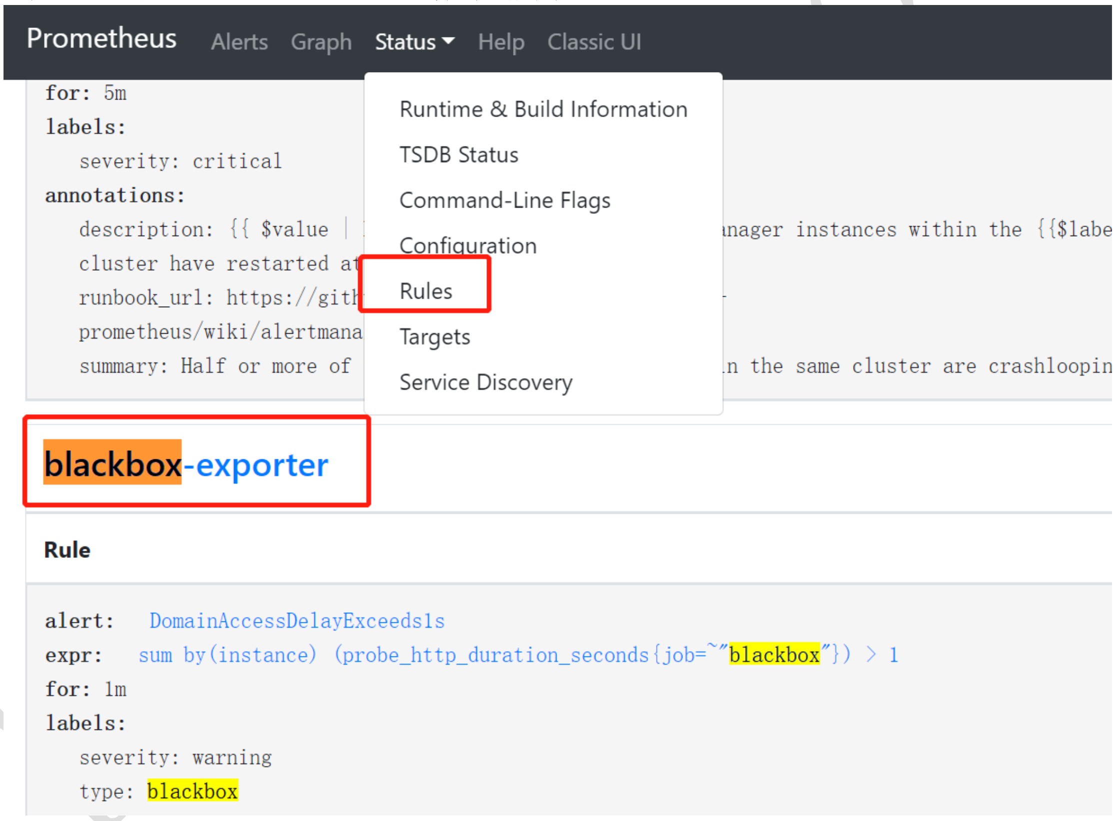
```
    # cat blackbox.yaml
    apiVersion: monitoring.coreos.com/v1
    kind: PrometheusRule
    metadata:
      labels:
        app.kubernetes.io/component: exporter
        app.kubernetes.io/name: blackbox-exporter
        prometheus: k8s
        role: alert-rules
      name: blackbox
      namespace: monitoring
    spec:
      groups:
      - name: blackbox-exporter
```

```
        rules:
        - alert: DomainAccessDelayExceeds1s
          annotations:
            description:  域名：{{ $labels.instance }} 探测延迟大于 1 秒，当前延迟
为：{{ $value }}
            summary: 域名探测，访问延迟超过 1 秒
          expr: sum(probe_http_duration_seconds{job=~"blackbox"}) by
(instance) > 1
          for: 1m
          labels:
            severity: warning
            type: blackbox
```

创建并查看该 PrometheusRule：

```
# kubectl create -f blackbox.yaml
prometheusrule.monitoring.coreos.com/blackbox created
# kubectl get -f blackbox.yaml
NAME       AGE
blackbox   65s
```

之后也可以在 Prometheus 的 Web UI 看到此规则：



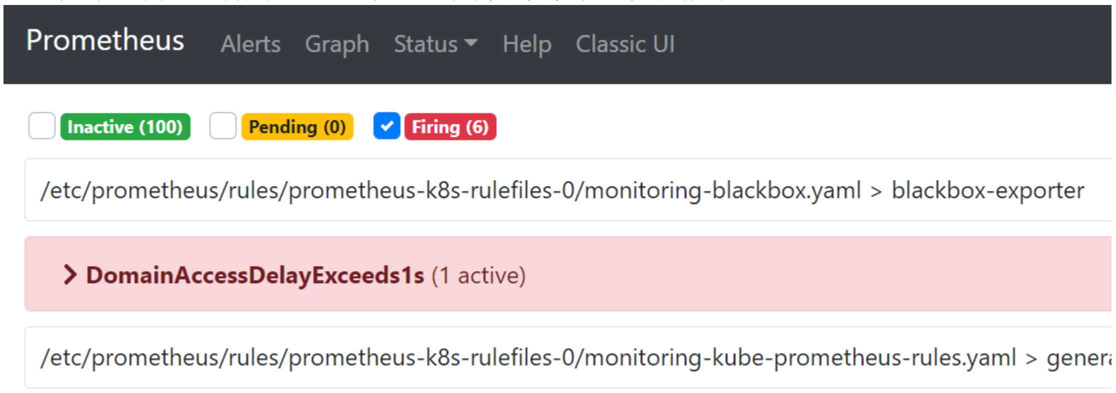
```
for: 5m
labels:
  severity: critical
annotations:
  description: {{ $value |                                   nager instances within the {{$labe
  cluster have restarted at
  runbook_url: https://gitl
  prometheus/wiki/alertmana
  summary: Half or more of                                   n the same cluster are crashloopin
```

blackbox-exporter

Rule

```
alert:  DomainAccessDelayExceeds1s
expr:  sum by(instance) (probe_http_duration_seconds{job=~"blackbox"}) > 1
for: 1m
labels:
  severity: warning
  type: blackbox
```

如果探测延迟有超过 1s 的域名，就会触发告警，如图所示：

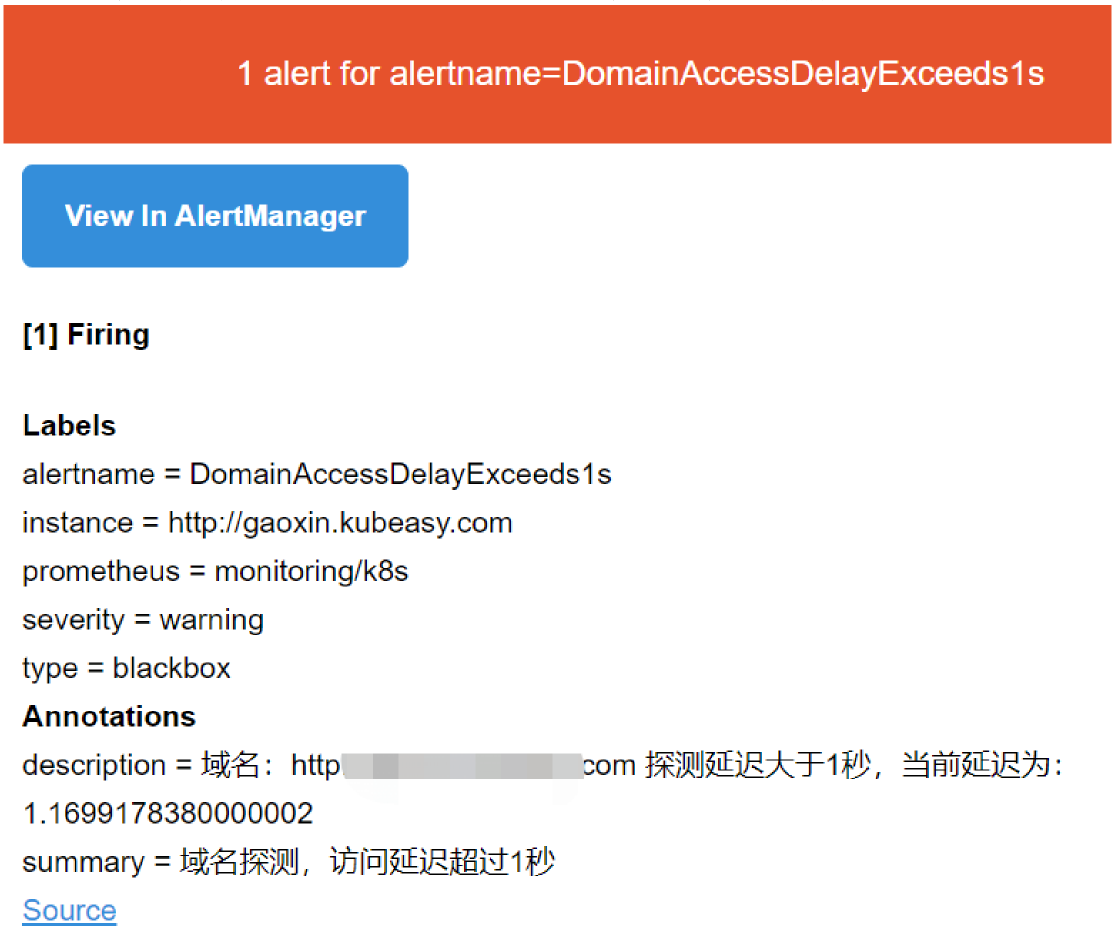


/etc/prometheus/rules/prometheus-k8s-rulefiles-0/monitoring-blackbox.yaml > blackbox-exporter

> DomainAccessDelayExceeds1s (1 active)

/etc/prometheus/rules/prometheus-k8s-rulefiles-0/monitoring-kube-prometheus-rules.yaml > gener

由于告警路由并未匹配黑盒监控的标签，所以会发送给默认的收件人，也就是邮箱：



1 alert for alertname=DomainAccessDelayExceeds1s

View In AlertManager

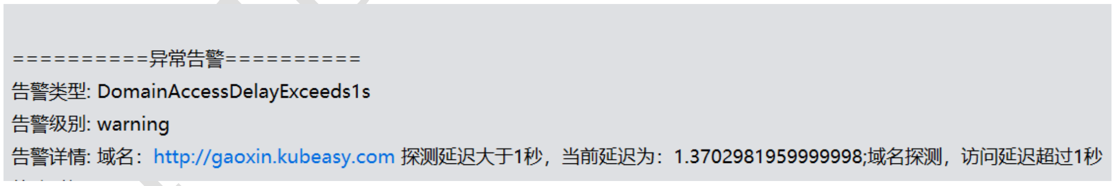
[1] Firing

Labels
alertname = DomainAccessDelayExceeds1s
instance = http://gaoxin.kubeasy.com
prometheus = monitoring/k8s
severity = warning
type = blackbox
Annotations
description = 域名：http[redacted]com 探测延迟大于1秒，当前延迟为：1.1699178380000002
summary = 域名探测，访问延迟超过1秒
Source

接下来可以根据实际业务情况将告警发送给指定的人，此时可以更改路由，将域名探测发送至微信，配置如下（部分代码）：

```
        - match:
            type: blackbox
          receiver: "wechat-ops"
          repeat_interval: 10m
```

之后在微信端即可收到告警：



==========异常告警==========
告警类型: DomainAccessDelayExceeds1s
告警级别: warning
告警详情: 域名：http://gaoxin.kubeasy.com 探测延迟大于1秒，当前延迟为：1.3702981959999998;域名探测，访问延迟超过1秒