一：node-exporter、cadvisor 和 prometheus server：

1.1：安装 cadvisor：

1.1.1：docker 部署 cadvisor：

```
# docker run \
--volume=/:/rootfs:ro \
--volume=/var/run:/var/run:ro \
--volume=/sys:/sys:ro \
--volume=/var/lib/docker/:/var/lib/docker:ro \
--volume=/dev/disk/:/dev/disk:ro \
--publish=8080:8080 \
--detach=true \
--name=cadvisor \
--privileged \
--device=/dev/kmsg \
gcr.io/cadvisor/cadvisor:v0.39.2
```

1.1.2：通过 daemonset 运行 cadvisor：

```
# docker tag gcr.io/cadvisor/cadvisor:v0.39.2 harbor.magedu.net/magedu/cadvisor:v0.39.2
# docker push harbor.magedu.net/magedu/cadvisor:v0.39.2
# kubectl   create   ns monitoring
# vim case1-daemonset-deploy-cadvisor.yaml

# kubectl apply   -f case1-daemonset-deploy-cadvisor.yaml
# kubectl   get pod   -n monitoring -o wide
NAME                READY   STATUS    RESTARTS   AGE     IP              NODE              NOMINATED  NODE
READINESS GATES
cadvisor-5tj6d      1/1     Running   0          2m35s   172.31.7.101    172.31.7.101    <none>           <none>
cadvisor-7mnk9      1/1     Running   0          2m35s   172.31.7.112    172.31.7.112    <none>           <none>
cadvisor-c5mb9      1/1     Running   0          2m35s   172.31.7.102    172.31.7.102    <none>           <none>
cadvisor-djbsd      1/1     Running   0          2m35s   172.31.7.113    172.31.7.113    <none>           <none>
cadvisor-gz62f      1/1     Running   0          2m35s   172.31.7.111    172.31.7.111    <none>           <none>
cadvisor-jkwgh      1/1     Running   0          2m35s   172.31.7.103    172.31.7.103    <none>           <none>
```

1.1.3：cadvisor 指标数据：

指标名称	类型	含义
container_cpu_load_average_10s	gauge	过去 10 秒容器 CPU 的平均负载
container_cpu_usage_seconds_total	counter	容器在每个 CPU 内核上的累积占用时间（单位：秒）
container_cpu_system_seconds_total	counter	System CPU 累积占用时间（单位：秒）
container_cpu_user_seconds_total	counter	User CPU 累积占用时间（单位：秒）
container_fs_usage_bytes	gauge	容器中文件系统的使用量(单位：字节)
container_fs_limit_bytes	gauge	容器可以使用的文件系统总量(单

位：字节)

container_fs_reads_bytes_total	counter	容器累积读取数据的总量(单位：字节)
container_fs_writes_bytes_total	counter	容器累积写入数据的总量(单位：字节)
container_memory_max_usage_bytes	gauge	容器的最大内存使用量（单位：字节）
container_memory_usage_bytes	gauge	容器当前的内存使用量（单位：字节
container_spec_memory_limit_bytes	gauge	容器的内存使用量限制
machine_memory_bytes	gauge	当前主机的内存总量
container_network_receive_bytes_total	counter	容器网络累积接收数据总量（单位：字节）
container_network_transmit_bytes_total	counter	容器网络累积传输数据总量（单位：字节）

当能够正常采集到 cAdvisor 的样本数据后，可以通过以下表达式计算容器的 CPU 使用率：

（1）sum(irate(container_cpu_usage_seconds_total{image!=""}[1m])) without (cpu)

容器 CPU 使用率

（2）container_memory_usage_bytes{image!=""}

查询容器内存使用量（单位：字节）：

（3）sum(rate(container_network_receive_bytes_total{image!=""}[1m])) without (interface)

查询容器网络接收量（速率）（单位：字节/秒）：

（4）sum(rate(container_network_transmit_bytes_total{image!=""}[1m])) without (interface)

容器网络传输量 字节/秒

（5）sum(rate(container_fs_reads_bytes_total{image!=""}[1m])) without (device)

容器文件系统读取速率 字节/秒

（6）sum(rate(container_fs_writes_bytes_total{image!=""}[1m])) without (device)

容器文件系统写入速率 字节/秒

cadvisor 常用容器监控指标

（1）网络流量

```
sum(rate(container_network_receive_bytes_total{name=~".+"}[1m])) by (name)
 ##容器网络接收的字节数（1 分钟内），根据名称查询 name=~".+"

sum(rate(container_network_transmit_bytes_total{name=~".+"}[1m])) by (name)
 ##容器网络传输的字节数（1 分钟内），根据名称查询 name=~".+"
```

（2）容器 CPU 相关

```
sum(rate(container_cpu_system_seconds_total[1m]))
###所用容器 system cpu 的累计使用时间（1min 钟内）

sum(irate(container_cpu_system_seconds_total{image!=""}[1m])) without (cpu)
```

```
###每个容器 system cpu 的使用时间（1min 钟内）

sum(rate(container_cpu_usage_seconds_total{name=~".+"}[1m])) by (name) * 100

#每个容器的 cpu 使用率


sum(sum(rate(container_cpu_usage_seconds_total{name=~".+"}[1m])) by (name) * 100)
#总容器的 cpu 使用率
```

1.2：node-exporter：

1.2.1：docker 部署 node-exporter：

```
# docker run -d \
--net="host" \
--pid="host" \
-v "/:/host:ro,rslave" \
quay.io/prometheus/node-exporter:latest \
--path.rootfs=/host
```

1.2.2：daemonset 运行 node-exporter：

```
# docker tag quay.io/prometheus/node-exporter:v1.3.1 harbor.magedu.net/magedu/node-exporter:v1.3.1
# docker push harbor.magedu.net/magedu/node-exporter:v1.3.1

# vim case2-daemonset-deploy-node-exporter.yaml

# kubectl apply    -f case2-daemonset-deploy-node-exporter.yaml
# kubectl    get pod -n monitoring -o wide
NAME                          READY     STATUS       RESTARTS     AGE       IP                NODE              NOMINATED NODE     READINESS GATES
node-exporter-49bld     1/1       Running     0              65s       172.31.7.101     172.31.7.101     <none>              <none>
node-exporter-8wvf2     1/1       Running     0              65s       172.31.7.102     172.31.7.102     <none>              <none>
node-exporter-hkz6m     1/1       Running     0              65s       172.31.7.103     172.31.7.103     <none>              <none>
node-exporter-jgzvn     1/1       Running     0              65s       172.31.7.113     172.31.7.113     <none>              <none>
node-exporter-mn9xm     1/1       Running     0              65s       172.31.7.111     172.31.7.111     <none>              <none>
node-exporter-pk2f2     1/1       Running     0              65s       172.31.7.112     172.31.7.112     <none>              <none>
```

1.2.3：node-exporter 指标数据：

```
# curl    172.30.7.111:9100/metrics
#node-export 默认的监听端口为 9100，可以使用浏览器或 curl 访问数据：
# HELP node_cpu_guest_seconds_total Seconds the CPUs spent in guests (VMs) for each mode.
# TYPE node_cpu_guest_seconds_total counter
node_cpu_guest_seconds_total{cpu="0",mode="nice"} 0
node_cpu_guest_seconds_total{cpu="0",mode="user"} 0
# HELP node_cpu_seconds_total Seconds the CPUs spent in each mode.
```

```
# TYPE node_cpu_seconds_total counter
node_cpu_seconds_total{cpu="0",mode="idle"} 4168.02
node_cpu_seconds_total{cpu="0",mode="iowait"} 0.84
node_cpu_seconds_total{cpu="0",mode="irq"} 0
node_cpu_seconds_total{cpu="0",mode="nice"} 1.96
node_cpu_seconds_total{cpu="0",mode="softirq"} 9.33
node_cpu_seconds_total{cpu="0",mode="steal"} 0
node_cpu_seconds_total{cpu="0",mode="system"} 167.92
node_cpu_seconds_total{cpu="0",mode="user"} 290.23

常见的指标:
node_boot_time：系统自启动以后的总结时间
node_cpu：系统 CPU 使用量
nodedisk*：磁盘 IO
nodefilesystem*：系统文件系统用量
node_load1：系统 CPU 负载
nodememeory*：内存使用量
nodenetwork*：网络带宽指标
node_time：当前系统时间
go_*：node exporter 中 go 相关指标
process_*：node exporter 自身进程相关运行指标
```

1.3：prometheus server：

1.3.1：docker 部署 prometheus server：

```
# docker run \
    -p 9090:9090 \
    -v /path/to/config:/etc/prometheus \
    prom/prometheus
```

1.3.2：yaml 部署 prometheus server：

1.3.2.1：prometheus cfg：

```
# kubectl apply    -f case3-1-prometheus-cfg.yaml
```

1.3.2.2：prometheus deployment：

```
# docker pull prom/prometheus:v2.31.2


root@k8s-node3:~# mkdir    -p /data/prometheusdata
# chmod    777 /data/prometheusdata
#将 prometheus 运行在 node3 并，提前准备数据目录并授权
# kubectl    create    serviceaccount    monitor -n monitoring #创建监控账号
#    kubectl    create    clusterrolebinding    monitor-clusterrolebinding    -n    monitoring    --clusterrole=cluster-admin
--serviceaccount=monitoring:monitor #对 monitoring  账号授权
```

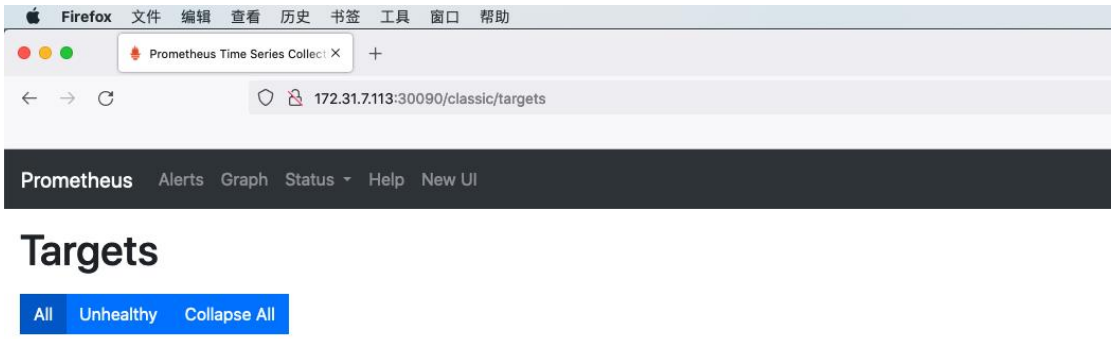
```
# kubectl apply    -f case3-2-prometheus-deployment.yaml
deployment.apps/prometheus-server created

# kubectl    get pod -n monitoring
NAME                                 READY    STATUS              RESTARTS    AGE
prometheus-server-7547cc665-dlzth    1/1      Running    0            10s
```

1.3.3：prometheus svc：

```
# kubectl apply    -f case3-3-prometheus-svc.yaml
service/prometheus created
```

1.3.4：验证 prometheus server：

```
# kubectl    get svc -n monitoring
NAME                TYPE            CLUSTER-IP        EXTERNAL-IP     PORT(S)              AGE
node-exporter       ClusterIP       10.0.133.100      <none>          9100/TCP             27m
prometheus          NodePort        10.0.45.238       <none>          9090:30090/TCP       28s
```



二：prometheus 的服务发现机制：

prometheus 默认是采用 pull 方式拉取监控数据的，也就是定时去目标主机上抓取 metrics 数据，每一个被抓取的目标需要暴露一个 HTTP 接口，prometheus 通过这个暴露的接口就可以获取到相应的指标数据，这种方式需要由目标服务决定采集的目标有哪些，通过配置在 scrape_configs 中的各种 job 来实现，无法动态感知新服务，如果后面增加了节点或者组件信息，就得手动修 promrtheus 配置，并重启 promethues，很不方便，所以出现了动态服务发现，动态服务发现能够自动发现集群中的新端点，并加入到配置中，通过服务发现，Prometheus 能查询到需要监控的 Target 列表，然后轮询这些 Target 获取监控数据。

prometheus 获取数据源 target 的方式有多种，如静态配置和服务发现配置，prometheus 支持多种服务发现，prometheus 目前支持的服务发现有很多种，常

用的主要分为以下几种：

kubernetes_sd_configs: #Kubernetes 服务发现，让 prometheus 动态发现 kubernetes 中被监控的目标

static_configs: #静态服务发现，基于 prometheus 配置文件指定的监控目标

dns_sd_configs: #DNS 服务发现监控目标

consul_sd_configs: #Consul 服务发现，基于 consul 服务动态发现监控目标

file_sd_configs: #基于指定的文件实现服务发现，基于指定的文件发现监控目标

promethues 的静态静态服务发现 static_configs：每当有一个新的目标实例需要监控，都需要手动修改配置文件配置目标 target。

promethues 的 consul 服务发现 consul_sd_configs：Prometheus 一直监视 consul 服务，当发现在 consul 中注册的服务有变化，prometheus 就会自动监控到所有注册到 consul 中的目标资源。

promethues 的 k8s 服务发现 kubernetes_sd_configs：Prometheus 与 Kubernetes 的 API 进行交互，动态的发现 Kubernetes 中部署的所有可监控的目标资源。

2.1：kubernetes_sd_configs：

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

promethues 的 relabeling（重新修改标签）功能很强大，它能够在抓取到目标实例之前把目标实例的元数据标签动态重新修改，动态添加或者覆盖标签

prometheus 加载 target 成功之后，在 Target 实例中，都包含一些 Metadata 标签信息，默认的标签有：

__address__：以<host>:<port> 格式显示目标 targets 的地址

__scheme__：采集的目标服务地址的 Scheme 形式，HTTP 或者 HTTPS

__metrics_path__：采集的目标服务的访问路径

2.1.1：基础功能-重新标记目的：

为了更好的识别监控指标,便于后期调用数据绘图、告警等需求，prometheus 支持对发现的目标进行 label 修改，在两个阶段可以重新标记：

relabel_configs ： 在采集之前（比如在采集数据之前重新定义元标签），可以使用 relabel_configs 添加一些标签、也可以只采集特定目标或过滤目标。

metric_relabel_configs：如果是已经抓取到指标数据时，可以使用 metric_relabel_configs 做最后的重新标记和过滤。


```
- job_name: 'kubernetes-apiserver' #job 名称
  kubernetes_sd_configs:  #基于 kubernetes_sd_configs 实现服务发现
  - role: endpoints #发现 endpoints
  scheme: https #当前 jod 使用的发现协议
  tls_config: #证书配置
    ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt #容器里的证书路径
  bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token #容器里的 token 路径
  relabel_configs: #重新 re 修改标签 label 配置 configs
  - source_labels: [__meta_kubernetes_namespace, __meta_kubernetes_service_name, __meta_kubernetes_endpoint_port_name] #源标签
```

```
action: keep #action 定义了 relabel 的具体动作，action 支持多种
        regex: default;kubernetes;https #发现了 default 命名空间的 kubernetes 服务切是 https 协议
```

2.1.2：label 详解：

source_labels：源标签，没有经过 relabel 处理之前的标签名字

target_label：通过 action 处理之后的新的标签名字

regex：正则表达式，匹配源标签

replacement：通过分组替换后标签（target_label）对应的值

2.1.3：action 详解：

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

replace：替换标签值，根据 regex 正则匹配到源标签的值，使用 replacement 来引用表达式匹配的分组

keep：满足 regex 正则条件的实例进行采集，把 source_labels 中没有匹配到 regex 正则内容的 Target 实例丢掉，即只采集匹配成功的实例。

drop： 满足 regex 正则条件的实例不采集，把 source_labels 中匹配到 regex 正则内容的 Target 实例丢掉，即只采集没有匹配到的实例。

hashmod：使用 hashmod 计算 source_labels 的 Hash 值并进行对比，基于自定义的模数取模，以实现对目标进行分类、重新赋值等功能：

```
scrape_configs:
  - job_name: ip_job
    relabel_configs:
    - source_labels: [__address__]
      modulus:       4
      target_label:  __ip_hash
      action:        hashmod
    - source_labels: [__ip_hash]
      regex:         ^1$
      action:        keep
```

labelmap：匹配 regex 所有标签名称,然后复制匹配标签的值进行分组，通过 replacement 分组引用（${1},${2},...）替代

labelkeep：匹配 regex 所有标签名称,其它不匹配的标签都将从标签集中删除

labeldrop：匹配 regex 所有标签名称,其它匹配的标签都将从标签集中删除

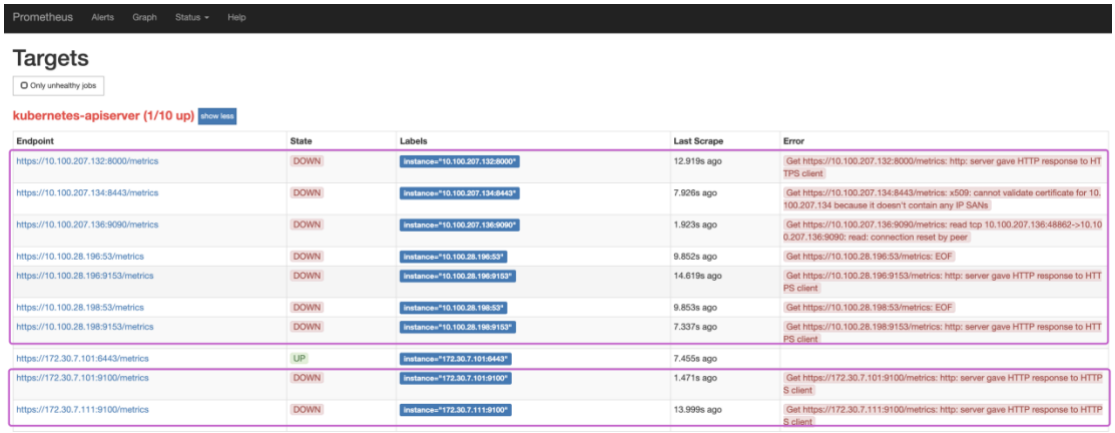
2.1.4：测试 label 功能：

```
# vim case3-1-prometheus-cfg.yaml
 50 #         relabel_configs:
 51 #           - source_labels: [__meta_kubernetes_namespace, __meta_kubernetes_service_name, __meta_kubernetes_endpoint_port_name]
 52 #             action: keep
 53 #             regex: default;kubernetes;https #删除 50-53

# kubectl apply -f case3-1-prometheus-cfg.yaml
```

```
configmap/prometheus-config configured

# kubectl   delete   -f case3-2-prometheus-deployment.yaml   #重新加载配置
deployment.apps "prometheus-server" deleted
# kubectl   apply   -f case3-2-prometheus-deployment.yaml
deployment.apps/prometheus-server created
```

如下图：

好多 down 的，是因为做服务发现的时候没有进行过滤，pod 也被匹配成功并进行监控，但是 pod 并没有提供 metrics 指标数据，所以不通：



2.1.5：查看 prometheus server 容器证书：

```
# kubectl   get pod -n monitoring

# kubectl   exec -it prometheus-server-7547cc665-gqzbx ls /var/run/secrets/kubernetes.io/serviceaccount/   -n monitoring
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
ca.crt          namespace    token
```

2.1.6：支持的发现目标类型：

发现类型可以配置以下类型之一来发现目标：

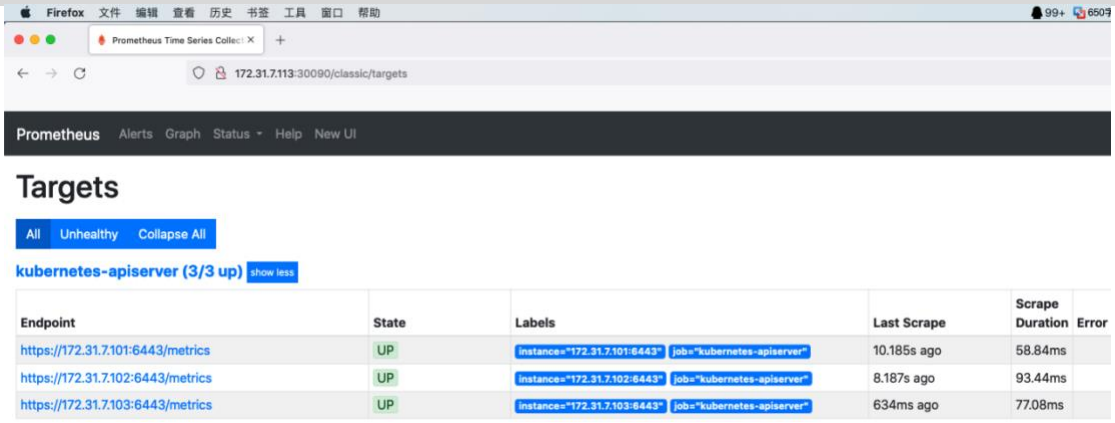
https://prometheus.io/docs/prometheus/latest/configuration/configuration/#kubernetes_sd_config

```
node
service
pod
endpoints
Endpointslice #对 endpoint 进行切片
ingress
```

2.1.7：监控 api-server 实现：

apiserver 作为 Kubernetes 最核心的组件，它的监控也是非常有必要的，对于 apiserver 的监控，我们可以直接通过 kubernetes 的 service 来获取：

```
# kubectl   get svc
NAME             TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)     AGE
kubernetes       ClusterIP      10.0.0.1       <none>          443/TCP     113d

#    kubectl    get ep
NAME             ENDPOINTS                                                  AGE
kubernetes       172.31.7.101:6443,172.31.7.102:6443,172.31.7.103:6443      113d

# vim case3-1-prometheus-cfg.yaml
      - job_name: 'kubernetes-apiserver'
        kubernetes_sd_configs:
        - role: endpoints
        scheme: https
        tls_config:
          ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
        relabel_configs:
        -          source_labels:          [__meta_kubernetes_namespace,          __meta_kubernetes_service_name,
__meta_kubernetes_endpoint_port_name]
          action: keep
          regex: default;kubernetes;https #含义为匹配 default 的 namespace，svc 名称是 kubernetes 并且协议是 https，匹配成功后进行保留，并且把 regex 作为 source_labels 相对应的值。即 labels 为 key、regex 为值。label 替换方式如下：
#__meta_kubernetes_namespace=default,__meta_kubernetes_service_name=kubernetes,__meta_kubernetes_endpoint_port_name=https
最终，匹配到 api-server 的地址:
# kubectl   get ep
NAME             ENDPOINTS                                                  AGE
kubernetes       172.31.7.101:6443,172.31.7.102:6443,172.31.7.103:6443      113d
```



2.1.8：api-server 指标数据：

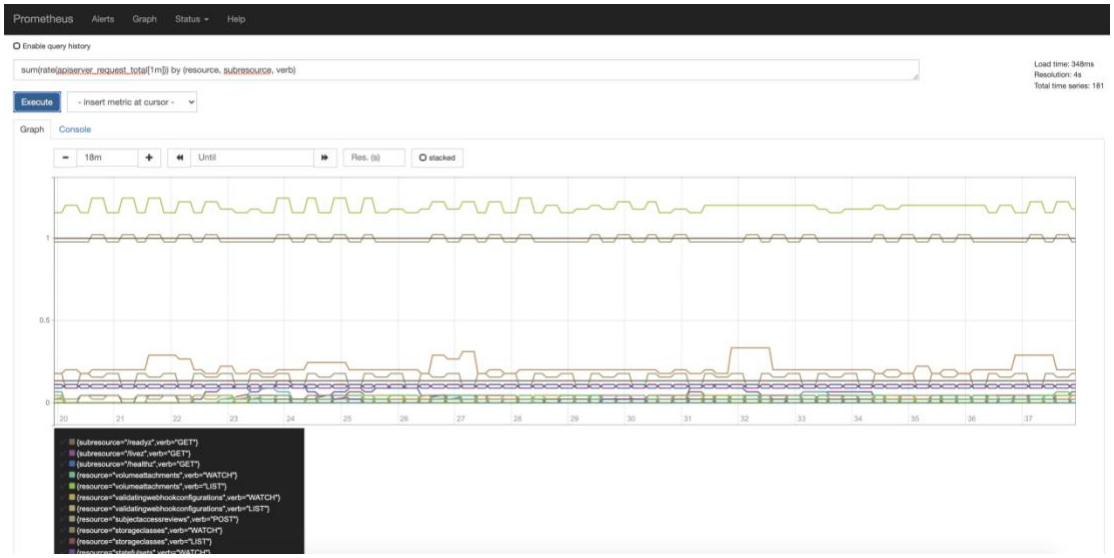
Apiserver 组件是 k8s 集群的入口，所有请求都是从 apiserver 进来的，所以对

apiserver 指标做监控可以用来判断集群的健康状况。

2.1.8.1：apiserver_request_total：

以下 promQL 语句为查询 apiserver 最近一分钟不同方法的请求数量统计：

apiserver_request_total 为请求各个服务的访问详细统计：

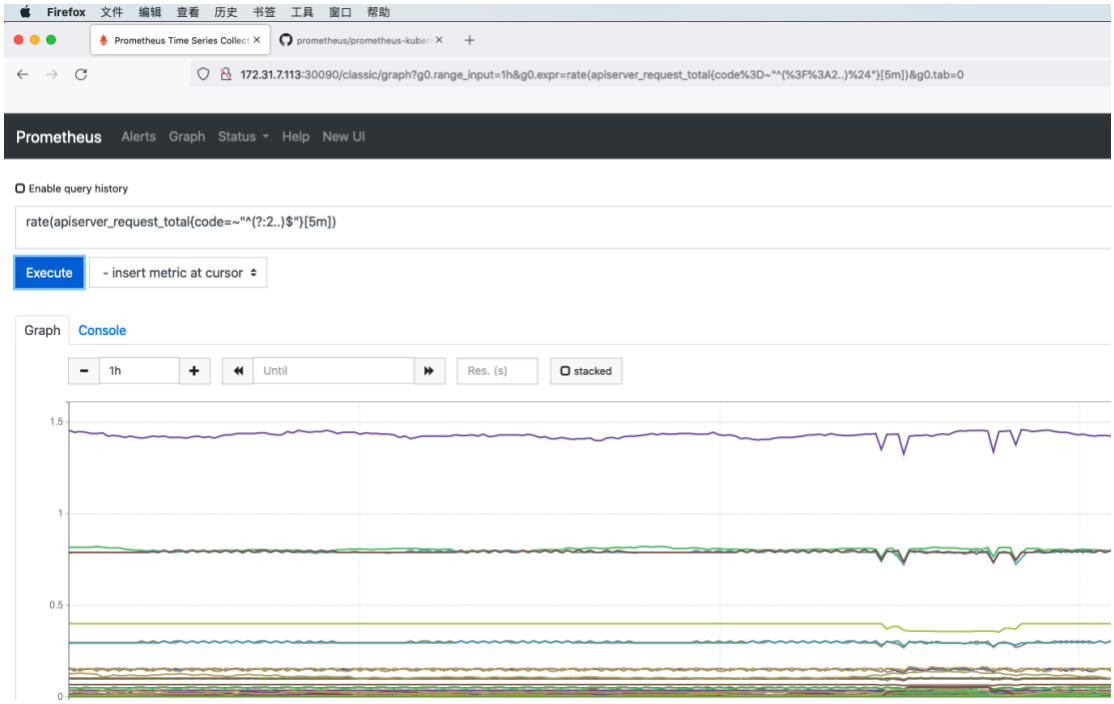


irate 和 rate 都会用于计算某个指标在一定时间间隔内的变化速率。但是它们的计算方法有所不同：irate 取的是在指定时间范围内的最近两个数据点来算速率，而 rate 会取指定时间范围内所有数据点，算出一组速率，然后取平均值作为结果。

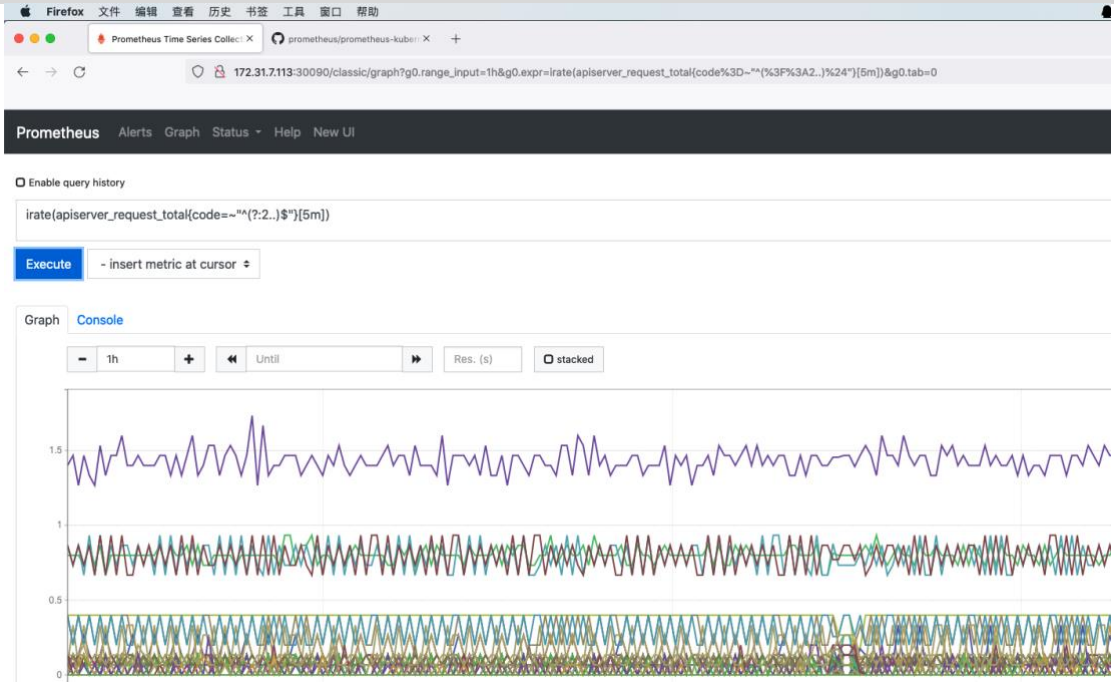
所以官网文档说：irate 适合快速变化的计数器（counter），而 rate 适合缓慢变化的计数器（counter）。

根据以上算法我们也可以理解，对于快速变化的计数器，如果使用 rate，因为使用了平均值，很容易把峰值削平。除非我们把时间间隔设置得足够小，就能够减弱这种效应。

```
rate(apiserver_request_total{code=~"^(?:2..)$"}[5m])
```



irate(apiserver_request_total{code=~"^(?:2..)$"}[5m])



2.1.8.2：关于 annotation_prometheus_io_scrape：

在 k8s 中，基于 prometheus 的发现规则，需要在被发现的目的 target 定义注解匹配 annotation_prometheus_io_scrape=true，且必须匹配成功该注解才会保留监控 target，然后再进行数据抓取并进行标签替换，如 annotation_prometheus_io_scheme 标签为 http 或 https：

```
    - job_name: 'kubernetes-service-endpoints' #job 名称
      kubernetes_sd_configs: #sd_configs 发现
      - role: endpoints #角色，endpoints 发现
```

```
relabel_configs: #标签重写配置

#annotation_prometheus_io_scrape 的值为 true，保留标签然后再向下执行
      - source_labels: [__meta_kubernetes_service_annotation_prometheus_io_scrape]
        action: keep
        regex: true

# 将 __meta_kubernetes_service_annotation_prometheus_io_scheme 修改为 __scheme__
      - source_labels: [__meta_kubernetes_service_annotation_prometheus_io_scheme]
        action: replace
        target_label: __scheme__
        regex: (https?) #正则匹配协议 http 或 https，即其它协议不替换

#将 source_labels 替换为 target_label，target_label
      - source_labels: [__meta_kubernetes_service_annotation_prometheus_io_path]
        action: replace
        target_label: __metrics_path__
        regex: (.+)  #路径为为 1 到任意长度

#地址发现及标签重写
      - source_labels: [__address__, __meta_kubernetes_service_annotation_prometheus_io_port]
        action: replace
        target_label: __address__
        regex: ([^:]+)(?::\d+)?;(\d+)
        replacement: $1:$2 #格式为地址:端口

#发现新的 lable 并用新的 service name 作为 label、将发现的值依然新的 label 的值
      - action: labelmap
        regex: __meta_kubernetes_service_label_(.+) #通过正则匹配 service name

#将__meta_kubernetes_namespace 替换为 kubernetes_namespace
      - source_labels: [__meta_kubernetes_namespace]
        action: replace
        target_label: kubernetes_namespace

#将__meta_kubernetes_service_name 替换为 kubernetes_name
      - source_labels: [__meta_kubernetes_service_name]
```

```
action: replace
        target_label: kubernetes_name
```

2.1.9：kube-dns 的服务发现：

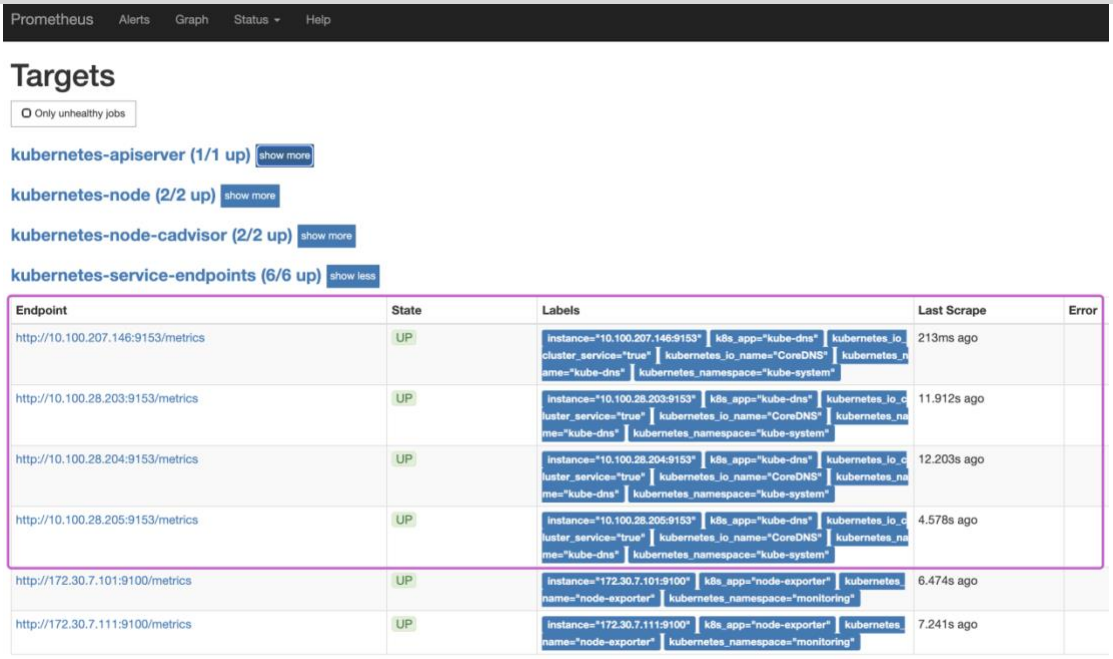
2.1.9.1：查看 kube-dns 状态：

```
# kubectl    describe    svc kube-dns    -n kube-system
Name:                          kube-dns
Namespace:                     kube-system
Labels:                        k8s-app=kube-dns
                               kubernetes.io/cluster-service=true
                               kubernetes.io/name=CoreDNS
Annotations:                   field.cattle.io/publicEndpoints:

[{"addresses":["172.31.7.101"],"port":44081,"protocol":"UDP","serviceName":"kube-system:kube-dns","allNodes
":true},{"addresses":["172.31.7...
                               prometheus.io/port: 9153    #注解标签，用于 prometheus 服务发现端口
                               prometheus.io/scrape: true    #运行 prometheus 抓取数据
Selector:                      k8s-app=kube-dns
Type:                          NodePort
IP Family Policy:              SingleStack
IP Families:                   IPv4
IP:                            10.0.0.2
IPs:                           10.0.0.2
Port:                          dns    53/UDP
TargetPort:                    53/UDP
NodePort:                      dns    44081/UDP
Endpoints:                     10.200.2.64:53
Port:                          dns-tcp    53/TCP
TargetPort:                    53/TCP
NodePort:                      dns-tcp    44081/TCP
Endpoints:                     10.200.2.64:53
Port:                          metrics    9153/TCP
TargetPort:                    9153/TCP
NodePort:                      metrics    30009/TCP
Endpoints:                     10.200.2.64:9153
Session Affinity:              None
External Traffic Policy:    Cluster
Events:                        <none
```

2.1.9.2：修改副本数验证发现：

新添加一个 node 节点或修改 deployment 控制器的副本数，以让 endpoint 数量发生变化，验证能否自动发现新添加的 pod：

```
# kubectl    edit deployments coredns -n kube-system
2
```

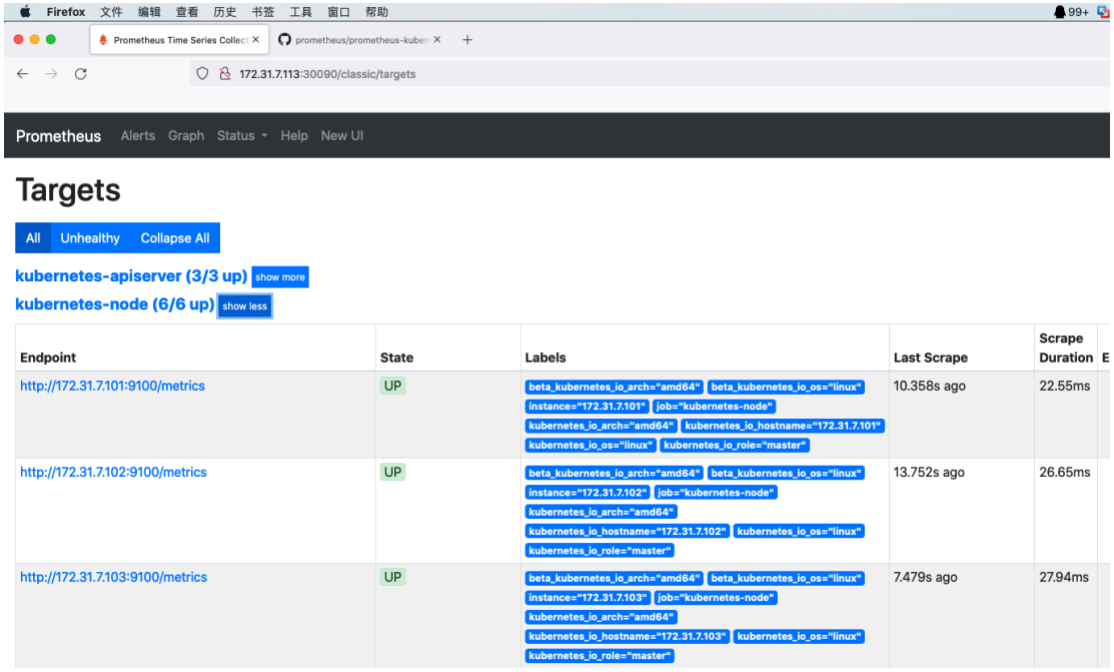


2.1.10：node 节点发现及指标：

2.1.10.1：配置详解：

```
kind: ConfigMap
apiVersion: v1
metadata:
  labels:
    app: prometheus
  name: prometheus-config
  namespace: monitor-sa
data:
  prometheus.yml: |
    global:
      scrape_interval: 15s
      scrape_timeout: 10s
      evaluation_interval: 1m
    scrape_configs:
    - job_name: 'kubernetes-node' #job name
      kubernetes_sd_configs: #发现配置
      - role: node #发现角色
      relabel_configs: #标签重写配置

      - source_labels: [__address__] #源标签
        regex: '(.*):10250' #通过正则匹配后缀为:10250 的实例，10250 是 kubelet 端口
```

```
replacement: '${1}:9100' #重写为 IP:9100,即将端口替换为 prometheus node-exporter 的端口
        target_label: __address__ #将[__address__]替换为__address__
        action: replace    #将[__address__] 的值依然赋值给__address__

#发现新的 lable 并用新的 service name 作为 label、将发现的值依然新的 label 的值
    - action: labelmap
        regex: __meta_kubernetes_node_label_(.+)
```



2.1.10.2：kubelet 监听端口：

```
# lsof   -i:10250
COMMAND PID USER     FD     TYPE DEVICE SIZE/OFF NODE NAME
kubelet 866 root     10u    IPv6   47583      0t0    TCP k8s-node1:10250->k8s-master1:47644 (ESTABLISHED)
kubelet 866 root     29u    IPv6   27079      0t0    TCP *:10250 (LISTEN)
```

2.1.10.3：node 节点指标数据：

```
# curl   http://172.30.7.111:9100/metrics


#HELP：解释当前指标的含义，上面表示在每种模式下 node 节点的 cpu 花费的时间，以 s 为单位
#TYPE：说明当前指标的数据类型,如:
# HELP node_load1 1m load average.
# TYPE node_load1 gauge
node_load1 0.49
# HELP node_load15 15m load average.
# TYPE node_load15 gauge
node_load15 0.72
# HELP node_load5 5m load average.
```

```
# TYPE node_load5 gauge
node_load5 0.54
```

2.1.10.4：常见监控指标：

```
node_cpu_：CPU 相关指标
node_load1：load average #系统负载指标
node_load5
node_load15

node_memory_：内存相关指标

node_network_：网络相关指标

node_disk_：磁盘 IO 相关指标
node_filesystem_：文件系统相关指标

node_boot_time_seconds：系统启动时间监控

go_*：node exporte 运行过程中 go 相关指标
process_*：node exporter 运行时进程内部进程指标
```

2.1.11：pod 发现及指标数据：

2.1.11.1：prometheus job 配置：

```
    - job_name: 'kubernetes-node-cadvisor'   #job 名称
      kubernetes_sd_configs:   #基于 k8s 的服务发现
      - role:   node #角色
      scheme: https #协议
      tls_config: #证书配置
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt #默认证书路径
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token #默认 token 路径
      relabel_configs: #标签重写配置
      - action: labelmap
        regex: __meta_kubernetes_node_label_(.+)

#replacement 指定的替换后的标签（target_label）对应的值为 kubernetes.default.svc:443
      - target_label: __address__
        replacement: kubernetes.default.svc:443

#将[__meta_kubernetes_node_name]重写为 __metrics_path__
      - source_labels: [__meta_kubernetes_node_name]
        regex: (.+) #至少 1 位长度以上
        target_label: __metrics_path__
        replacement: /api/v1/nodes/${1}/proxy/metrics/cadvisor #指定值
```

tls_config 配置的证书地址是每个 Pod 连接 apiserver 所使用的地址，无论证书是否用得上，在 Pod 启动的时候 kubelet 都会给每一个 pod 自动注入 ca 的公钥，即所有的 pod 启动的时候都会有一个 ca 公钥被注入进去用于在访问 apiserver 的时候被调用。

```
# kubectl    exec    test-pod1    ls /var/run/secrets/kubernetes.io/serviceaccount/
ca.crt
namespace
token

# kubectl    exec    test-pod1 md5sum    /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
dc631be0bb3e4ca96cb9c671e607aa89    /var/run/secrets/kubernetes.io/serviceaccount/ca.crt

]# md5sum    /etc/kubernetes/pki/ca.crt
dc631be0bb3e4ca96cb9c671e607aa89    /etc/kubernetes/pki/ca.crt
[root@k8s-master1 yaml]#
```

2.1.11.2：pod 监控指标：

CPU

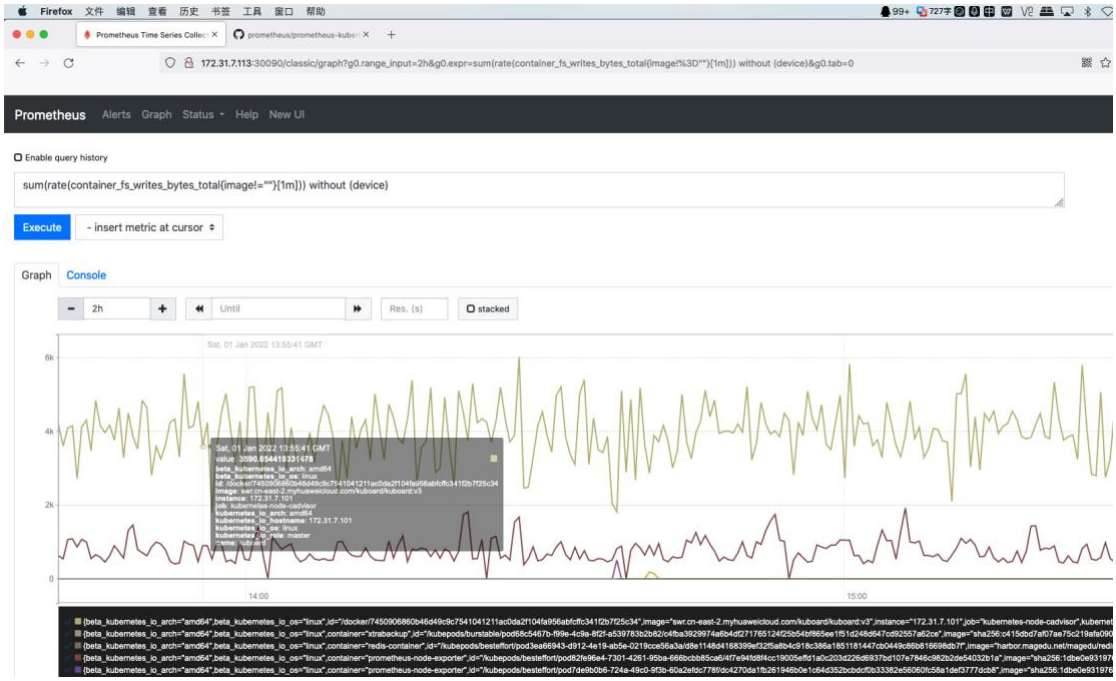
内存

磁盘

网卡

```
sum(rate(container_cpu_usage_seconds_total{image!=""}[1m])) without (instance)

sum(rate(container_memory_usage_bytes{image!=""}[1m])) without (instance)

sum(rate(container_fs_io_current{image!=""}[1m])) without (device)
sum(rate(container_fs_writes_bytes_total{image!=""}[1m])) without (device)
sum(rate(container_fs_reads_bytes_total{image!=""}[1m])) without (device)

sum(rate(container_network_receive_bytes_total{image!=""}[1m])) without (interface)
```

2.1.11.2：验证数据：



2.1.12：prometheus 部署在 k8s 集群以外并实现服务发现：

2.1.12.1：创建用户并授权：

```
# kubectl apply    -f case4-prom-rbac.yaml
```

2.1.12.2：获取 token：

```
# kubectl get sa prometheus    -n monitoring    -o yaml
apiVersion: v1
kind: ServiceAccount
metadata:
   annotations:
      kubectl.kubernetes.io/last-applied-configuration: |

{"apiVersion":"v1","kind":"ServiceAccount","metadata":{"annotations":{},"name":"prometheus","namespace":"m
onitoring"}}
   creationTimestamp: "2022-01-01T15:47:09Z"
   name: prometheus
   namespace: monitoring
   resourceVersion: "1479441"
   uid: a74a1b0b-69c3-4b53-8a86-cad9e1d078ce
secrets:
- name: prometheus-token-ff8pt

将 token 保存至 prometheus server 节点的 k8s.token 文件，后期用于权限验证。
#    kubectl    describe secret prometheus-token-ff8pt -n monitoring
# vim /apps/prometheus/k8s.token
```

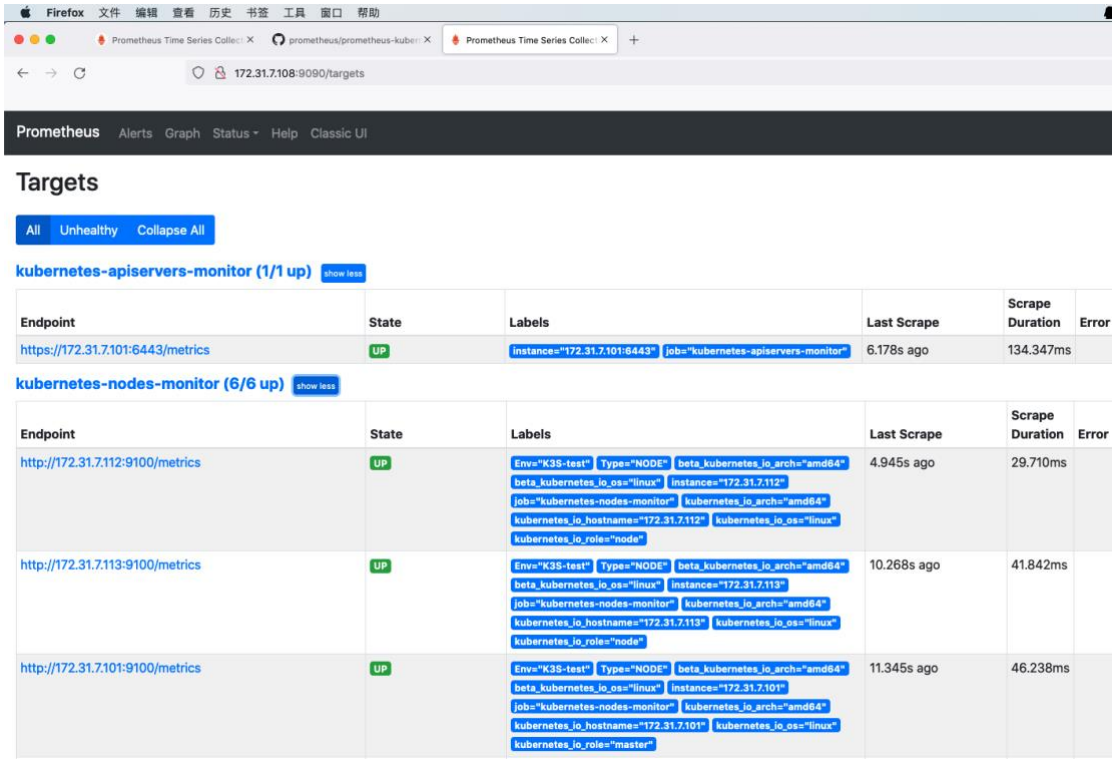
2.1.12.3：prometheus 添加 job：

```
# my global config
global:
  scrape_interval: 15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
    - static_configs:
        - targets:
          # - alertmanager:9093

# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: "prometheus"

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
      - targets: ["localhost:9090"]

  - job_name: 'kubernetes-apiservers-monitor'
    kubernetes_sd_configs:
    - role: endpoints
      api_server: https://172.31.7.101:6443
      tls_config:
        insecure_skip_verify: true
      bearer_token_file: /apps/prometheus/k8s.token
    scheme: https
    tls_config:
      insecure_skip_verify: true
    bearer_token_file: /apps/prometheus/k8s.token
    relabel_configs:
    -       source_labels:       [__meta_kubernetes_namespace,       __meta_kubernetes_service_name,
__meta_kubernetes_endpoint_port_name]
```

```
      action: keep
      regex: default;kubernetes;https
    - target_label: __address__
      replacement: 172.31.7.101:6443

- job_name: 'kubernetes-nodes-monitor'
    scheme: http
    tls_config:
      insecure_skip_verify: true
    bearer_token_file: /apps/prometheus/k8s.token
    kubernetes_sd_configs:
    - role: node
      api_server: https://172.31.7.101:6443
      tls_config:
        insecure_skip_verify: true
      bearer_token_file: /apps/prometheus/k8s.token
    relabel_configs:
      - source_labels: [__address__]
        regex: '(.*):10250'
        replacement: '${1}:9100'
        target_label: __address__
        action: replace
      - source_labels: [__meta_kubernetes_node_label_failure_domain_beta_kubernetes_io_region]
        regex: '(.*)'
        replacement: '${1}'
        action: replace
        target_label: LOC
      - source_labels: [__meta_kubernetes_node_label_failure_domain_beta_kubernetes_io_region]
        regex: '(.*)'
        replacement: 'NODE'
        action: replace
        target_label: Type
      - source_labels: [__meta_kubernetes_node_label_failure_domain_beta_kubernetes_io_region]
        regex: '(.*)'
        replacement: 'K3S-test'
        action: replace
        target_label: Env
      - action: labelmap
        regex: __meta_kubernetes_node_label_(.+)
- job_name: 'kubernetes-pods-monitor'
    kubernetes_sd_configs:
    - role: pod
      api_server: https://172.31.7.101:6443
      tls_config:
```

```
        insecure_skip_verify: true
    bearer_token_file: /apps/prometheus/k8s.token
relabel_configs:
- source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_scrape]
    action: keep
    regex: true
- source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_path]
    action: replace
    target_label: __metrics_path__
    regex: (.+)
- source_labels: [__address__, __meta_kubernetes_pod_annotation_prometheus_io_port]
    action: replace
    regex: ([^:]+)(?::\d+)?;(\d+)
    replacement: $1:$2
    target_label: __address__
- action: labelmap
    regex: __meta_kubernetes_pod_label_(.+)
- source_labels: [__meta_kubernetes_namespace]
    action: replace
    target_label: kubernetes_namespace
- source_labels: [__meta_kubernetes_pod_name]
    action: replace
    target_label: kubernetes_pod_name
- source_labels: [__meta_kubernetes_pod_label_pod_template_hash]
    regex: '(.*)'
    replacement: 'K8S-test'
    action: replace
    target_label: Env
```

2.1.12.4：验证数据：



2.2：static_configs:

后续都是二进制部署的 prometheus 环境

2.2.1:prometheus 配置文件：

```
# A scrape configuration containing exactly one endpoint to scrape: #端点的抓取配置
# Here it's Prometheus itself. #prometheus 的默认配置
scrape_configs:
#作业名称 job=<job_name>会自动添加到此配置的时间序列数据中
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: "prometheus" #job 名称
    # metrics_path defaults to '/metrics' #默认 uri
    # scheme defaults to 'http'. #协议
    static_configs: #静态服务配置
      - targets: ["localhost:9090"] #目标端点地址
  - job_name: 'promethues-node'
    static_configs:
      - targets: ['172.30.7.101:9100','172.30.7.111:9100']
  - job_name: 'prometheus-containers'
    static_configs:
      - targets: ["172.30.7.101:8080","172.30.7.111:8080"]
```

2.3：consul_sd_configs：

https://www.consul.io/

Consul 是分布式 k/v 数据存储集群，目前常用于服务的服务注册和发现。

2.3.1：部署 consul 集群：

https://releases.hashicorp.com/consul/

环境：

```
172.31.2.181
172.31.2.182
172.31.2.183

Node1:
# unzip    consul_1.11.1_linux_amd64.zip
# cp consul /usr/local/bin/
# scp consul 172.31.2.182:/usr/local/bin/
# scp consul 172.31.2.183:/usr/local/bin/

# consul    -h #验证可执行

分别创建数据目录：
# mkdir /data/consul/ -p

启动服务：
node1:
nohup  consul agent -server -bootstrap -bind=172.31.2.181 -client=172.31.2.181 -data-dir=/data/consul -ui -node=172.31.2.181 &

node2:
nohup consul agent -bind=172.31.2.182 -client=172.31.2.182 -data-dir=/data/consul -node=172.31.2.182 -join=172.31.2.181 &

node3:
nohup ./consul agent -bind=172.31.2.183 -client=172.31.2.183 -data-dir=/data/consul -node=172.31.2.183 -join=172.31.2.181 &
```

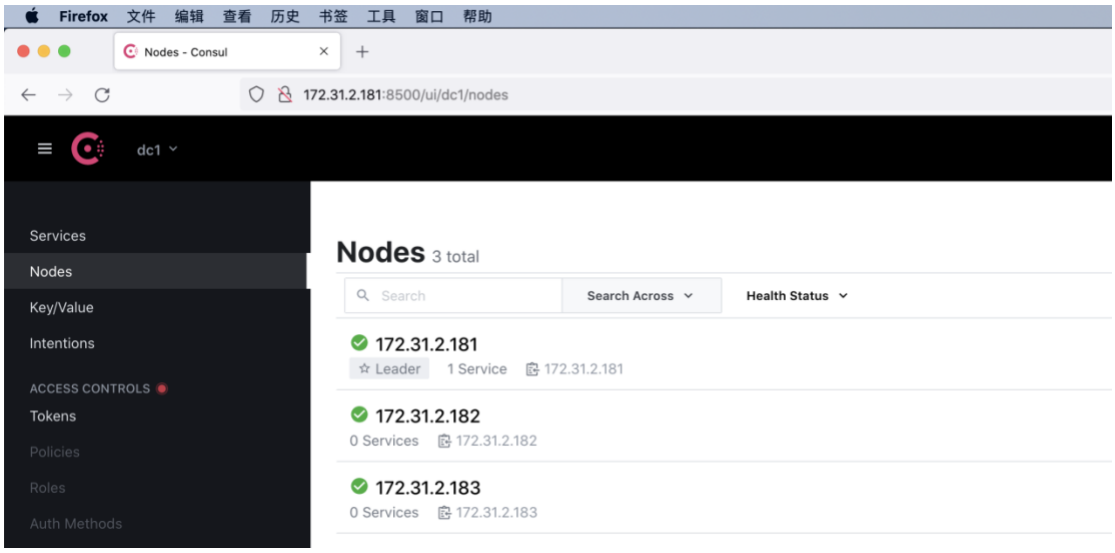
2.3.2：验证集群：

日志：

```
2021-12-27T07:13:07.079Z [INFO]  agent.server: New leader elected: payload=172.31.2.181
2021-12-27T07:13:07.080Z [INFO]  agent.leader: started routine: routine="federation state anti-entropy"
2021-12-27T07:13:07.080Z [INFO]  agent.leader: started routine: routine="federation state pruning"
2021-12-27T07:13:07.080Z [INFO]  agent.server: member joined, marking health alive: member=172.31.2.181 partition=default
2021-12-27T07:13:07.082Z [INFO]  agent.server: federation state anti-entropy synced
2021-12-27T07:13:07.285Z [INFO]  agent: Synced node info
```

```
2021-12-27T07:13:57.363Z [INFO]  agent.server.serf.lan: serf: EventMemberJoin: 172.31.2.183 172.31.2.183
2021-12-27T07:13:57.363Z [INFO]  agent.server: member joined, marking health alive: member=172.31.2.183 partition=default
2021-12-27T07:14:04.060Z [INFO]  agent.server.serf.lan: serf: EventMemberJoin: 172.31.2.182 172.31.2.182
2021-12-27T07:14:04.060Z [INFO]  agent.server: member joined, marking health alive: member=172.31.2.182 partition=default
```

web 截图：

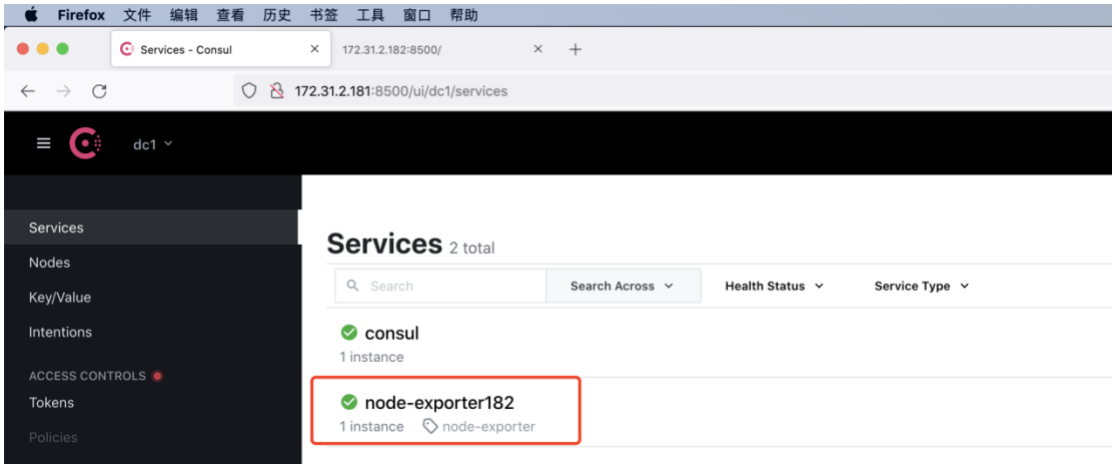


2.3.3：测试写入数据：

通过 consul 的 API 写入数据

```
~# curl -X PUT -d '{"id": "node-exporter181","name": "node-exporter181","address": "172.31.2.181","port":9100,"tags": ["node-exporter"],"checks": [{"http": "http://172.31.2.181:9100/","interval": "5s"}]}' http://172.31.2.181:8500/v1/agent/service/register

~# curl -X PUT -d '{"id": "node-exporter182","name": "node-exporter182","address": "172.31.2.182","port":9100,"tags": ["node-exporter"],"checks": [{"http": "http://172.31.2.182:9100/","interval": "5s"}]}' http://172.31.2.181:8500/v1/agent/service/register
```

2.3.4：consul 验证数据：



2.3.5：配置 prometheus 到 consul 发现服务：

2.3.5.1：主要配置字段：

```
static_configs:        #配置数据源
consul_sd_configs: #指定基于 consul 服务发现的配置
rebel_configs：#重新标记
services: []       #表示匹配 consul 中所有的 service
```

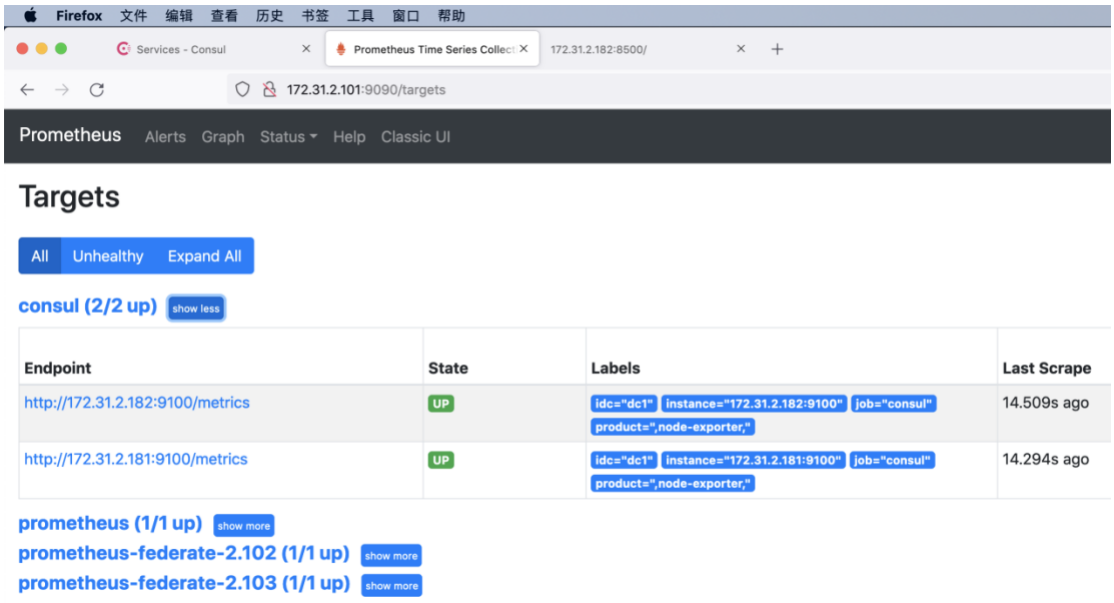
2.3.5.2：k8s 部署 prometheus 配置文件：

```
yaml]# vim prometheus-cfg.yam
yaml]# kubectl apply -f prometheus-cfg.yaml && kubectl delete -f prometheus-deploy.yaml && kubectl apply -f prometheus-deploy.yaml
    - job_name: 'nginx-monitor-serverA'
      static_configs:
      - targets: ['172.30.7.201:9913']
    - job_name: consul
      honor_labels: true
      metrics_path: /metrics
      scheme: http
      consul_sd_configs:
        - server: 172.31.2.181:8500
          services: []
        - server: 172.31.2.182:8500
          services: []
        - server: 172.31.2.183:8500
          services: []
      relabel_configs:
      - source_labels: ['__meta_consul_tags']
        target_label: 'product'
      - source_labels: ['__meta_consul_dc']
        target_label: 'idc'
      - source_labels: ['__meta_consul_service']
        regex: "consul"
```

```
        action: drop
```

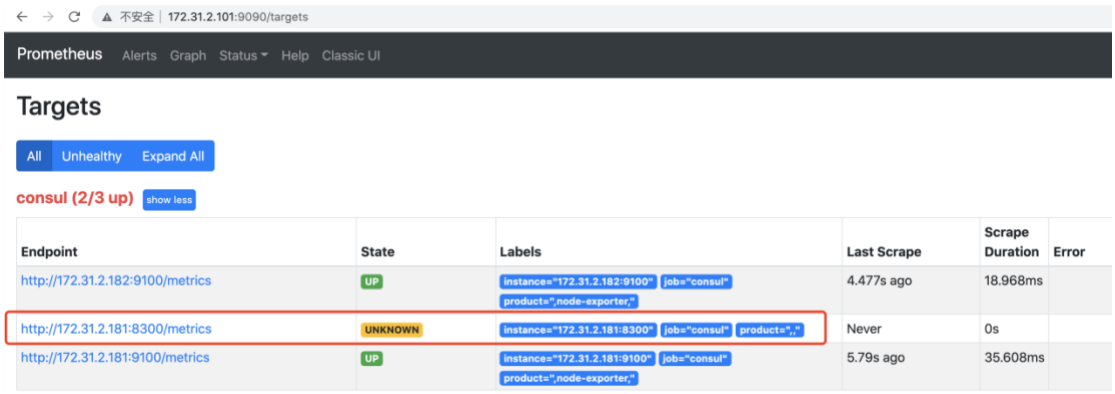
2.3.5.3：二进制-prometheus 配置文件：

```
# cat /apps/prometheus/prometheus.yml
  - job_name: consul
    honor_labels: true
    metrics_path: /metrics
    scheme: http
    consul_sd_configs:
      - server: 172.31.2.181:8500
        services: []    #发现的目标服务名称，空为所有服务，可以写 servicea,servcieb,servicec
      - server: 172.31.2.182:8500
        services: []
      - server: 172.31.2.183:8500
        services: []
    relabel_configs:
    - source_labels: ['__meta_consul_tags']
      target_label: 'product'
    - source_labels: ['__meta_consul_dc']
      target_label: 'idc'
    - source_labels: ['__meta_consul_service']
      regex: "consul"
      action: drop
# systemctl   restart   prometheus.service
```



匹配删除__meta_consul_service 为 consul 的发现结果：

```
    - source_labels: ['__meta_consul_service']
      regex: "consul"
      action: drop
```



2.3.6：consul 服务注册与删除：

```
# curl -X PUT -d '{"id": "node-exporter103","name": "node-exporter103","address": "172.31.2.103","port":9100,"tags": ["node-exporter103"],"checks": [{"http": "http://172.31.2.103:9100/","interval": "5s"}]}' http://172.31.2.181:8500/v1/agent/service/register


# curl --request PUT http://172.31.2.181:8500/v1/agent/service/deregister/node-exporter103
```

2.4：file_sd_configs：

2.4.1：编辑 sd_configs 文件：

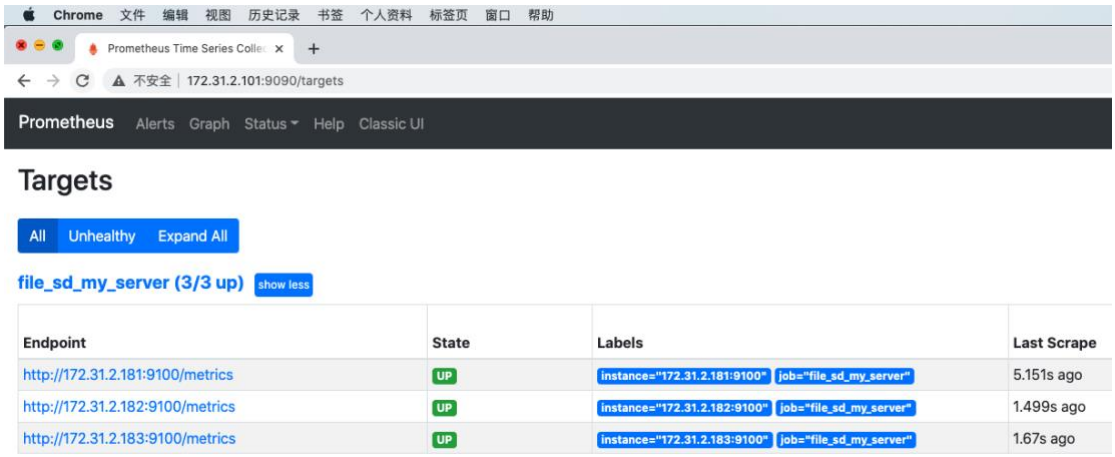
```
# pwd
```

```
/apps/prometheus
#mkdir    file_sd
#vim file_sd/sd_my_server.json
[
  {
    "targets":   ["172.31.2.181:9100","172.31.2.182:9100","172.31.2.183:9100"]
  }
]
```

2.4.2：prometheus 调用 sd_configs：

```
# vim prometheus.yml
  - job_name: 'file_sd_my_server'
    file_sd_configs:
      - files:
        - /apps/prometheus/file_sd/sd_my_server.json
        refresh_interval: 10s
# systemctl    restart prometheus
```

2.4.3：验证数据：



2.5：DNS 服务发现：

基于 DNS 的服务发现允许配置指定一组 DNS 域名，这些域名会定期查询以发现目标列表,域名需要可以被配置的 DNS 服务器解析为 IP。

此服务发现方法仅支持基本的 DNS A、AAAA 和 SRV 记录查询。
A 记录： 域名解析为 IP
SRV：SRV 记录了哪台计算机提供了具体哪个服务，格式为：自定义的服务的名字.协议的类型.域名（例如：_example-server._tcp.www.mydns.com）

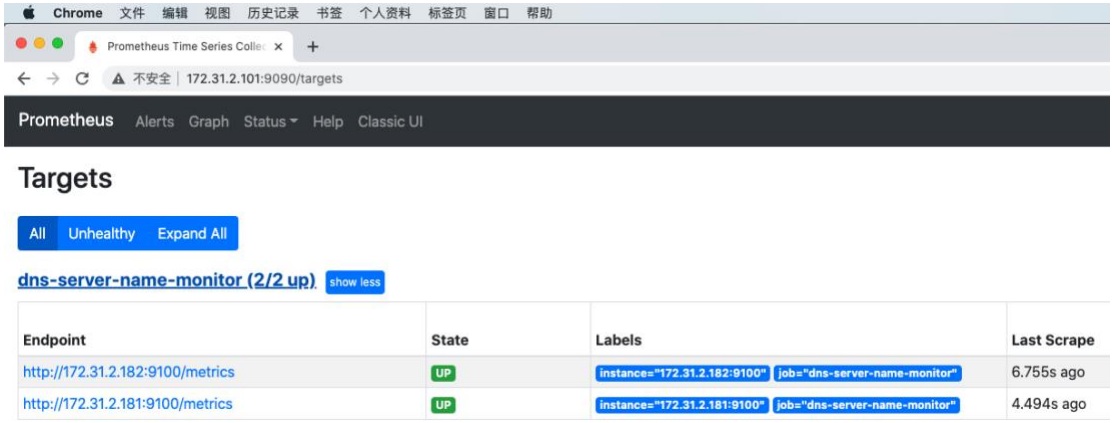
prometheus 会对收集的指标数据进行重新打标，重新标记期间，可以使用以下元标签：
__meta_dns_name：产生发现目标的记录名称。
__meta_dns_srv_record_target: SRV 记录的目标字段

__meta_dns_srv_record_port: SRV 记录的端口字段

2.5.1：A 记录服务发现：

```
# vim /etc/hosts
172.31.2.181 node1.example.com
172.31.2.182 node2.example.com

# vim /apps/prometheus/prometheus.yml
  - job_name: 'dns-server-name-monitor'
    metrics_path: "/metrics"
    dns_sd_configs:
    - names: ["node1.example.com", "node2.example.com"]
      type: A
      port: 9100
# systemctl   restart prometheus.service
```

2.5.2：验证服务状态：



5.2.2：SRV 服务发现：

需要有 DNS 服务器实现域名解析

```
# vim /apps/prometheus/prometheus.yml
  - job_name: 'dns-node-monitor-srv'
    metrics_path: "/metrics"
    dns_sd_configs:
    - names: ["_prometheus._tcp.node.example.com"]
      type: SRV
      port: 9100

# systemctl   restart   prometheus.service
```

三： kube-state-metrics 组件介绍：

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

Kube-state-metrics:通过监听 API Server 生成有关资源对象的状态指标，比如 Deployment、Node、Pod，需要注意的是 kube-state-metrics 只是简单的提供一个 metrics 数据，并不会存储这些指标数据，所以我们可以使用 Prometheus 来抓取这些数据然后存储，主要关注的是业务相关的一些元数据，比如 Deployment、Pod、副本状态等；调度了多少个 replicas？现在可用的有几个？；多少个 Pod 是 running/stopped/terminated 状态？；Pod 重启了多少次？；我有多少 job 在运行中。

kube-state-metrics is a simple service that listens to the Kubernetes API server and generates metrics about the state of the objects.

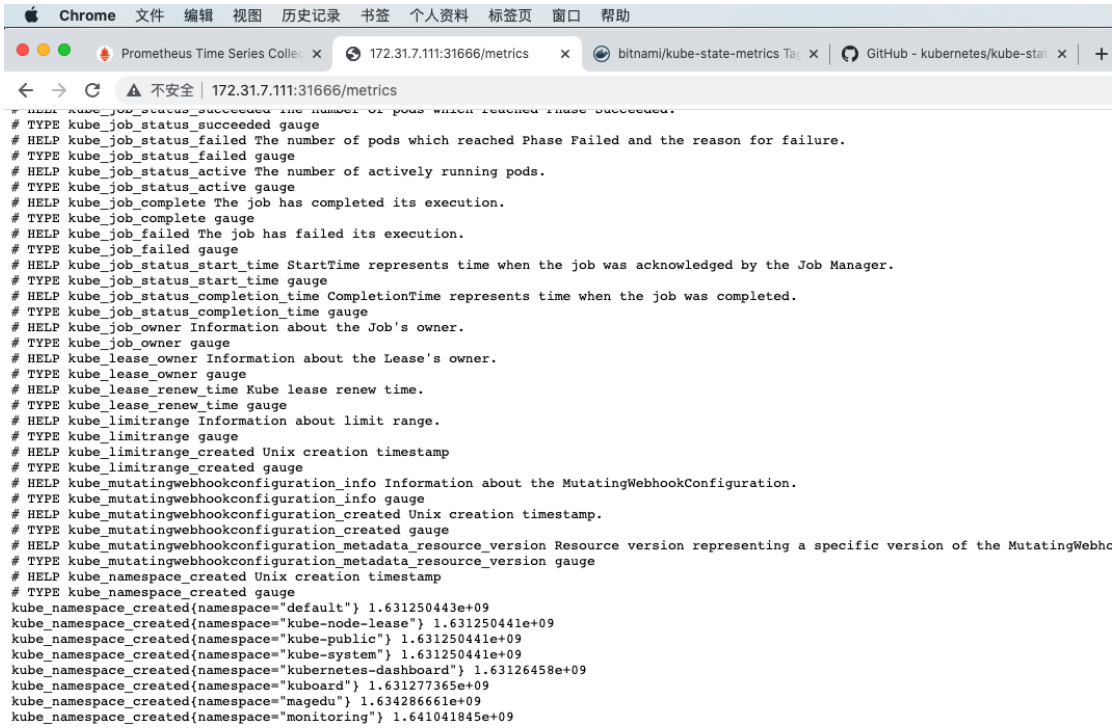
Github：https://github.com/kubernetes/kube-state-metrics
https://quay.io/repository/coreos/kube-state-metrics?tag=latest&tab=tags

https://xie.infoq.cn/article/9e1fff6306649e65480a96bb1 #指标

3.1：部署 kube-state-metrics：

```
# kubectl apply    -f case5-kube-state-metrics-deploy.yaml
```

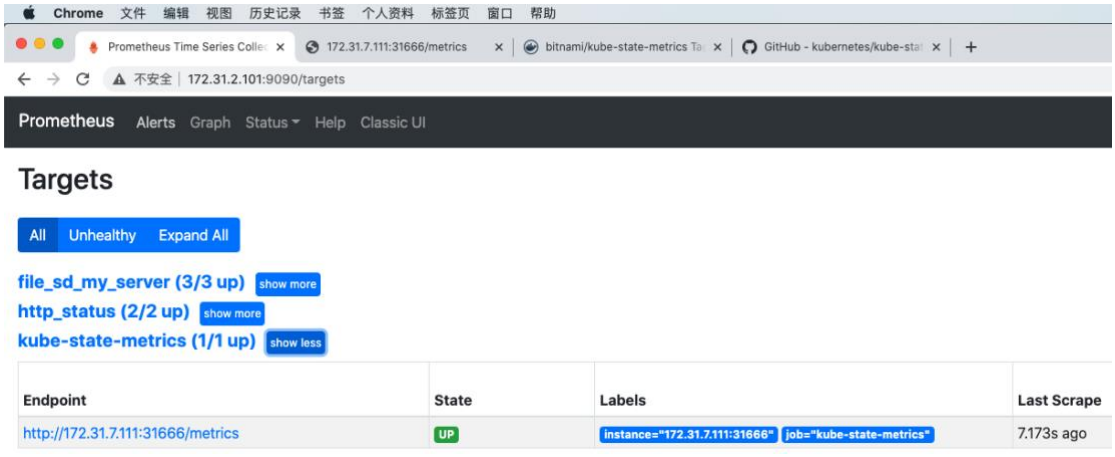
3.2：验证数据：



3.3：prometheus 采集数据：

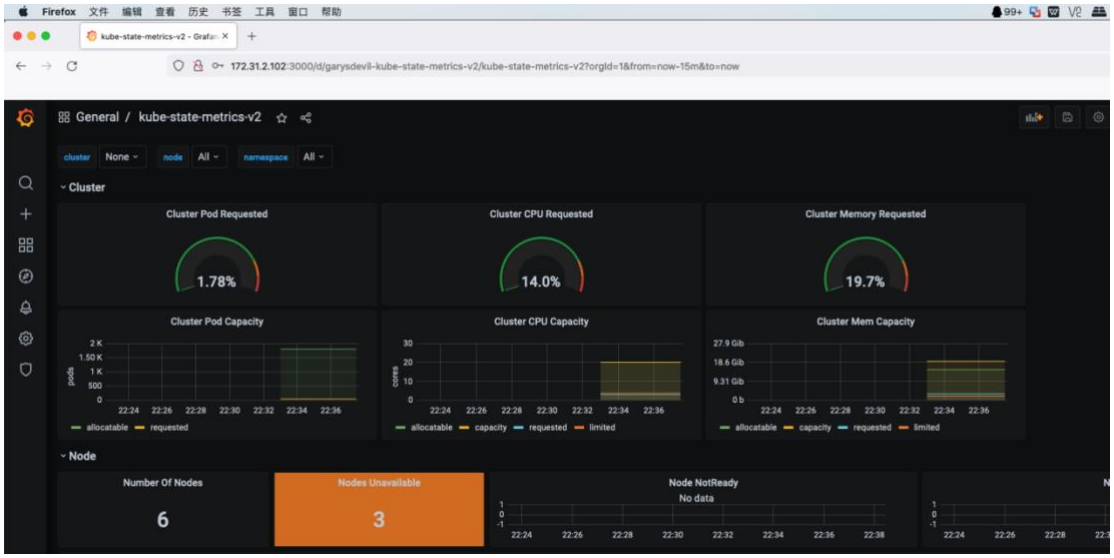
```
# vim /apps/prometheus/prometheus.yml
  - job_name: "kube-state-metrics"
    static_configs:
      - targets: ["172.31.7.111:31666"]
```

3.4：验证 prometheus 状态：

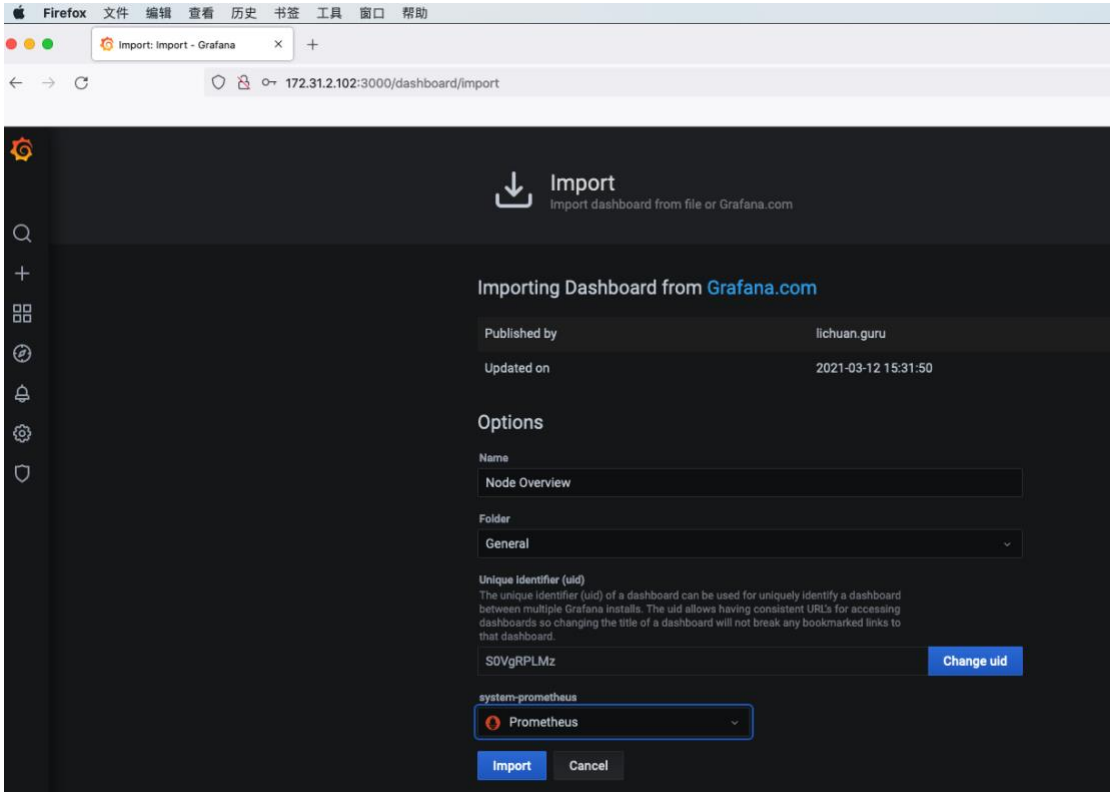


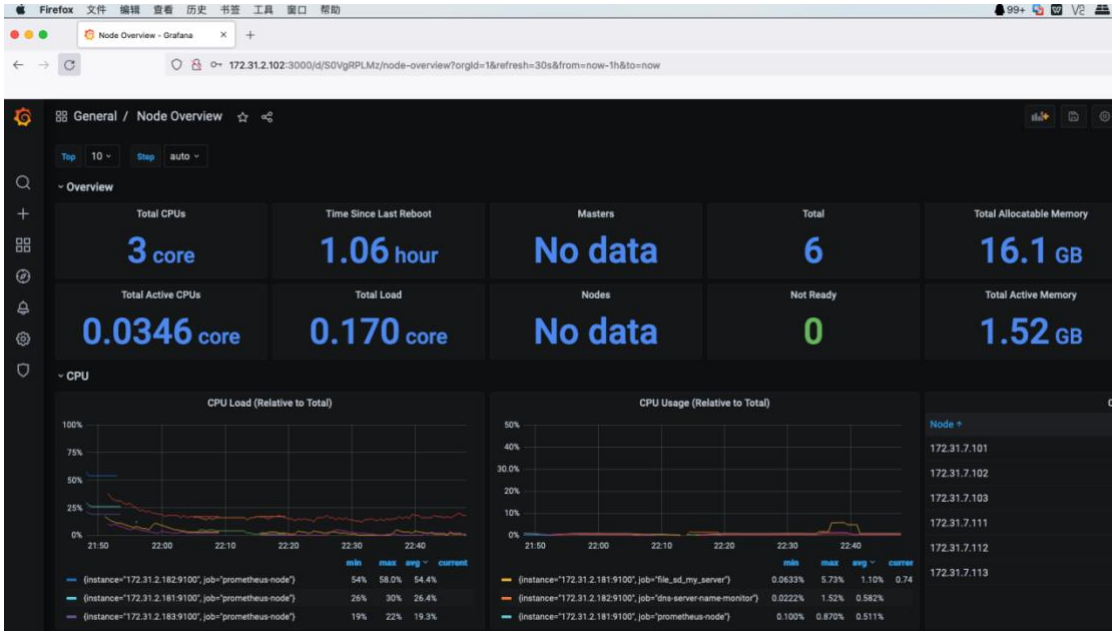
3.5：grafana 导入模板：

13332:


Firefox 文件 编辑 查看 历史 书签 工具 窗口 帮助
172.31.2.102:3000/d/garysdevil-kube-state-metrics-v2/kube-state-metrics-v2?orgId=1&from=now-15m&to=now
General / kube-state-metrics-v2
cluster None node All namespace All
Cluster
Cluster Pod Requested
1.78%
Cluster CPU Requested
14.0%
Cluster Memory Requested
19.7%
Cluster Pod Capacity
Cluster CPU Capacity
Cluster Mem Capacity
allocatable requested
allocatable capacity requested limited
Node
Number Of Nodes
6
Nodes Unavailable
3
Node NotReady
No data

13824:


Firefox 文件 编辑 查看 历史 书签 工具 窗口 帮助
Import: Import - Grafana
172.31.2.102:3000/dashboard/import
Import
Import dashboard from file or Grafana.com
Importing Dashboard from Grafana.com
Published by
lichuan.guru
Updated on
2021-03-12 15:31:50
Options
Name
Node Overview
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.
S0VgRPLMz
Change uid
system-prometheus
Prometheus
Import
Cancel



四：监控扩展：

基于第三方 exporter 实现对目的服务的监控

4.1：监控 tomcat：

监控 tomcat 的活跃连接数、堆栈内存等信息：

```
# TYPE tomcat_connections_active_total gauge
tomcat_connections_active_total{name="http-nio-8080",} 2.0

# TYPE jvm_memory_bytes_used gauge
jvm_memory_bytes_used{area="heap",} 2.4451216E7
```

4.1.1：自定义镜像：

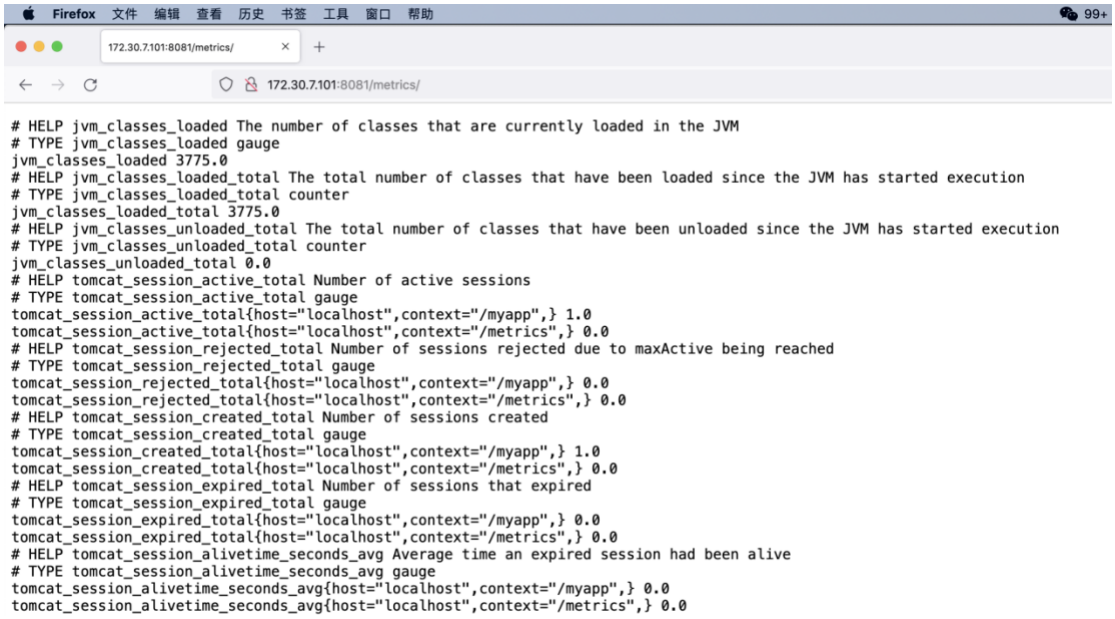
```
# cat Dockerfile
FROM tomcat:8.5.73-jdk11-corretto

LABEL maintainer="jack 2973707860@qq.com"

ADD server.xml /usr/local/tomcat/conf/server.xml

RUN mkdir /data/tomcat/webapps -p
ADD myapp /data/tomcat/webapps/myapp
ADD metrics.war /data/tomcat/webapps
ADD simpleclient-0.8.0.jar    /usr/local/tomcat/lib/
ADD simpleclient_common-0.8.0.jar /usr/local/tomcat/lib/
ADD simpleclient_hotspot-0.8.0.jar /usr/local/tomcat/lib/
ADD simpleclient_servlet-0.8.0.jar /usr/local/tomcat/lib/
```

```
ADD tomcat_exporter_client-0.0.12.jar /usr/local/tomcat/lib/
#ADD run_tomcat.sh /apps/tomcat/bin/
EXPOSE 8080 8443 8009
#CMD ["/apps/tomcat/bin/catalina.sh","run"]
#CMD ["/apps/tomcat/bin/run_tomcat.sh"]
```

4.1.2：构建并测试镜像

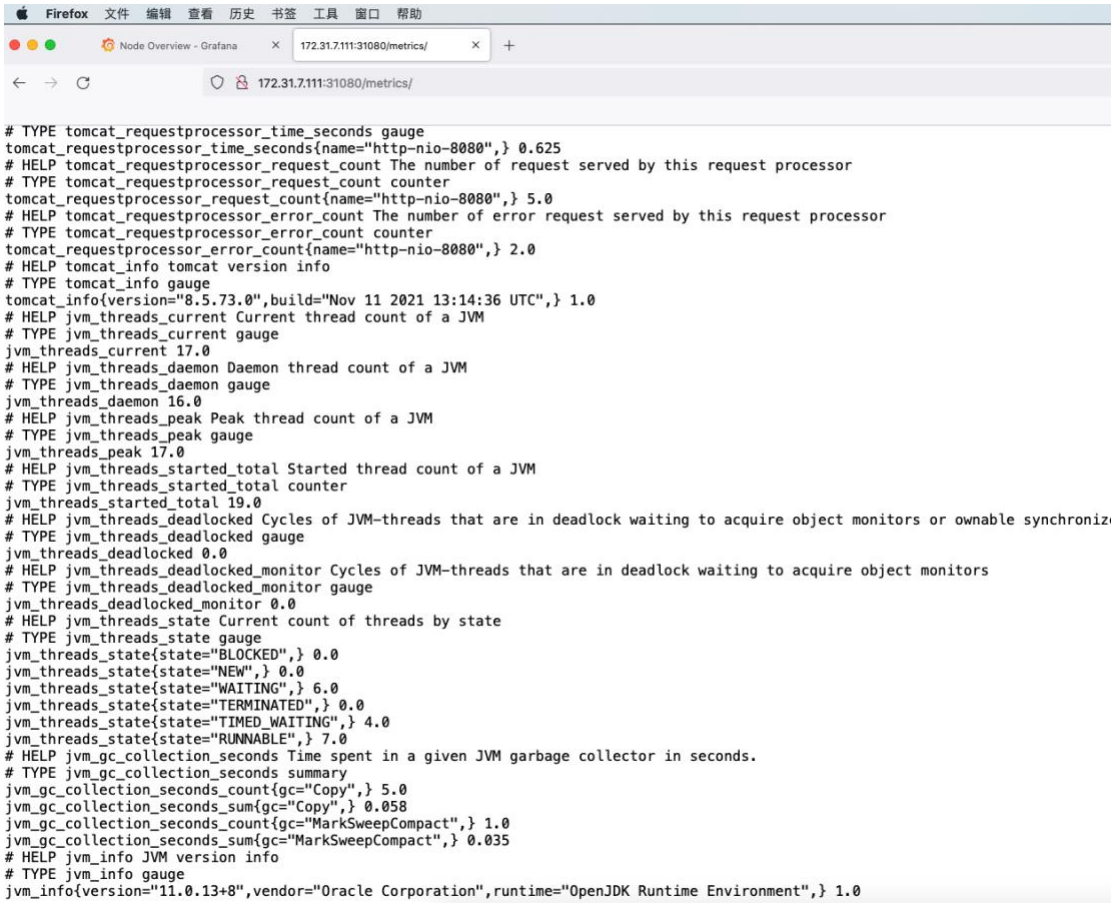
```
# bash build-command.sh
# docker run -it    --rm -p 8081:8080 harbor.studylinux.net/myapp/tomcat-app1:v1
```

Metrics 界面：



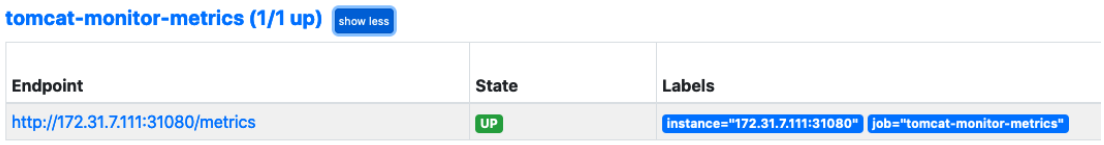
4.1.3：创建 pod 并验证：

```
l# pwd
/root/prometheus-files/case/app-monitor-case/tomcat/yaml


# kubectl apply    -f tomcat-deploy.yaml    -f tomcat-svc.yaml
```



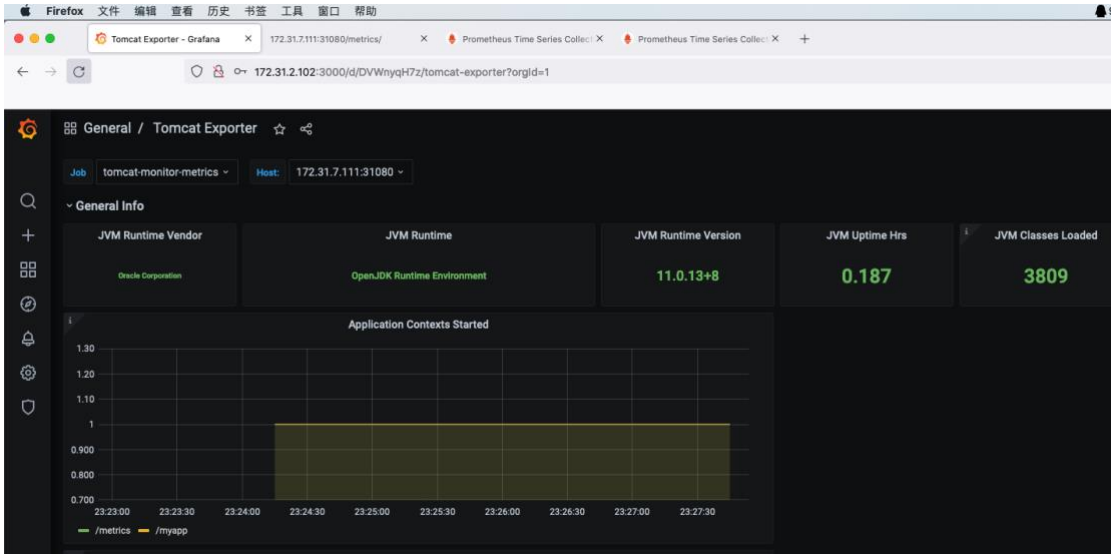
4.1.4：prometheus 采集并验证数据：

```
  - job_name: "tomcat-monitor-metrics"
    static_configs:
      - targets: ["172.31.7.111:31080"]
# systemctl    restart prometheus.service
```



4.1.5：grafana 导入模板：

https://github.com/nlighten/tomcat_exporter
https://github.com/nlighten/tomcat_exporter/tree/master/dashboard



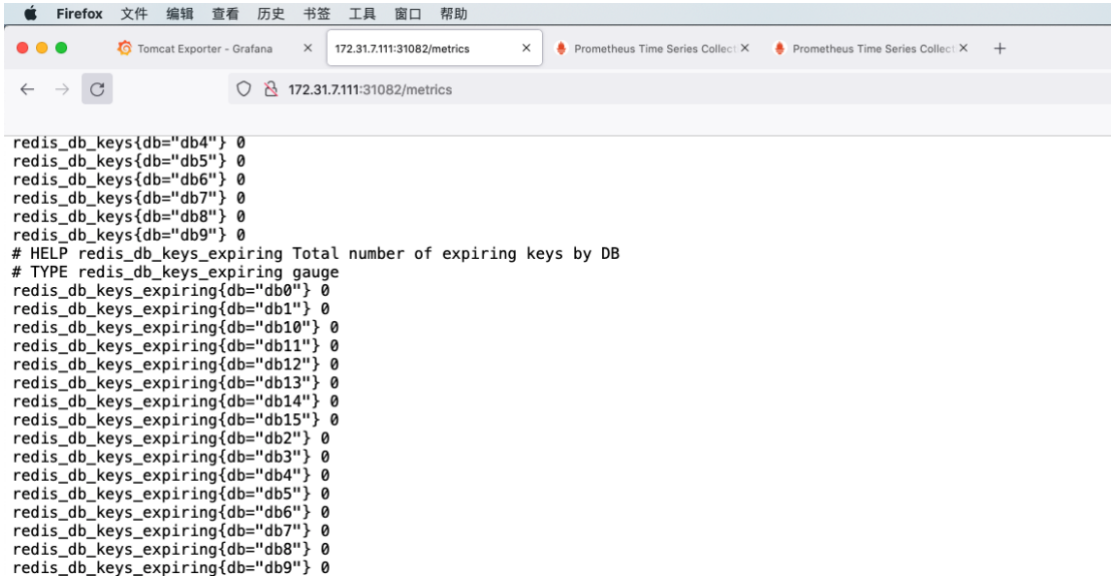
4.2：监控 Redis：

https://github.com/oliver006/redis_exporter

4.2.1：部署 Redis：

```
# pwd
/root/prometheus-files/case/app-monitor-case/redis/yaml

# kubectl apply    -f .
deployment.apps/redis created
service/redis-exporter-service created
service/redis-redis-service created
```

4.2.2：验证 metrics：



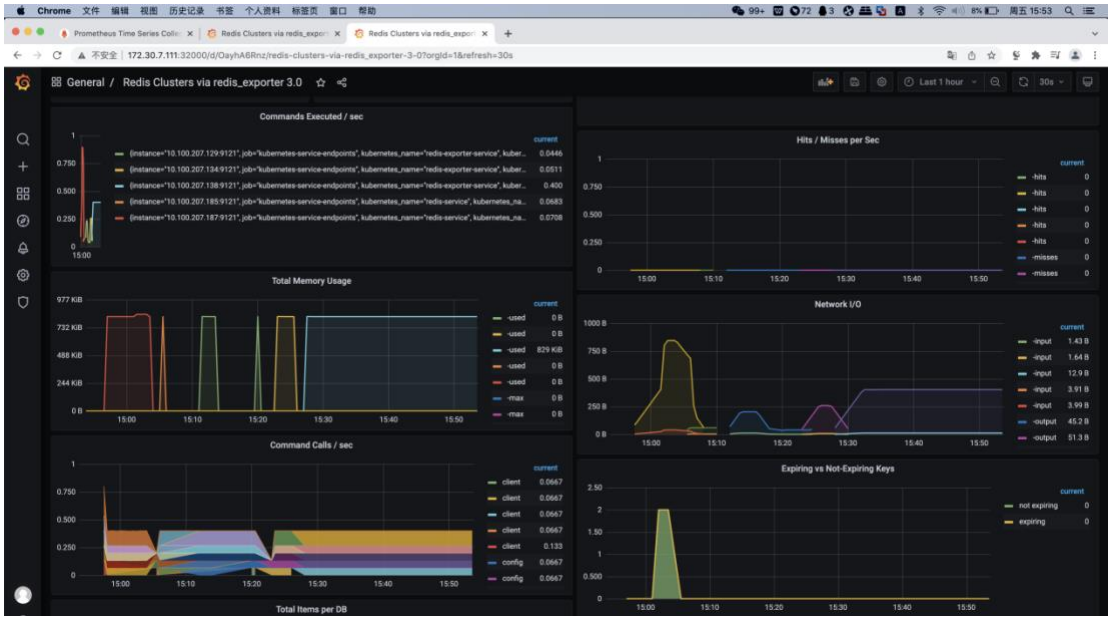
4.2.3：prometheus 采集数据：

```
  - job_name: "redis-monitor-metrics"
```

```
      static_configs:
        - targets: ["172.31.7.111:31082"]
# systemctl    restart prometheus.service
```

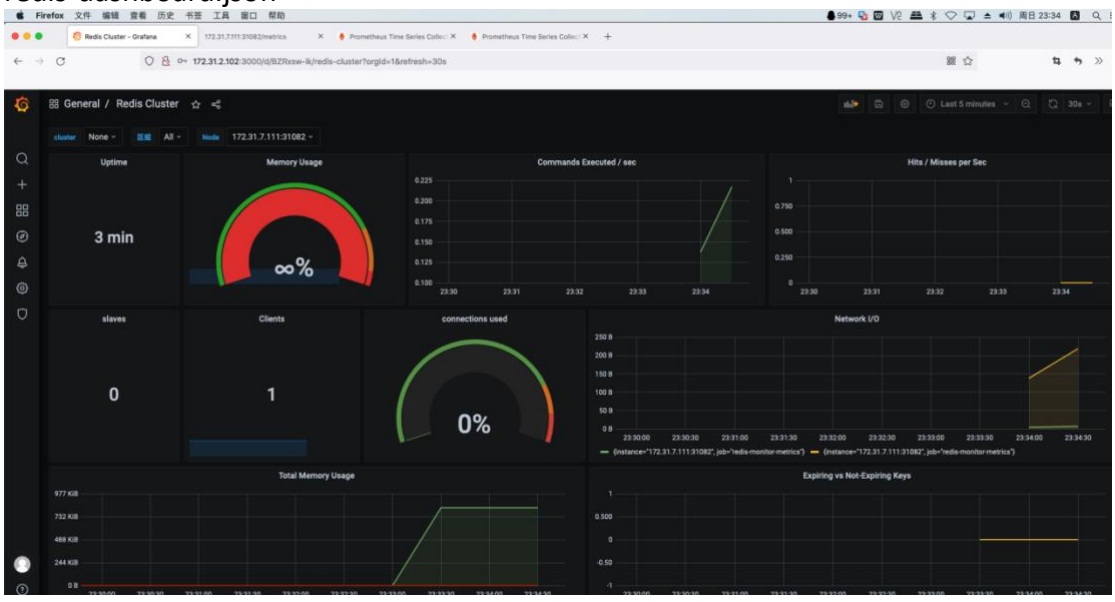
4.2.4：grafana 导入模板并验证：

14615



导入自定义模板：

redis-dashboard.json



4.3：监控 MySQL：

mysqld_exporter 监控 MySQL 服务的运行状态

https://github.com/prometheus/mysqld_exporter

4.3.1：安装 mysql：

```
root@prometheus-node2:~# apt install mariadb-server
root@prometheus-node2:~# mysql
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 50
Server version: 10.3.32-MariaDB-0ubuntu0.20.04.1 Ubuntu 20.04

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

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

MariaDB [(none)]>
```

4.3.2：授权监控账户权限：

```
mysql> CREATE USER 'mysql_exporter'@'localhost' IDENTIFIED BY 'imnot007*';
Query OK, 0 rows affected (0.00 sec)

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

#验证权限：
# mysql -umysql_exporter -pimnot007* -hlocalhost
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 2
Server version: 5.6.43 MySQL Community Server (GPL)

Copyright (c) 2000, 2019, 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>
```

4.3.3：准备 mysqld_exporter 环境：

```
# tar xf mysqld_exporter-0.13.0.linux-amd64.tar.gz
# cd mysqld_exporter-0.13.0.linux-amd64/
# cp mysqld_exporter    /usr/local/bin/
# cp mysqld_exporter    /usr/local/bin/

#免密码登录配置：
```

```
# cat /root/.my.cnf
[client]
user=mysql_exporter
password=imnot007*
```

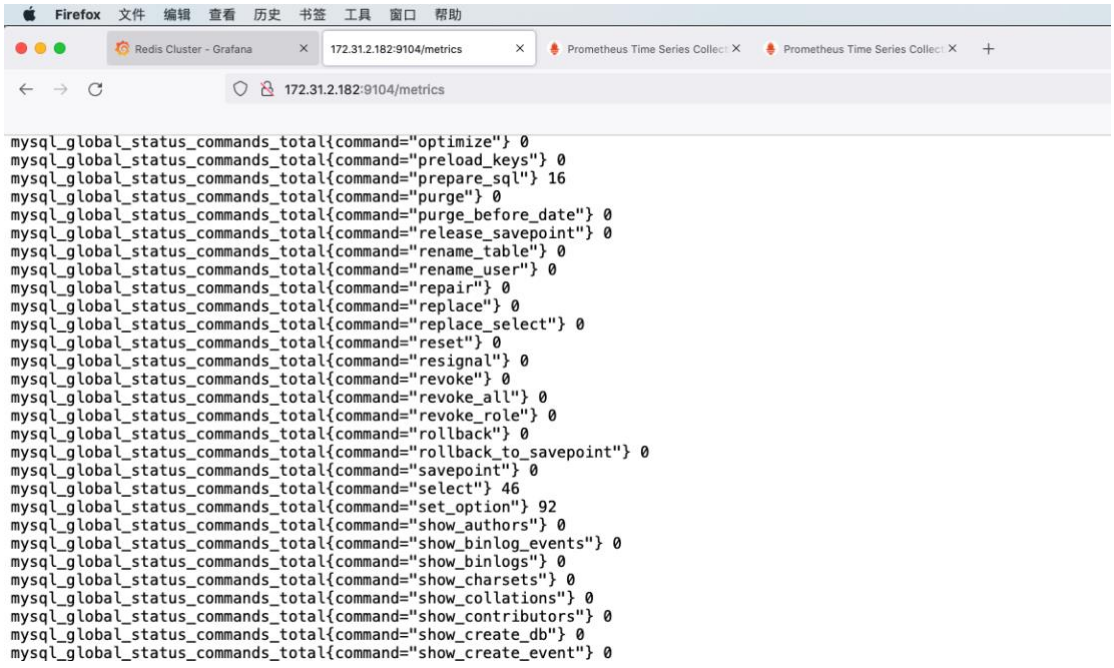
验证权限：

```
# mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.6.43 MySQL Community Server (GPL)

Copyright (c) 2000, 2019, 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>
```

4.3.4：启动 mysql_exporter：

```
# vim    /etc/systemd/system/mysqld_exporter.service
[Unit]
Description=Prometheus Node Exporter
After=network.target

[Service]
ExecStart=/usr/local/bin/mysqld_exporter --config.my-cnf=/root/.my.cnf

[Install]
WantedBy=multi-user.target

# systemctl  daemon-reload
# systemctl  restart  mysqld_exporter
# systemctl  enable mysqld_exporter
```

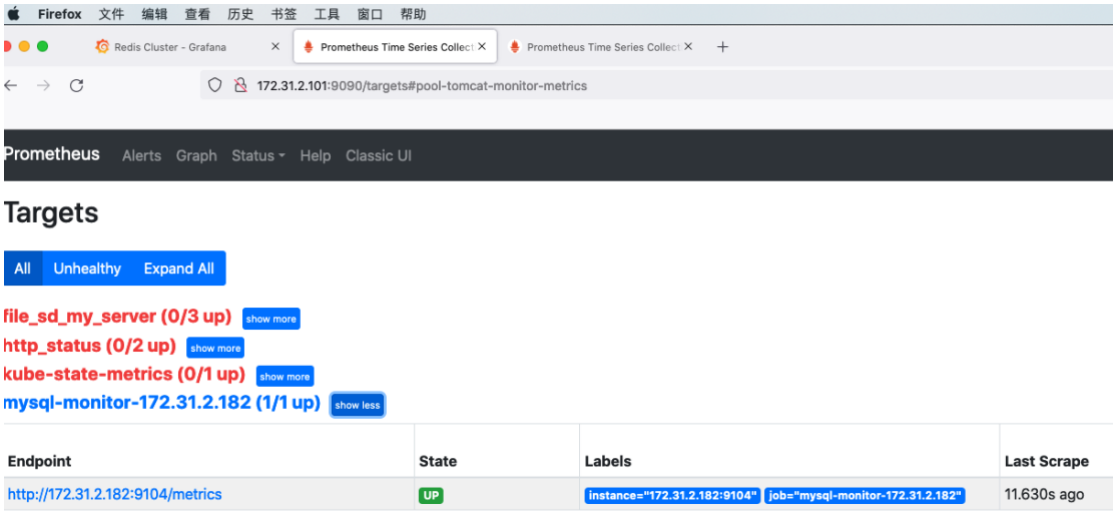
4.3.5：验证 metrics：



4.3.6：prometheus 采集数据：

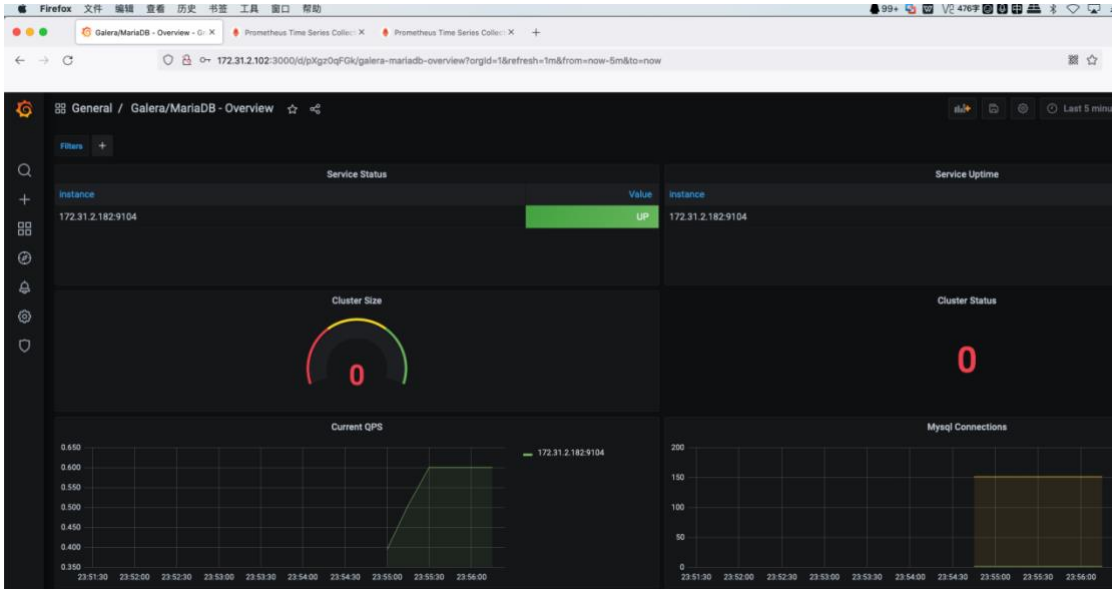
```
# vim prometheus.yml
  - job_name: mysql-monitor-172.31.2.182
    static_configs:
    - targets: ['172.31.2.182:9104']
# systemctl    restart prometheus.service
```

4.3.7：验证指标数据：

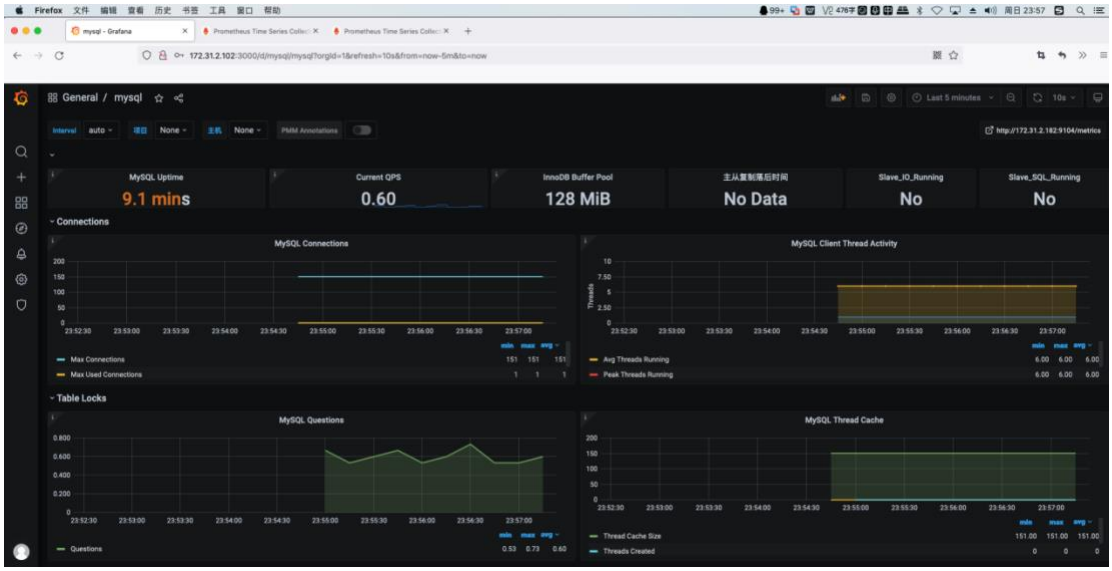


4.3.8：导入模板：

13106



11323



4.4：监控 haproxy:

通过 haproxy_exporter 监控 haproxy

https://github.com/prometheus/haproxy_exporter

4. 4. 1：部署 haproxy：

```
# yum    install    haproxy

修改 socket 文件
        # turn on stats unix socket
        #stats socket /var/lib/haproxy/stats
        stats socket /var/lib/haproxy/haproxy.sock mode 600 level admin

# systemctl    restart    haproxy
```

4.4.2：部署 haproxy_exporter：

```
# tar    xvf haproxy_exporter-0.12.0.linux-amd64.tar.gz
# cp haproxy_exporter-0.12.0.linux-amd64/haproxy_exporter    /usr/local/bin/
```

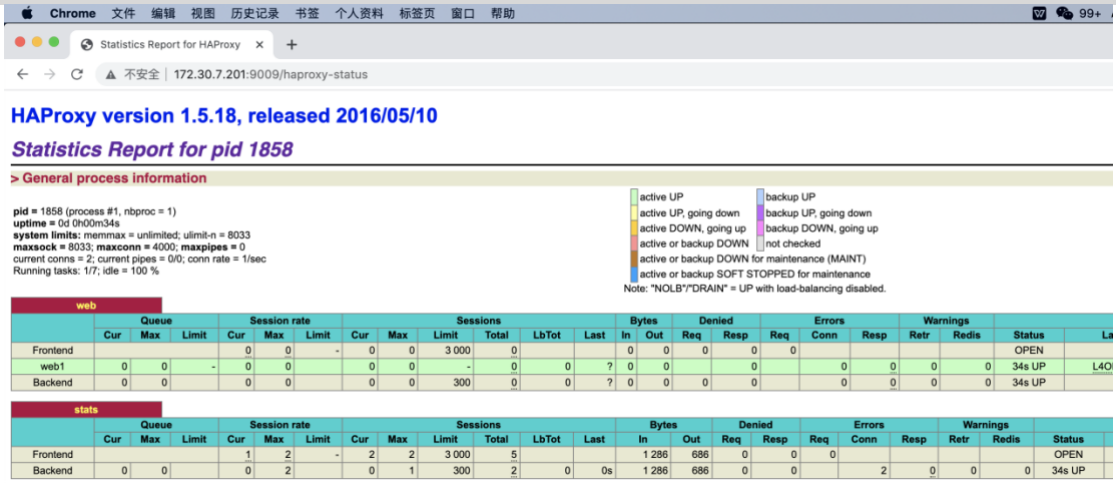
启动方式一：

```
# haproxy_exporter --haproxy.scrape-uri=unix:/var/lib/haproxy/haproxy.sock
```

启动方式二：

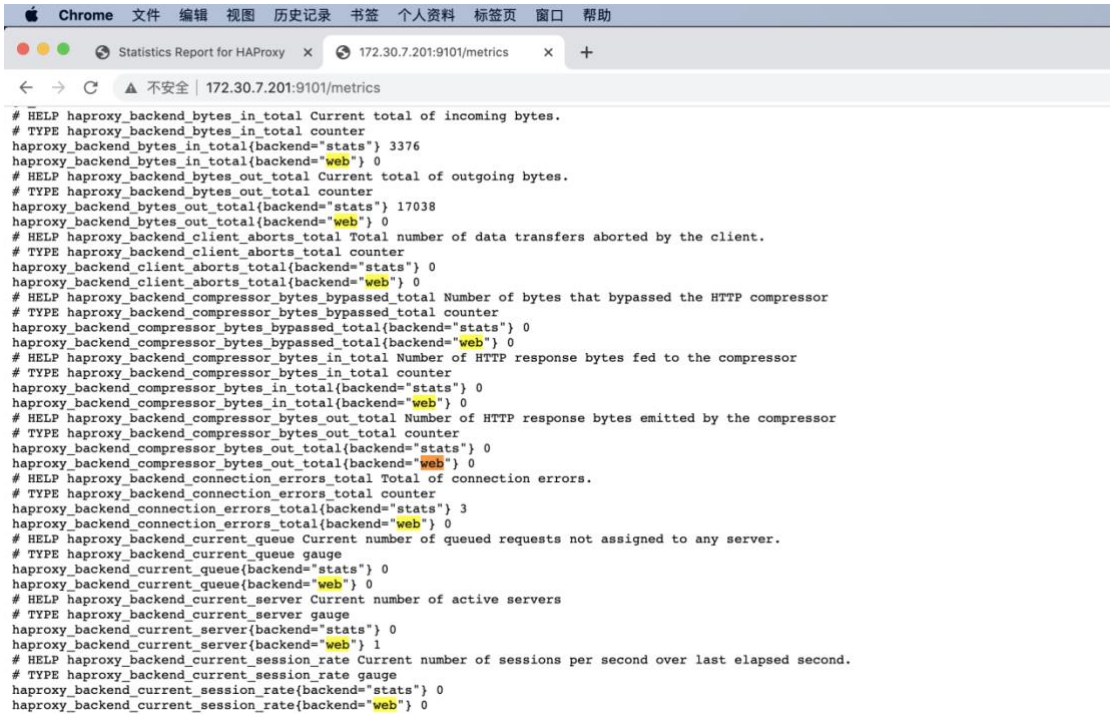
开启状态页：

```
listen stats
   bind :9009
   stats enable
   #stats hide-version
   stats uri    /haproxy-status
   stats realm HAPorxy\ Stats\ Page
   stats auth haadmin:123456
   stats auth admin:123456
```




```
# haproxy_exporter --haproxy.scrape-uri="http://haadmin:q1w2e3r4ys@127.0.0.1:9009/haproxy-status;csv" &
```

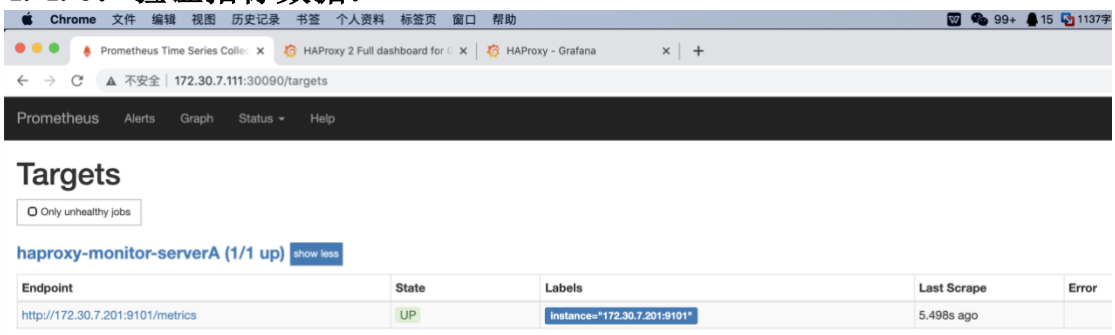
4.4.3：验证 metrics 数据：



4.4.4：prometheus 添加 job：

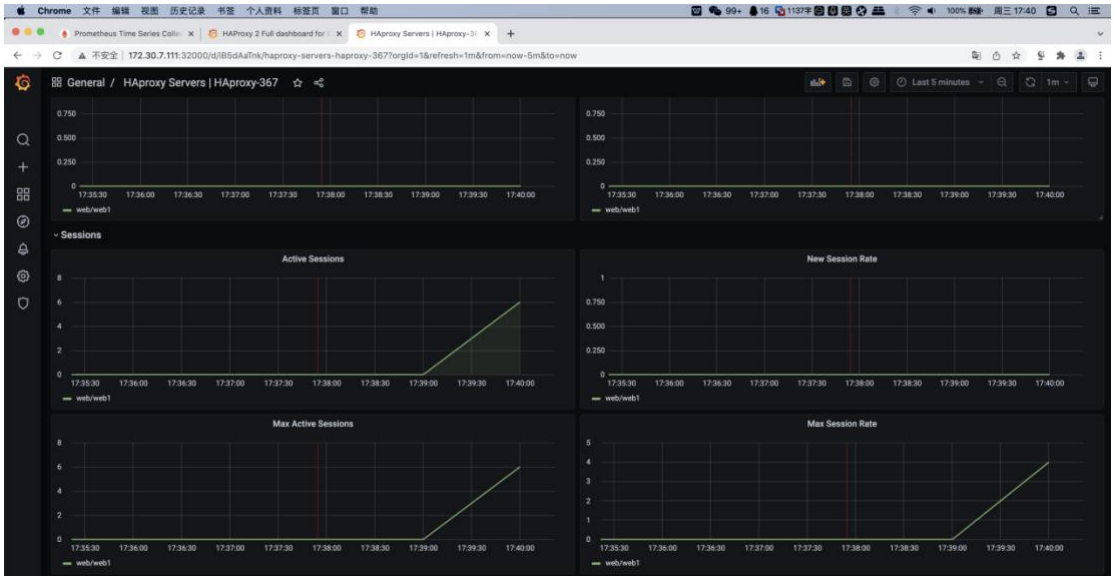
```
    - job_name: 'mysql-monitor-serverA'
      static_configs:
      - targets: ['172.30.7.101:9104']
    - job_name: 'haproxy-monitor-serverA'
      static_configs:
      - targets: ['172.30.7.101:9101']


# kubectl apply  -f  prometheus-cfg.yaml
# kubectl  delete  -f prometheus-deploy.yaml  && kubectl  apply  -f prometheus-deploy.yaml
```

4.4.5：验证指标数据：



4.4.6：grafana 导入模板并验证：

367 2428



4.5：监控 nginx：

通过 prometheus 监控 nginx

需要在编译安装 nginx 的时候添加 nginx-module-vts 模块，
github 地址：https://github.com/vozlt/nginx-module-vts

4.5.1：编译安装 nginx：

```
# git clone https://github.com/vozlt/nginx-module-vts.git
# wget http://nginx.org/download/nginx-1.20.2.tar.gz
# tar xvf nginx-1.20.2.tar.gz
# cd nginx-1.20.2/
./configure --prefix=/apps/nginx \
--with-http_ssl_module \
--with-http_v2_module \
--with-http_realip_module \
--with-http_stub_status_module    \
--with-http_gzip_static_module \
--with-pcre \
--with-file-aio \
--with-stream \
--with-stream_ssl_module \
--with-stream_realip_module \
--add-module=/usr/local//src/nginx-module-vts/

# make
# make install
```

4.5.2：编辑 nginx 配置文件：

```
# vim /apps/nginx/conf/nginx.conf
```

http 范围配置：

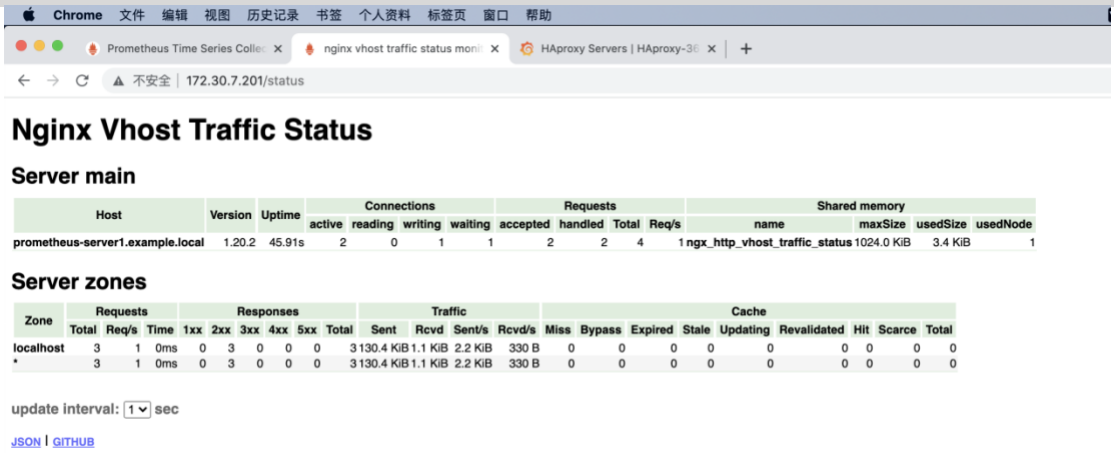
```
#gzip  on;
vhost_traffic_status_zone;
```

server 字段配置：

```
location / {
    root   html;
    index  index.html index.htm;
}

location /status {
   vhost_traffic_status_display;
   vhost_traffic_status_display_format html;
}
```

4.5.3：启动 nginx 并验证 web 状态页：

```
# /apps/nginx/sbin/nginx  -t
nginx: the configuration file /apps/nginx/conf/nginx.conf syntax is ok
nginx: configuration file /apps/nginx/conf/nginx.conf test is successful
# /apps/nginx/sbin/nginx
```



五：alertmanager：

5.1：邮件：

略

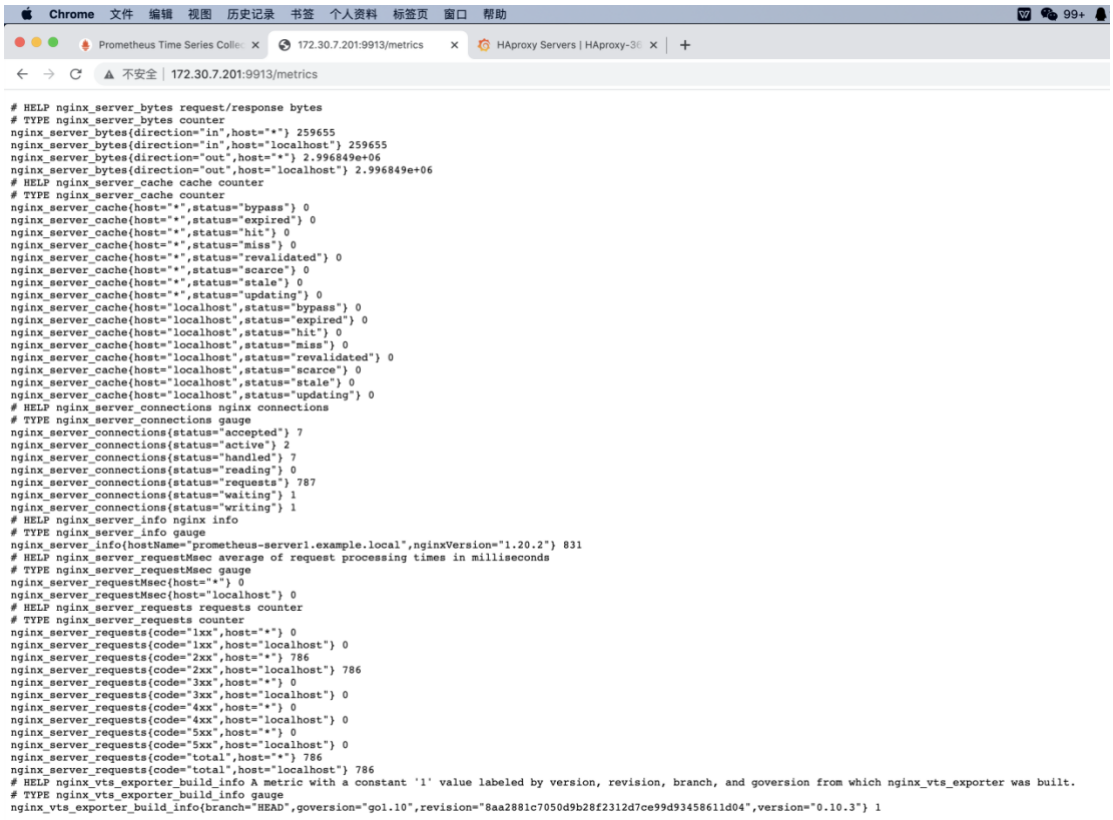
5.2：钉钉：

5.3：企业微信：

4.5.4：部署 nginx_exporter：

```
# tar xvf nginx-vts-exporter-0.10.3.linux-amd64.tar.gz
# cp nginx-vts-exporter-0.10.3.linux-amd64/nginx-vts-exporter    /usr/local/bin/
# nginx-vts-exporter    -nginx.scrape_uri http://127.0.0.1/status/format/json
```

4.5.5：验证指标数据：



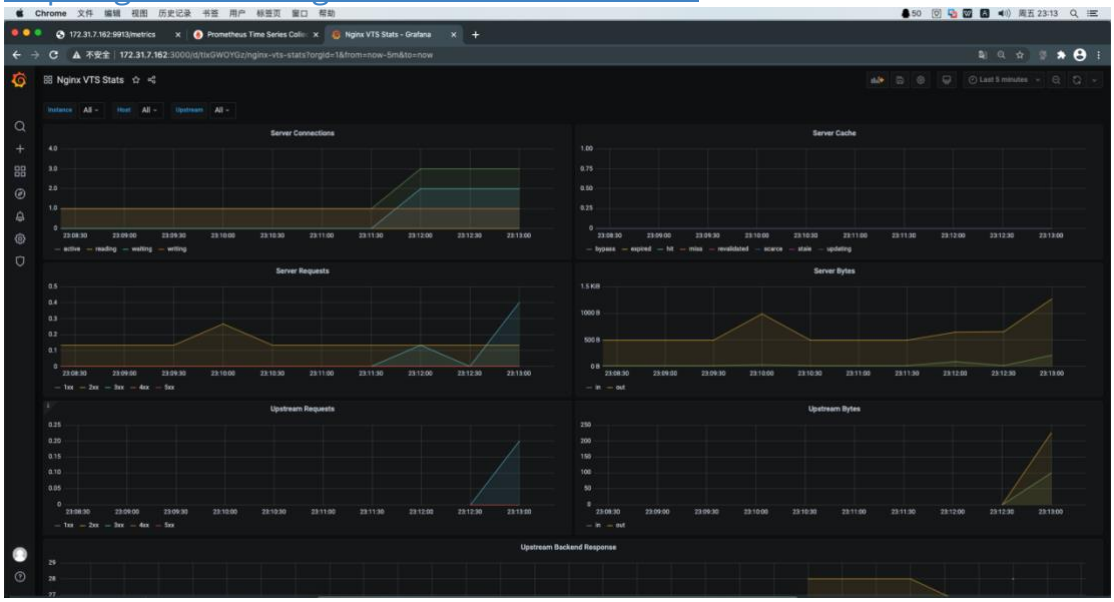
4.5.6：prometheus 添加 job：

```
  - job_name: mysql-monitor-172.31.2.182
```

```
    static_configs:
    - targets: ['172.31.2.182:9104']

  - job_name: 'nginx-monitor-serverA'
    static_configs:
    - targets: ['172.30.7.201:9913']
```

4.5.7：grafana 导入模板并验证：

https://grafana.com/grafana/dashboards/2949



六：pushgateway：

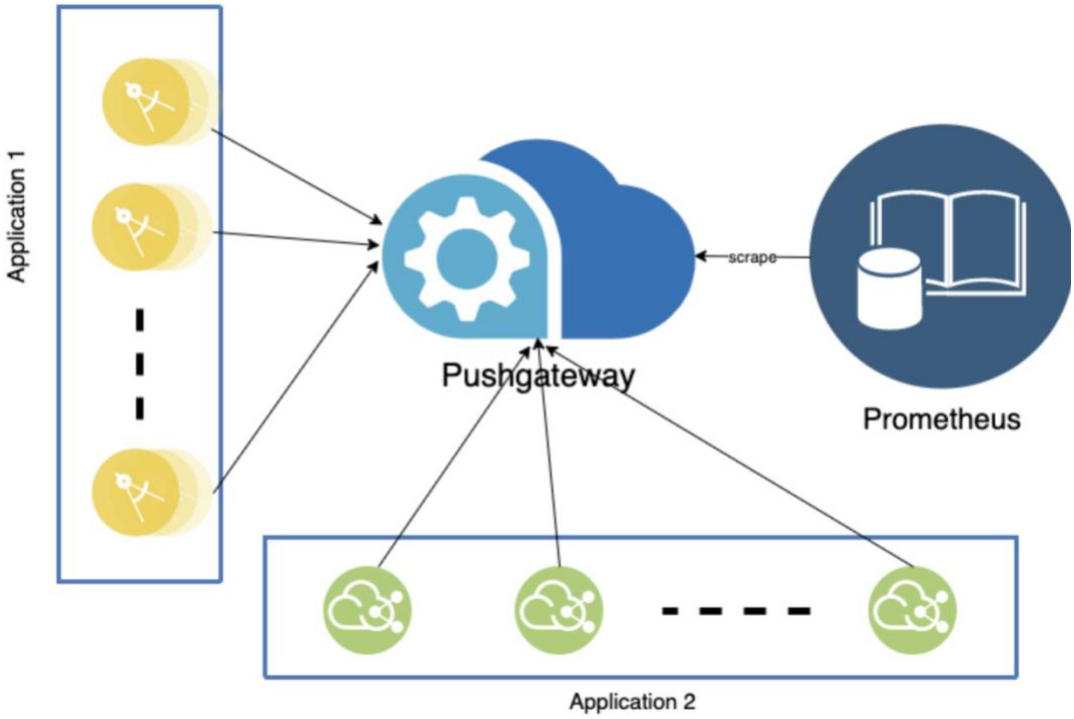
https://github.com/prometheus/pushgateway

6.1：pushgateway 简介：

pushgateway 是采用被动推送的方式，而不是类似于 prometheus server 主动连接 exporter 获取监控数据。

pushgateway 可以单独运行在一个节点，然后需要自定义监控脚本把需要监控的主动推送给 pushgateway 的 API 接口，然后 pushgateway 再等待 prometheus server 抓取数据，即 pushgateway 本身没有任何抓取监控数据的功能，目前 pushgateway 只是被动的等待数据从客户端推送过来。

--persistence.file="" #数据保存的文件，默认只保存在内存中
--persistence.interval=5m #数据持久化的间隔时间

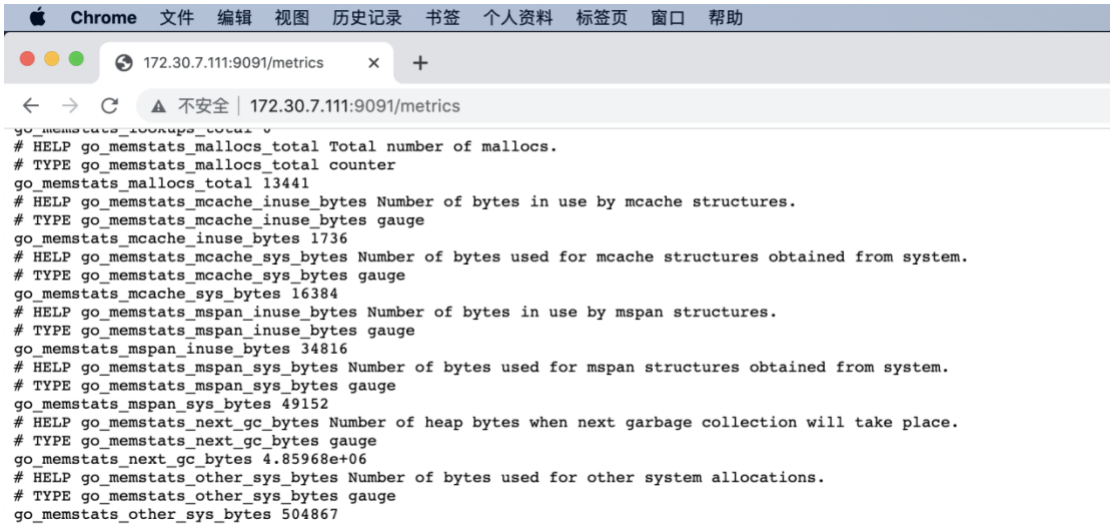


6.2：部署 pushgateway：

```
# docker load -i pushgateway.tar.gz
# docker run -d --name pushgateway -p 9091:9091 prom/pushgateway
f0d298f3ed831b2be06a2abd2cd63d6d0e013e8c8035290363f81e36e67e0785
```

6.3：prometheus 到 pushgateway 采集数据：

6.3.1：验证 pushgateway：

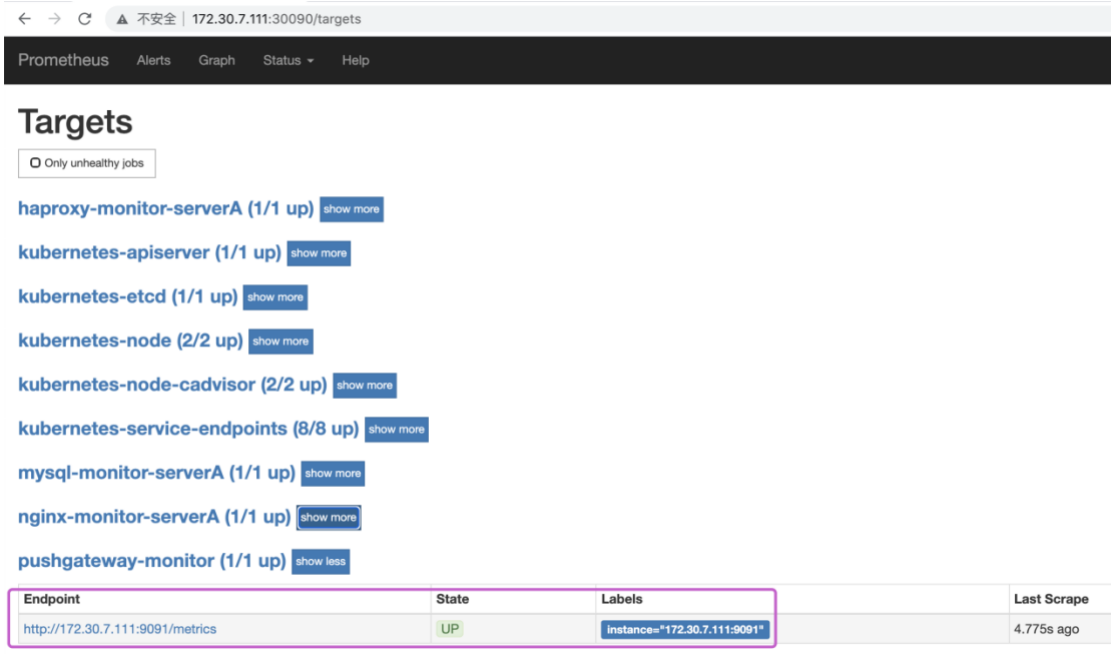



```
# HELP go_memstats_mallocs_total Total number of mallocs.
# TYPE go_memstats_mallocs_total counter
go_memstats_mallocs_total 13441
# HELP go_memstats_mcache_inuse_bytes Number of bytes in use by mcache structures.
# TYPE go_memstats_mcache_inuse_bytes gauge
go_memstats_mcache_inuse_bytes 1736
# HELP go_memstats_mcache_sys_bytes Number of bytes used for mcache structures obtained from system.
# TYPE go_memstats_mcache_sys_bytes gauge
go_memstats_mcache_sys_bytes 16384
# HELP go_memstats_mspan_inuse_bytes Number of bytes in use by mspan structures.
# TYPE go_memstats_mspan_inuse_bytes gauge
go_memstats_mspan_inuse_bytes 34816
# HELP go_memstats_mspan_sys_bytes Number of bytes used for mspan structures obtained from system.
# TYPE go_memstats_mspan_sys_bytes gauge
go_memstats_mspan_sys_bytes 49152
# HELP go_memstats_next_gc_bytes Number of heap bytes when next garbage collection will take place.
# TYPE go_memstats_next_gc_bytes gauge
go_memstats_next_gc_bytes 4.85968e+06
# HELP go_memstats_other_sys_bytes Number of bytes used for other system allocations.
# TYPE go_memstats_other_sys_bytes gauge
go_memstats_other_sys_bytes 504867
```

6.3.2：prometheus 配置数据采集：

```
# vim prometheus-cfg.yaml
      - job_name: 'nginx-monitor-serverA'
        static_configs:
        - targets: ['172.30.7.201:9913']
      - job_name: 'pushgateway-monitor'
        scrape_interval: 5s
        static_configs:
        - targets: ['172.30.7.111:9091']
        honor_labels: true


# honor_labels 控制 Prometheus 如何处理已经存在于已抓取数据中的标签与 Prometheus 将附加服务器端的标签之间的冲突（"job"和"instance"标签，手动配置的目标标签以及服务发现实现生成的标签）。
#  如果 honor_labels 设置为"true"，则通过保留已抓取数据的标签值并忽略冲突的服务器端标签来解决标签冲突。
#  如果 honor_labels 设置为"false"，则通过将已抓取数据中的冲突标签重命名为"exported_ <original-label>"（例如"exported_instance"，"exported_job"）然后附加服务器端标签来解决标签冲突

# kubectl apply -f prometheus-cfg.yaml && kubectl  delete  -f prometheus-deploy.yaml  && kubectl  apply -f prometheus-deploy.yaml
```

6.3.3：验证数据：



6.4：测试从客户端推送单条数据：

6.4.1：推送单条数据：

要 Push 数据到 PushGateway 中，可以通过其提供的 API 标准接口来添加，默认 URL 地址为：http://<ip>:9091/metrics/job/<JOBNAME>{/<LABEL_NAME>/<LABEL_VALUE>},

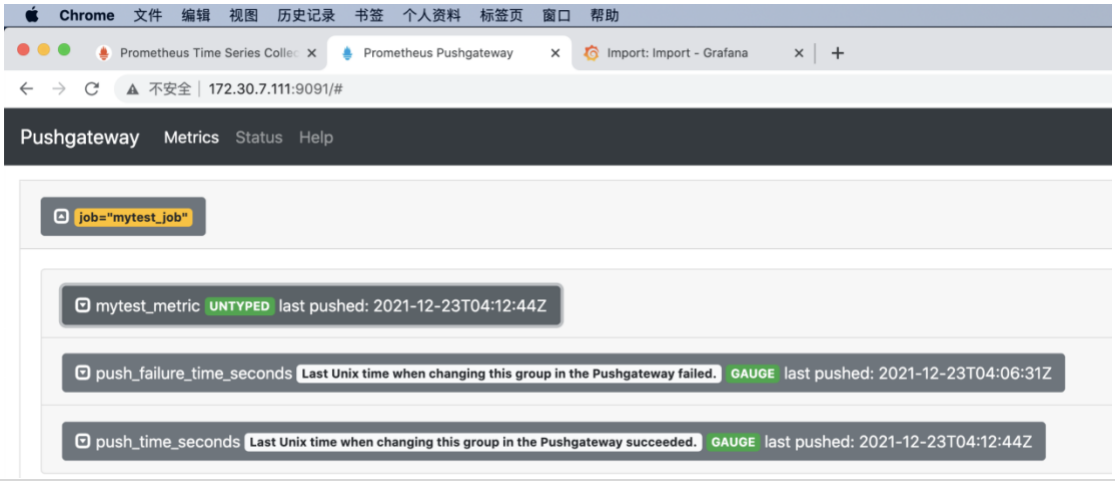
其中<JOBNAME>是必填项，为 job 标签值，后边可以跟任意数量的标签对，一般我们会添加一个 instance/<INSTANCE_NAME>实例名称标签，来方便区分各个指标。

推送一个 job 名称为 mytest_job，key 为 mytest_metric 值为 2022

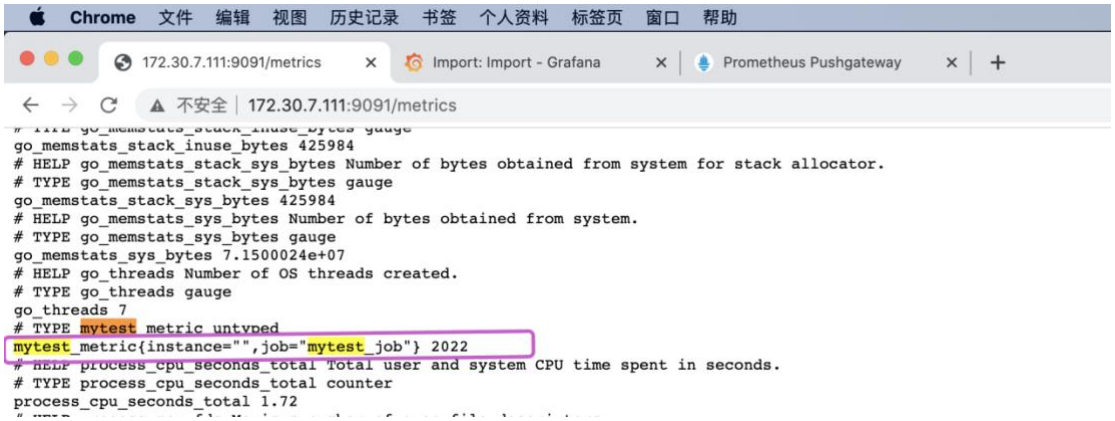
echo "mytest_metric 2022" | curl --data-binary @- http://172.30.7.111:9091/metrics/job/mytest_job

echo "mytest_metric 2026" | curl --data-binary @- http://172.30.7.111:9091/metrics/job/mytest_job

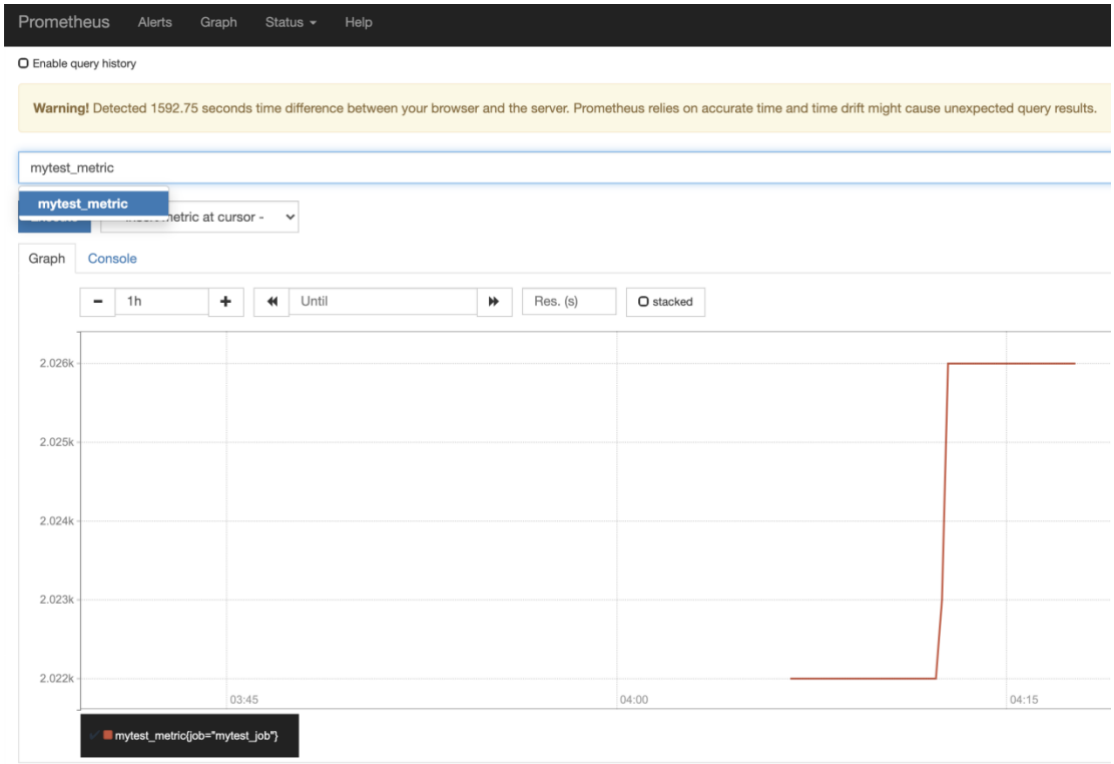
6.4.2：pushgateway 验证数据：



除了 mytest_metric 外，同时还新增了 push_time_seconds 和 push_failure_time_seconds 两个指标，这两个是 PushGateway 自动生成的指标，分别用于记录指标数据的成功上传时间和失败上传时间。



6.4.3：prometheus server 验证数据：

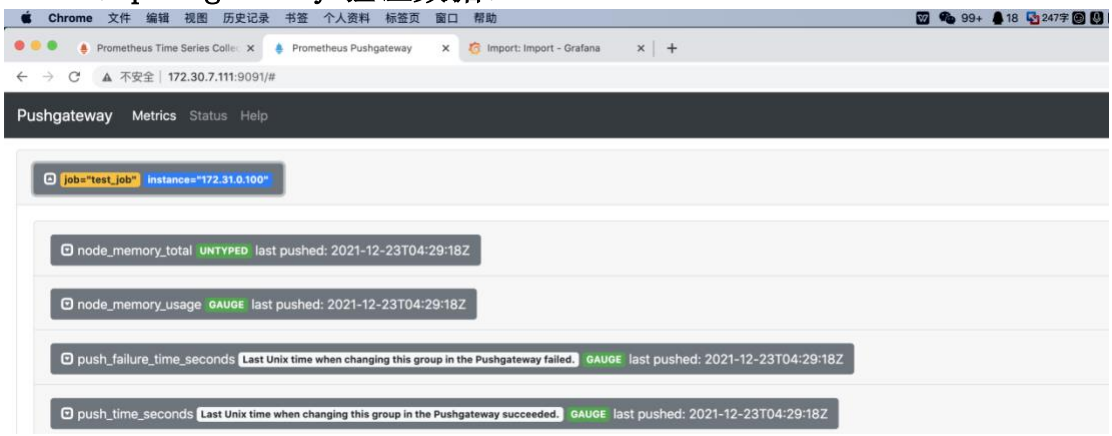


6.5：测试从客户端推送多条数据：

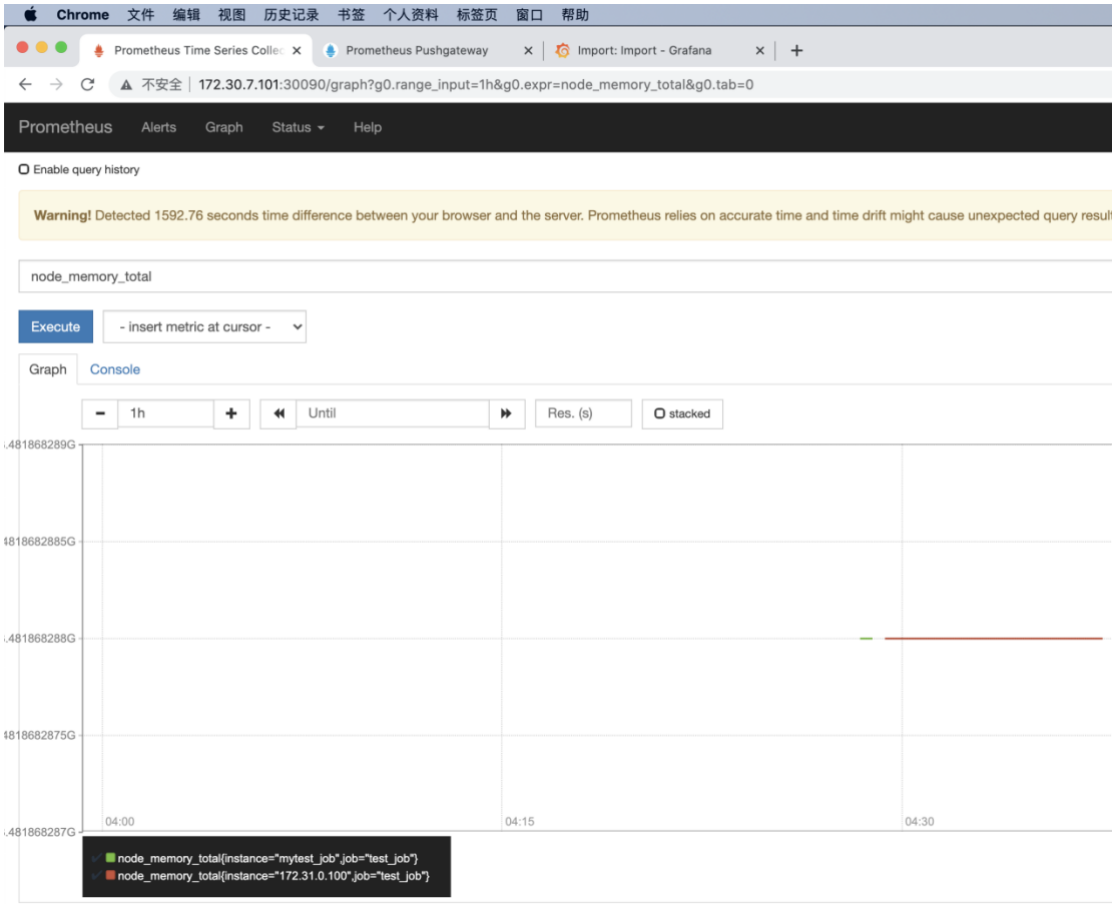
6.5.1：推送多条数据：

```
#cat <<EOF | curl --data-binary @- http://172.30.7.111:9091/metrics/job/test_job/instance/172.31.0.100
#TYPE node_memory_usage gauge
node_memory_usage 4311744512
# TYPE memory_total gauge
node_memory_total 103481868288
EOF
```

6.5.2：pushgateway 验证数据：



6.5.3：prometheus server 验证数据：



6.6：自定义收集数据：

基于自定义脚本实现数据的收集和推送

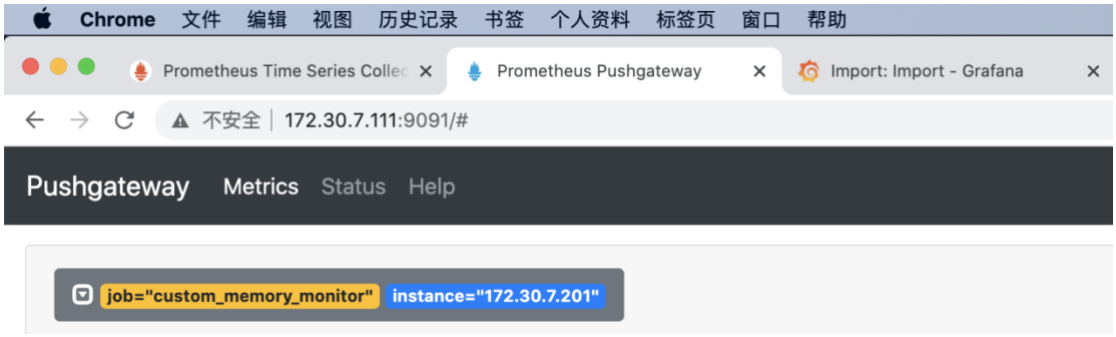
6.6.1：自定义脚本：

```
# cat mem_monitor.sh
#!/bin/bash

total_memory=$(free |awk '/Mem/{print $2}')
used_memory=$(free |awk '/Mem/{print $3}')

job_name="custom_memory_monitor"
instance_name=`ifconfig    eth0 | grep -w inet | awk '{print $2}'`
pushgateway_server="http://172.30.7.111:9091/metrics/job"

cat <<EOF | curl --data-binary @- ${pushgateway_server}/${job_name}/instance/${instance_name}
#TYPE custom_memory_total    gauge
custom_memory_total $total_memory
#TYPE custom_memory_used    gauge
custom_memory_used $used_memory
EO
```

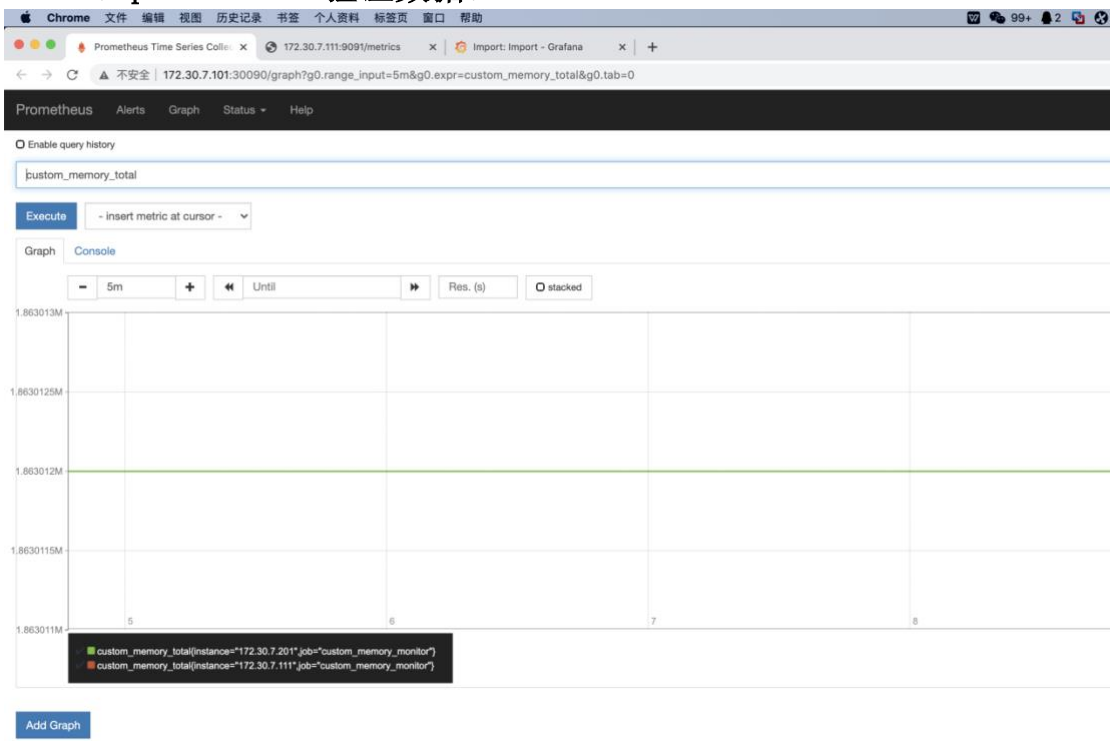
分别在不同主机执行脚本，验证指标数据收集和推送：

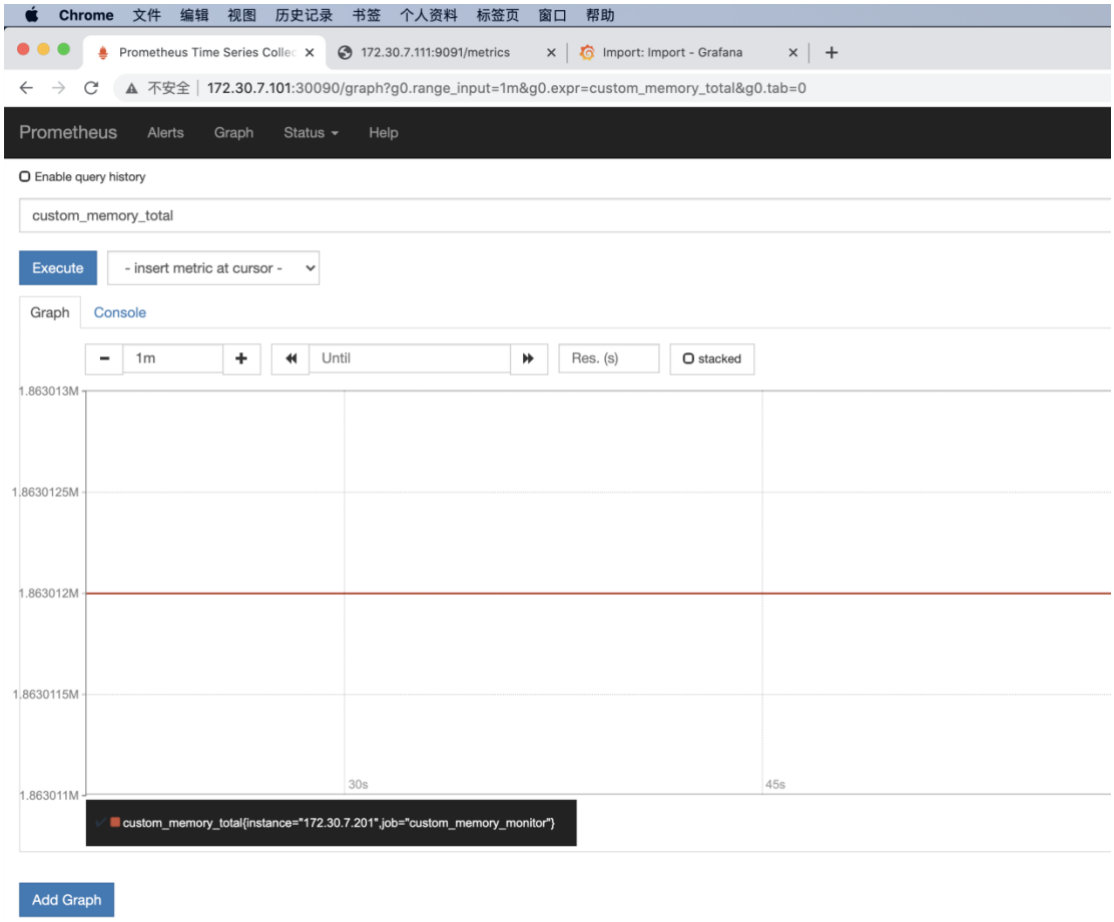
```
# bash mem_monitor.sh
```

6.6.2：pushgateway 验证数据：



6.6.3：prometheus 验证数据：





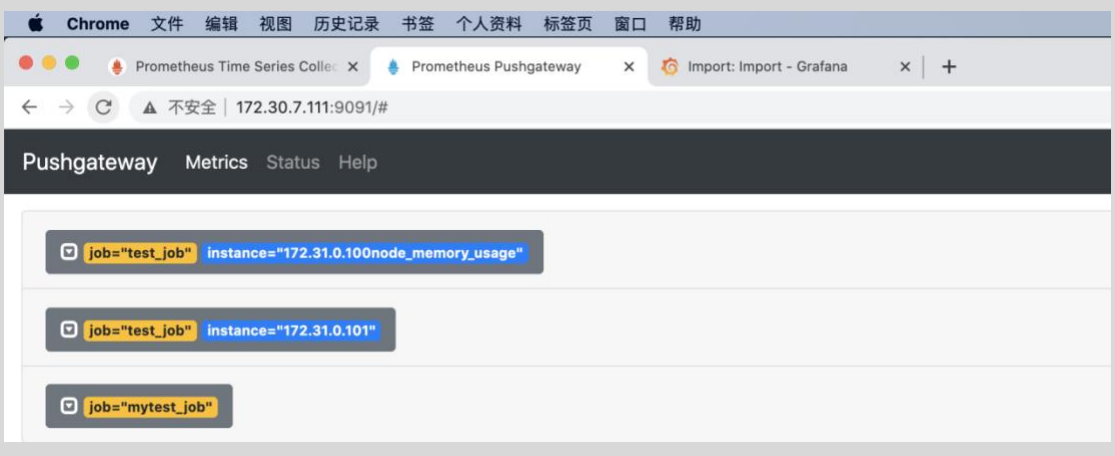
6.7：删除数据：

先对一个组写入多个 instance 的数据：

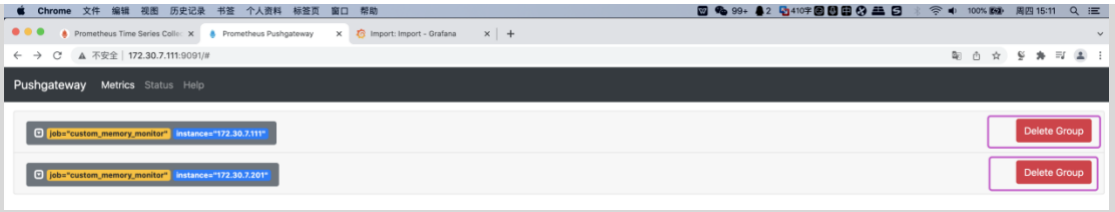
```
#cat <<EOF | curl --data-binary @- http://172.30.7.111:9091/metrics/job/test_job/instance/172.31.0.100
#TYPE node_memory_usage gauge
node_memory_usage 4311744512
# TYPE memory_total gauge
node_memory_total 103481868288
EOF

#cat <<EOF | curl --data-binary @- http://172.30.7.111:9091/metrics/job/test_job/instance/172.31.0.101
#TYPE node_memory_usage gauge
node_memory_usage 4311744512
# TYPE memory_total gauge
node_memory_total 103481868288
EOF
```

6.7.1：通过 API 删除指定组内指定实例的数据：

```
# curl -X DELETE http://172.30.7.111:9091/metrics/job/test_job/instance/172.31.0.100
```



6.7.2：通过 web 界面删除：



七.prometheus 联邦：

Prometheus Server 环境:

```
172.31.2.101 #主节点

172.31.2.102 #联邦节点 1
172.31.2.103 #联邦节点 2

172.31.2.181 #node1，成员 1 的采集服务器
172.31.2.182 #node2，成员 2 的采集服务器
```

7.1：部署 prometheus server：

Prometheus 主 server 和 prometheus 联邦 server 分别部署 prometheus

```
# pwd
/apps
# tar xvf prometheus-2.32.1.linux-amd64.tar.gz

# ln -sv /apps/prometheus-2.32.1.linux-amd64 /apps/prometheus
'/apps/prometheus' -> '/apps/prometheus-2.32.1.linux-amd64'

# cd /apps/prometheus
# ll
prometheus.yml #配置文件
```

```
prometheus #prometheus 服务可执行程序
promtool   #测试工具，用于检测配置 prometheus 配置文件、检测 metrics 数据等
        ./promtool check config prometheus.yml
                Checking prometheus.yml
        SUCCESS: 0 rule files found
# vim /etc/systemd/system/prometheus.service
[Unit]
Description=Prometheus Server
Documentation=https://prometheus.io/docs/introduction/overview/
After=network.target

[Service]
Restart=on-failure
WorkingDirectory=/apps/prometheus/
ExecStart=/apps/prometheus/prometheus      --config.file=/apps/prometheus/prometheus.yml

[Install]
WantedBy=multi-user.target

# systemctl    restart    prometheus
# systemctl    enable    prometheus
```

7.2：部署 node_exporter：

Node 节点(被监控节点)分别部署 node_exporter(exporter)

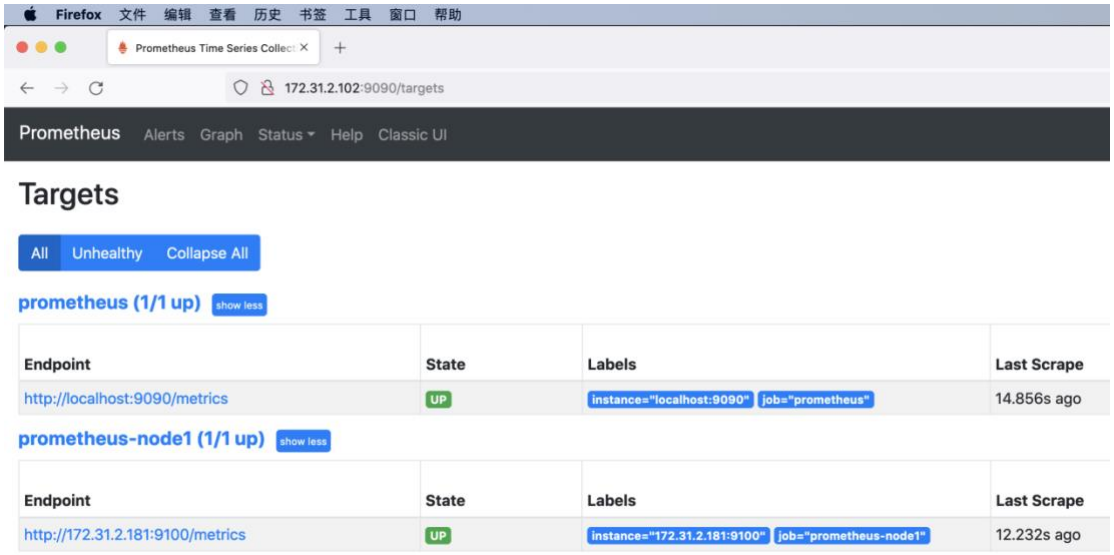
```
# tar xvf node_exporter-1.3.1.linux-amd64.tar.gz
# ln    -sv /apps/node_exporter-1.3.1.linux-amd64 /apps/node_exporter
'/apps/node_exporter' -> '/apps/node_exporter-1.3.1.linux-amd64'

# vim      /etc/systemd/system/node-exporter.service
[Unit]
Description=Prometheus Node Exporter
After=network.target

[Service]
ExecStart=/apps/node_exporter/node_exporter

[Install]
WantedBy=multi-user.target

#   systemctl      daemon-reload      &&   systemctl      restart      node-exporter   &&   systemctl      enable
node-exporter.service
```

14.3：配置联邦 server 监控 node_exporter：

分别在联邦节点 1 监控 node1，在联邦节点 2 监控 node2

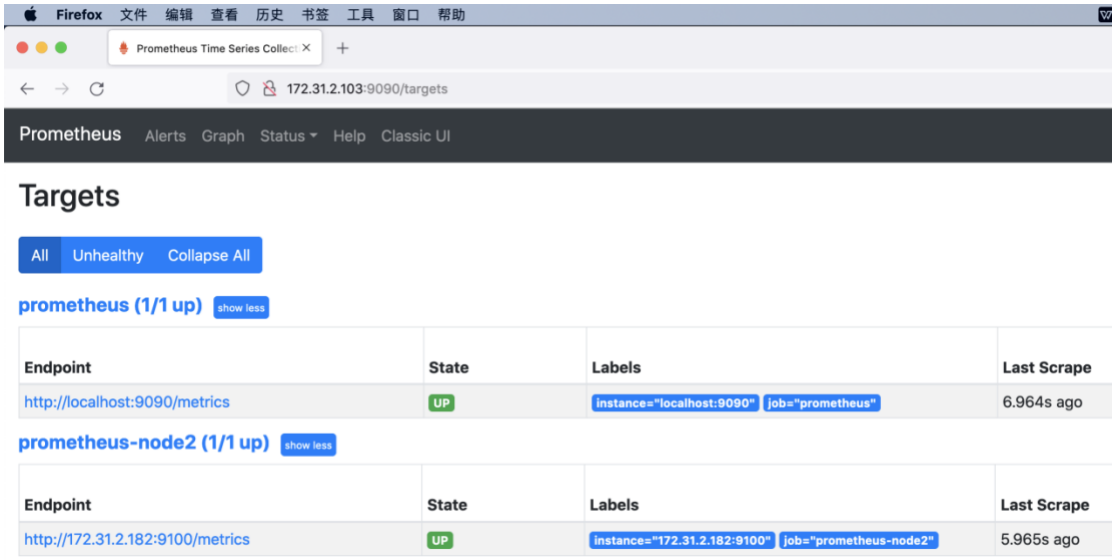
```
Prometheus  联邦节点 1：
# vim /apps/prometheus/prometheus.yml

  - job_name: "prometheus-node1"
    static_configs:
      - targets: ["172.31.2.181:9100"]

# systemctl   restart   prometheus.service
```

验证数据：




```
Prometheus  联邦节点 2：
# vim /apps/prometheus/prometheus.yml
  - job_name: "prometheus-node2"
    static_configs:
      - targets: ["172.31.2.182:9100"]

# systemctl   restart   prometheus.service
```

验证数据：



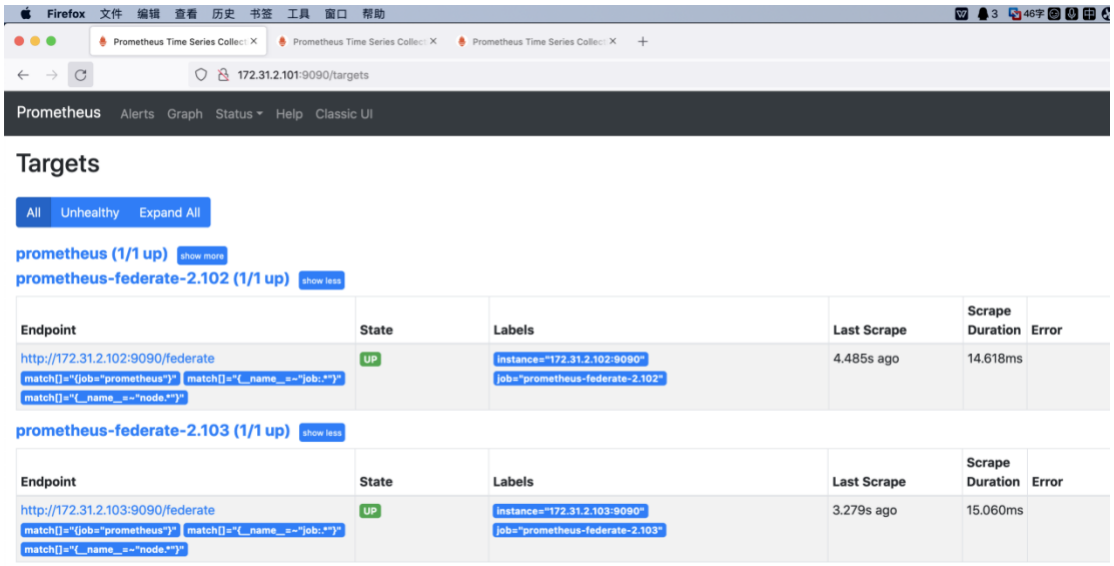
7.4：prometheus server 采集联邦 server：

```
- job_name: "prometheus"
    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.
    static_configs:
      - targets: ["localhost:9090"]

- job_name: 'prometheus-federate-2.102'
    scrape_interval: 10s
    honor_labels: true
    metrics_path: '/federate'
    params:
      'match[]':
        - '{job="prometheus"}'
        - '{__name__=~"job:.*"}'
        - '{__name__=~"node.*"}'
    static_configs:
    - targets:
      - '172.31.2.102:9090'

- job_name: 'prometheus-federate-2.103'
    scrape_interval: 10s
    honor_labels: true
    metrics_path: '/federate'
    params:
      'match[]':
        - '{job="prometheus"}'
        - '{__name__=~"job:.*"}'
        - '{__name__=~"node.*"}'
    static_configs:
```

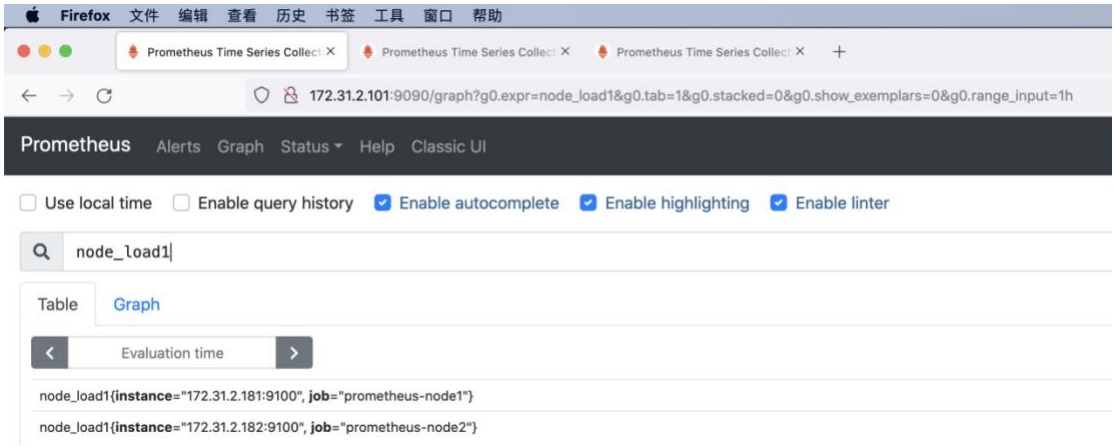
```
    - targets:
      - '172.31.2.103:9090'


# systemctl   restart   prometheus.service
```

7.5：验证 prometheus server：

7.5.1：验证数据收集状态：



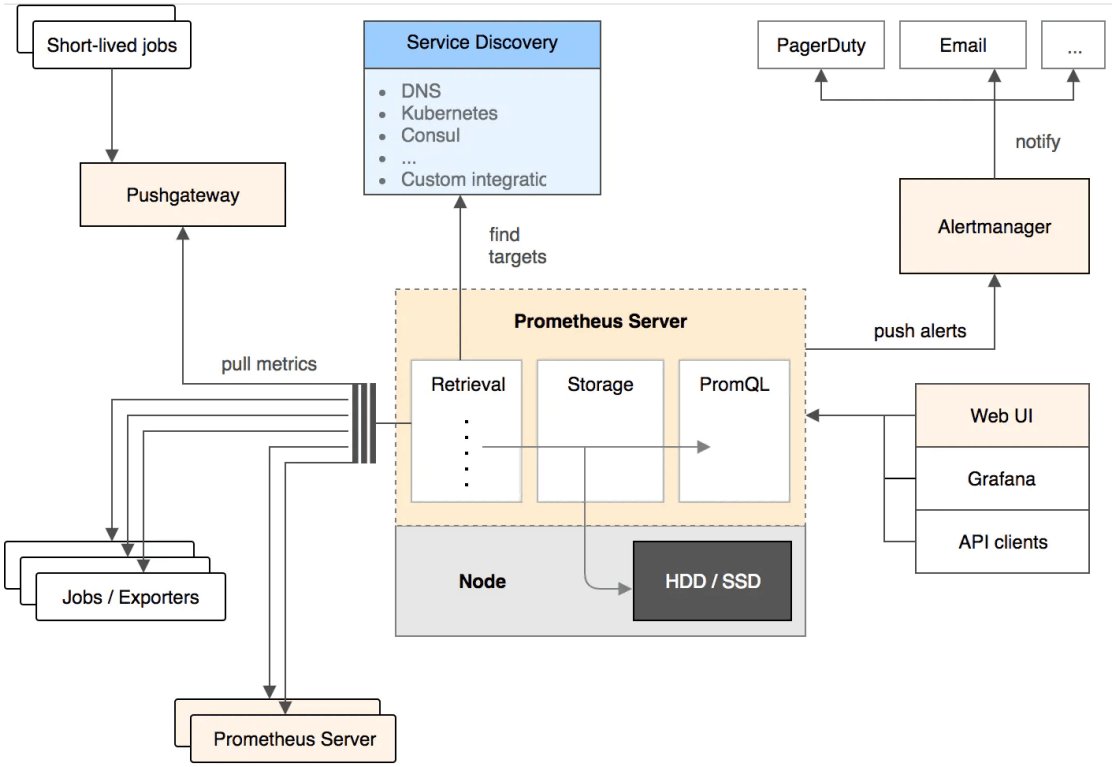
7.5.2：验证指标数据：



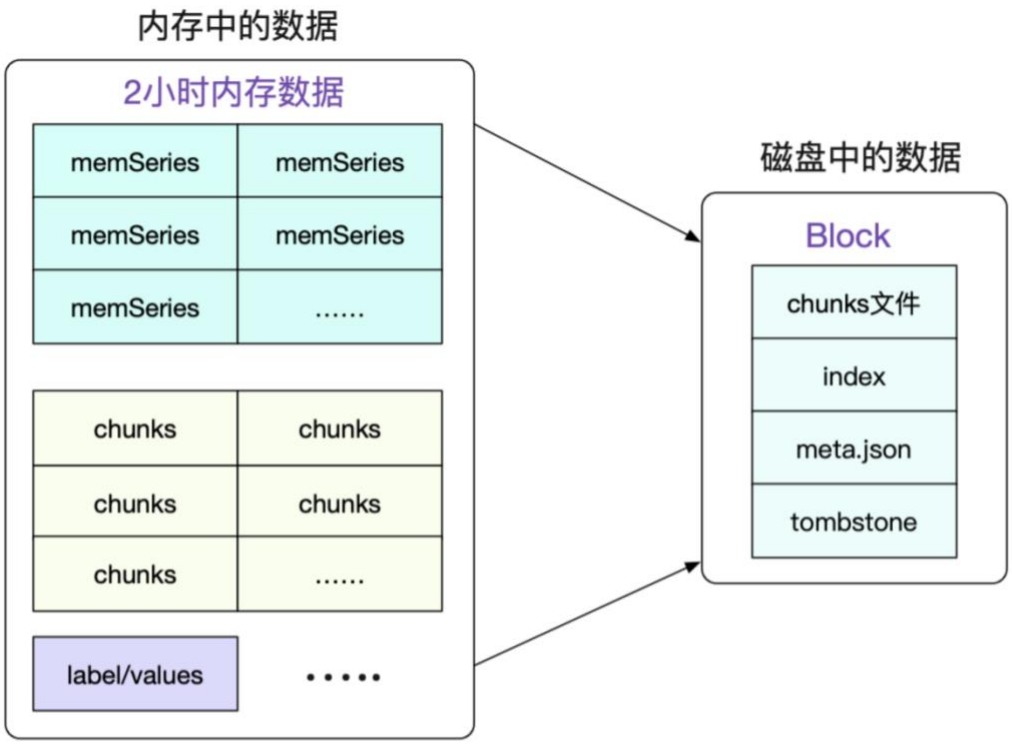
八：prometheus 存储系统：

Prometheus 有着非常高效的时间序列数据存储方法，每个采样数据仅仅占用 3.5byte 左右空间，上百万条时间序列，30 秒间隔，保留 60 天，大概 200 多 G 空间（引用官方 PPT）。

8.1：prometheus 本地存储简介：



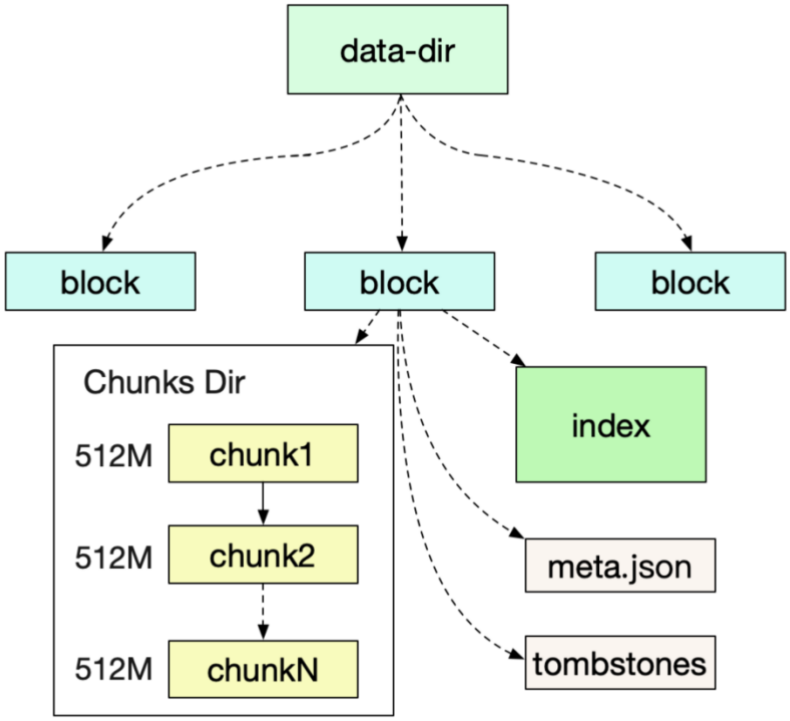
默认情况下，prometheus 将采集到的数据存储在本地的 TSDB 数据库中，路径默认为 prometheus 安装目录的 data 目录，数据写入过程为先把数据写入 wal 日志并放在内存，然后 2 小时后将内存数据保存至一个新的 block 块，同时再把新采集的数据写入内存并在 2 小时后再保存至一个新的 block 块，以此类推。



8.1.1：block 简介：

每个 block 为一个 data 目录中以 01 开头的存储目录，如下：

```
~# ls -l /apps/prometheus/data/
total 4
drwxr-xr-x 3 root root      68 Dec 24 05:00 01FQNCYZ0BPFA8AQDDZM1C5PRN      #block
drwxr-xr-x 3 root root      68 Dec 27 09:10 01FQXJESTQXVBEPM857SF9XBX7       #block
drwxr-xr-x 3 root root      68 Dec 28 07:20 01FQZYJJ605MEQ7KHBBM09PBVH      #block
drwxr-xr-x 3 root root      68 Dec 28 07:20 01FQZYJJABE4JGM85TXJ25WESQ      #block
```



8.1.2：block 的特性：

block 会压缩、合并历史数据块，以及删除过期的块，随着压缩、合并，block 的数量会减少，在压缩过程中会发生三件事：定期执行压缩、合并小的 block 到大的 block、清理过期的块。

每个块有 4 部分组成：

```
~# tree /apps/prometheus/data/01FQNCYZ0BPFA8AQDDZM1C5PRN/
/apps/prometheus/data/01FQNCYZ0BPFA8AQDDZM1C5PRN/
├── chunks
│   └── 000001 #数据目录,每个大小为 512MB 超过会被切分为多个
├── index          #索引文件，记录存储的数据的索引信息，通过文件内的几个表来查找时序数据
├── meta.json      #block 元数据信息，包含了样本数、采集数据数据的起始时间、压缩历史
└── tombstones     #逻辑数据，主要记载删除记录和标记要删除的内容，删除标记，可在查询块时排除样本。
```

8.1.3：本地存储配置参数：

```
--config.file="prometheus.yml" #指定配置文件
--web.listen-address="0.0.0.0:9090" #指定监听地址
--storage.tsdb.path="data/" #指定数存储目录
```

--storage.tsdb.retention.size=B, KB, MB, GB, TB, PB, EB　#指定 chunk 大小，默认 512MB
--storage.tsdb.retention.time= #数据保存时长，默认 15 天

--query.timeout=2m #最大查询超时时间
-query.max-concurrency=20　#最大查询并发数

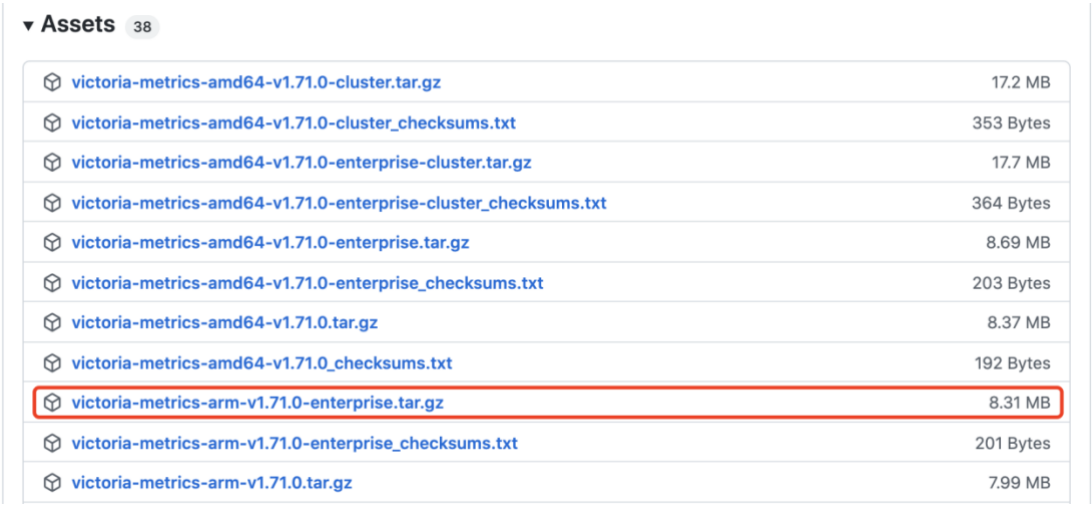
--web.read-timeout=5m #最大空闲超时时间
--web.max-connections=512 #最大并发连接数
--web.enable-lifecycle #启用 API 动态加载配置功能

8.2：远端存储之--victoriametrics：

https://github.com/VictoriaMetrics/VictoriaMetrics
https://docs.victoriametrics.com/Single-server-VictoriaMetrics.html

8.2.1：单机：

下载：



8.2.1.1：部署单机版：

 # tar xvf victoria-metrics-amd64-v1.71.0.tar.gz

参数：

-httpListenAddr=0.0.0.0:8428 #监听地址及端口

-storageDataPath　#VictoriaMetrics 将所有数据存储在此目录中，默认为执行启动 victoria 的当前目录下的 victoria-metrics-data 目录中。

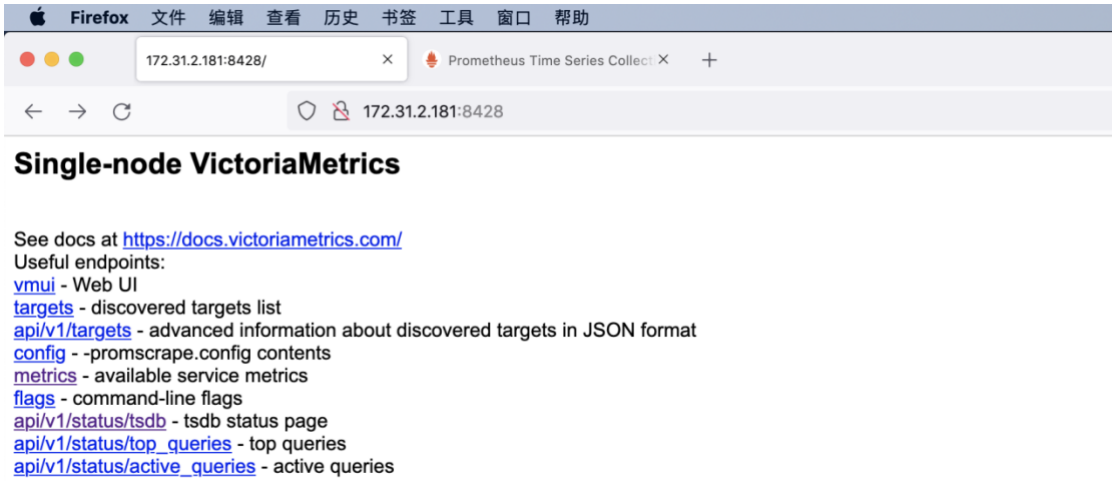
-retentionPeriod #存储数据的保留，较旧的数据会自动删除，默认保留期为 1 个月，默认单位为 m(月)，支持的单位有 h (hour), d (day), w (week), y (year)。

 # mv victoria-metrics-prod /usr/local/bin/

service 启动文件：

```
# cat /etc/systemd/system/victoria-metrics-prod.service
[Unit]
Description=For Victoria-metrics-prod Service
After=network.target

[Service]
ExecStart=/usr/local/bin/victoria-metrics-prod -httpListenAddr=0.0.0.0:8428
-storageDataPath=/data/victoria -retentionPeriod=3

[Install]
WantedBy=multi-user.target

# systemctl daemon-reload && systemctl restart victoria-metrics-prod.service
# systemctl enable victoria-metrics-prod.service
```

验证 web 页面：

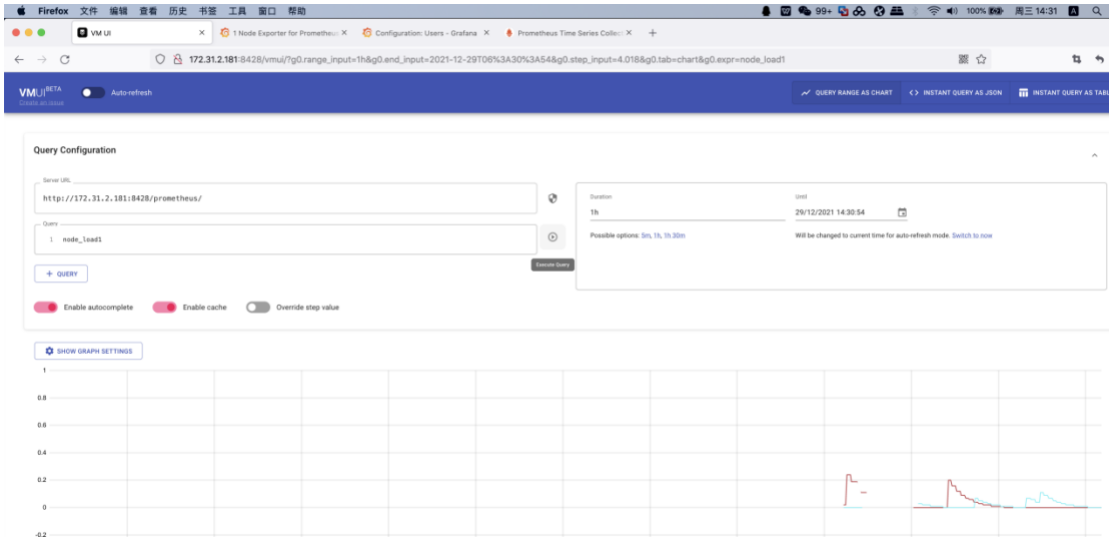


8.2.1.2：prometheus 设置：

```
global:
  scrape_interval: 15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

remote_write:
  - url: http://172.31.2.181:8428/api/v1/write
```

8.2.1.3：验证 VictoriaMetrics 数据：

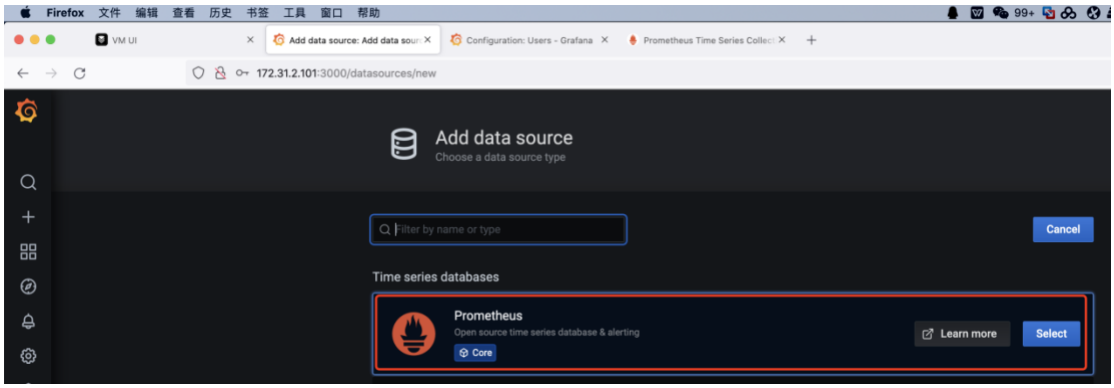
http://172.31.2.181:8428/



8.2.1.4：grafana 设置：

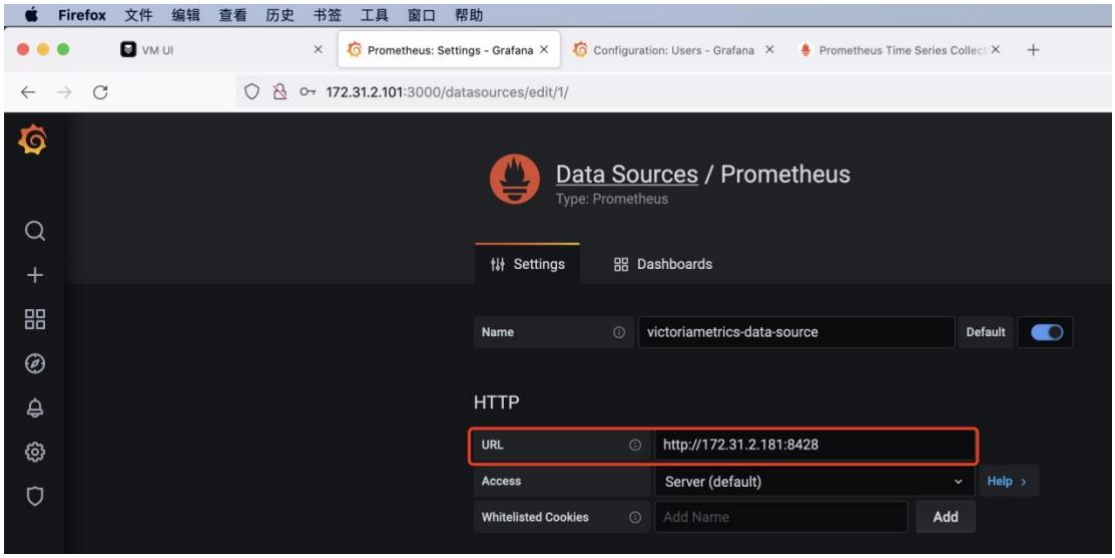
添加数据源：

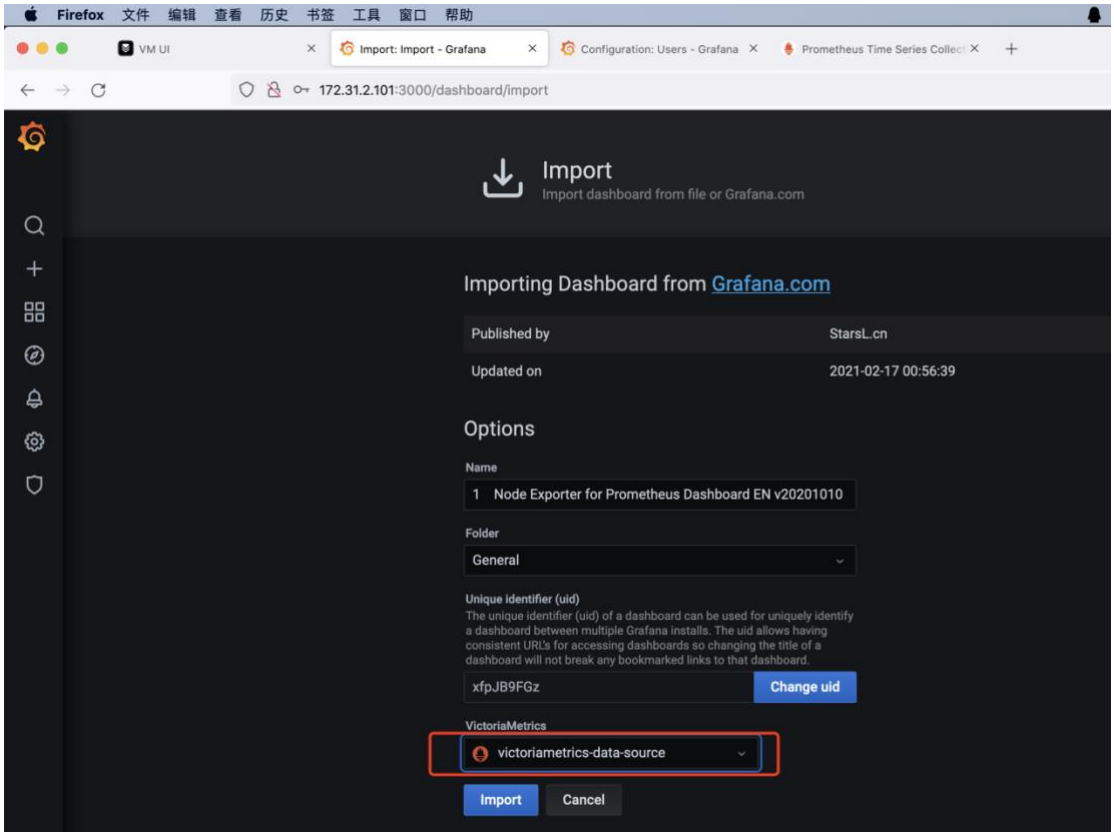
类型为 prometheus，地址及端口为 VictoriaMetrics：



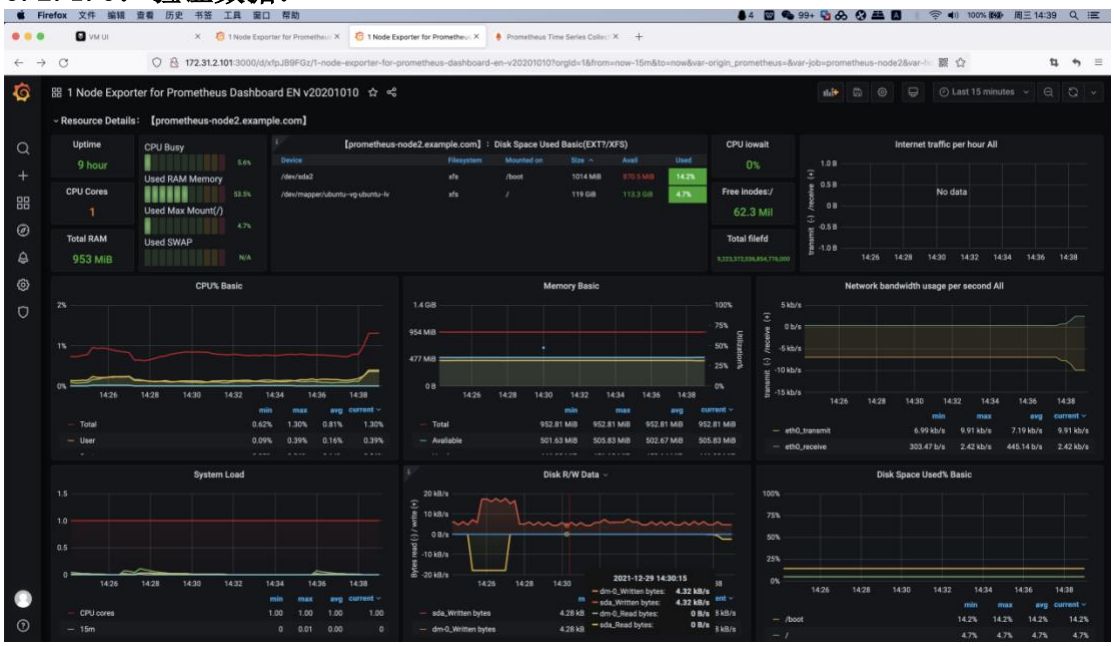
导入指定模板：

8919





8.2.1.5：验证数据：



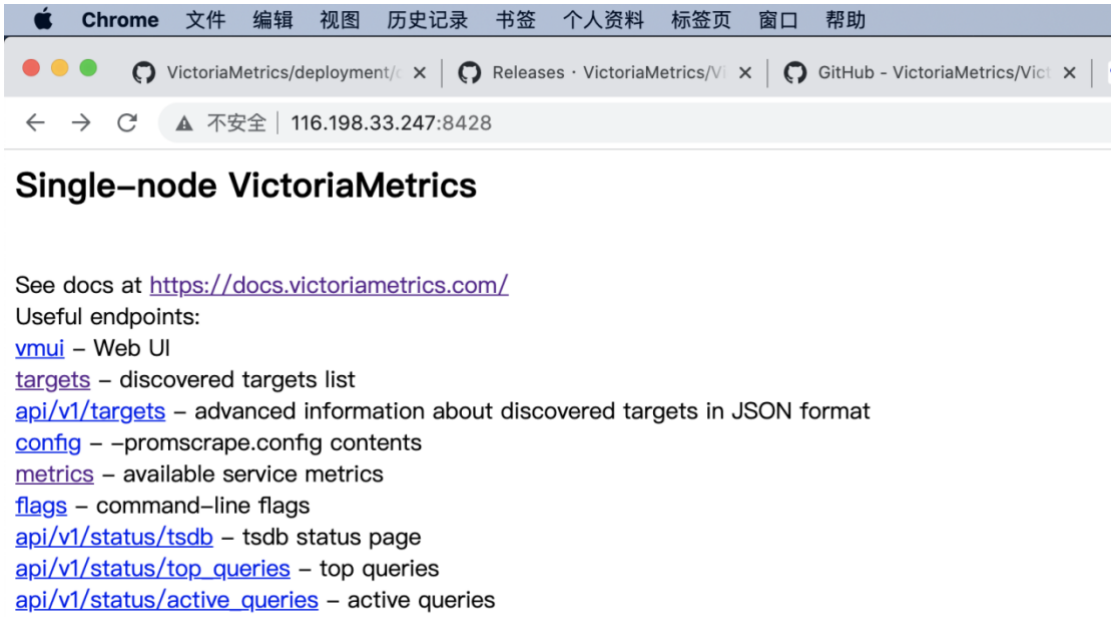
8.2.2：官方 docker-compsoe：

https://github.com/VictoriaMetrics/VictoriaMetrics/tree/cluster/deployment/docker

```
# git clone https://github.com/VictoriaMetrics/VictoriaMetrics.git
# cd VictoriaMetrics/deployment/docker/

# ls -l
total 56
```

```
-rw-r--r-- 1 root root       61 Dec 29 10:48 alertmanager.yml
-rw-r--r-- 1 root root 16487 Dec 29 10:48 alerts.yml
drwxr-xr-x 2 root root    4096 Dec 29 10:48 base
drwxr-xr-x 2 root root    4096 Dec 29 10:48 builder
-rw-r--r-- 1 root root    2807 Dec 29 10:48 docker-compose.yml
-rw-r--r-- 1 root root    5312 Dec 29 10:48 Makefile
-rw-r--r-- 1 root root     298 Dec 29 10:48 prometheus.yml
drwxr-xr-x 4 root root    4096 Dec 29 10:48 provisioning
-rw-r--r-- 1 root root    1488 Dec 29 10:48 README.md

# docker-compose up -d
```

验证 web 界面：



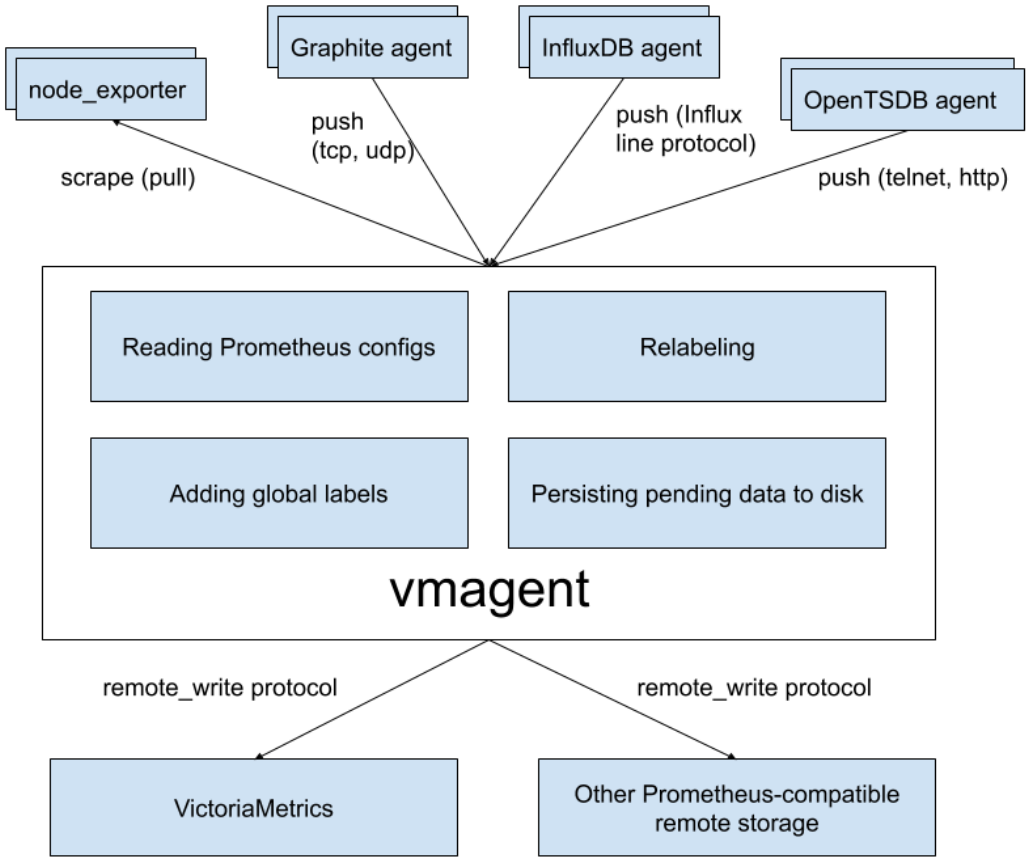
8.2.3：集群版本：

8.2.3.1：组件介绍：

vminsert　#写入组件(写)，vminsert 负责接收数据写入并根据对度量名称及其所有标签的一致 hash 结果将数据分散写入不同的后端 vmstorage 节点之间 vmstorage，vminsert 默认端口 8480

vmstorage #存储原始数据并返回给定时间范围内给定标签过滤器的查询数据，默认端口 8482
vmselect　#查询组件(读)，连接 vmstorage ，默认端口 8481

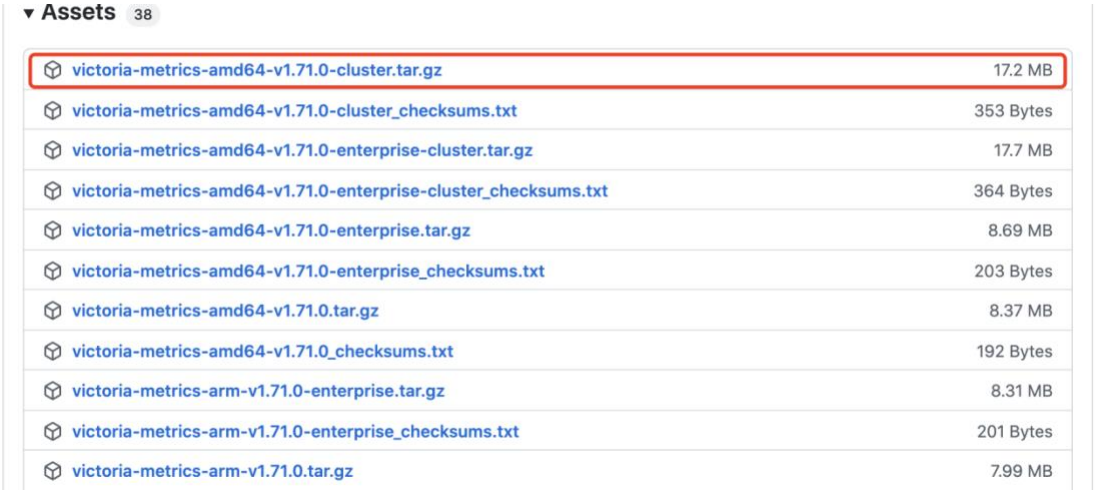
其它可选组件：
vmagent　#是一个很小但功能强大的代理，它可以从 node_exporter 各种来源收集度量数据，并将它们存储在 VictoriaMetrics 或任何其他支持远程写入协议的与 prometheus 兼容的存储系统中，有替代 prometheus server 的意向。



vmalert：　替换 prometheus server，以 VictoriaMetrics 为数据源，基于兼容 prometheus 的告警规则，判断数据是否异常，并将产生的通知发送给 alertermanager

Vmgateway：　读写 VictoriaMetrics 数据的代理网关，可实现限速和访问控制等功能，目前为企业版组件
vmctl：　VictoriaMetrics 的命令行工具，目前主要用于将 prometheus、opentsdb 等数据源的数据迁移到 VictoriaMetrics。

下载：



8.2.3.2：部署集群：

分别在各个 VictoriaMetrics 服务器进行安装配置：

```
# tar    xvf victoria-metrics-amd64-v1.71.0-cluster.tar.gz
vminsert-prod
vmselect-prod
vmstorage-prod
# mv vminsert-prod      vmselect-prod      vmstorage-prod    /usr/local/bin/

主要参数:
   -httpListenAddr string
         Address to listen for http connections (default ":8482")
   -vminsertAddr string
         TCP address to accept connections from vminsert services (default ":8400")
   -vmselectAddr string
         TCP address to accept connections from vmselect services (default ":8401")
```

8.2.3.2.1：部署 vmstorage-prod 组件：

```
# vim    /etc/systemd/system/vmstorage.service

[Unit]
Description=Vmstorage Server
After=network.target

[Service]
Restart=on-failure
WorkingDirectory=/tmp
ExecStart=/usr/local/bin/vmstorage-prod          -loggerTimezone          Asia/Shanghai          -storageDataPath
/data/vmstorage-data -httpListenAddr :8482 -vminsertAddr :8400 -vmselectAd
dr :8401

[Install]
WantedBy=multi-user.target
```

```
# systemctl restart vmstorage.service
# systemctl enable vmstorage.service
# systemctl status    vmstorage.service

# scp /etc/systemd/system/vmstorage.service    172.31.2.182:/etc/systemd/system/vmstorage.service
# scp /etc/systemd/system/vmstorage.service    172.31.2.183:/etc/systemd/system/vmstorage.service

# scp /usr/local/bin/vm*    172.31.2.182:/usr/local/bin/
# scp /usr/local/bin/vm*    172.31.2.183:/usr/local/bin/

182:
# systemctl    daemon-reload && systemctl    start vmstorage && systemctl    enable vmstorage

183:
# systemctl    daemon-reload && systemctl    start vmstorage && systemctl    enable vmstorage
```

8.2.3.2.2：部署 vminsert-prod 组件：

```
# vim /etc/systemd/system/vminsert.service
[Unit]
Description=Vminsert Server
After=network.target

[Service]
Restart=on-failure
WorkingDirectory=/tmp
ExecStart=/usr/local/bin/vminsert-prod                    -httpListenAddr                    :8480
-storageNode=172.31.2.181:8400,172.31.2.182:8400,172.31.2.183:8400

[Install]
WantedBy=multi-user.target

# systemctl    daemon-reload &&    systemctl    restart    vminsert &&    systemctl    enable vminsert

# scp /etc/systemd/system/vminsert.service    172.31.2.182:/etc/systemd/system/vminsert.service
# scp /etc/systemd/system/vminsert.service    172.31.2.183:/etc/systemd/system/vminsert.service
```

8.2.3.2.3：部署 vmselect-prod 组件：

```
# vim /etc/systemd/system/vmselect.service
[Unit]
Description=Vminsert Server
After=network.target

[Service]
```

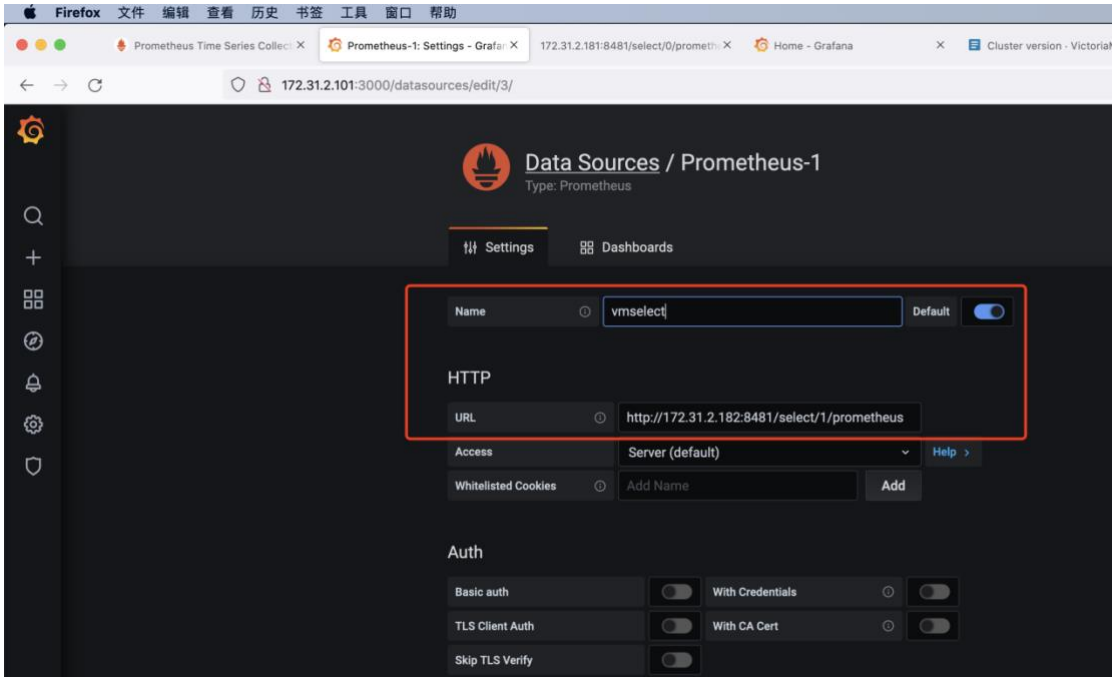
```
Restart=on-failure
WorkingDirectory=/tmp
ExecStart=/usr/local/bin/vmselect-prod -httpListenAddr :8481
-storageNode=172.31.2.181:8401,172.31.2.182:8401,172.31.2.183:8401

[Install]
WantedBy=multi-user.target

# systemctl daemon-reload && systemctl restart vmselect && systemctl enable vmselect

# scp /etc/systemd/system/vmselect.service 172.31.2.183:/etc/systemd/system/vmselect.service
# scp /etc/systemd/system/vmselect.service 172.31.2.183:/etc/systemd/system/vmselect.service

182:
systemctl daemon-reload && systemctl restart vmselect && systemctl enable vmselect
183:
systemctl daemon-reload && systemctl restart vmselect && systemctl enable vmselect
```

8.2.3.2.4：验证服务端口：

```
172.31.2.181：
# curl http://172.31.2.181:8480/metrics
# curl http://172.31.2.181:8481/metrics
# curl http://172.31.2.181:8482/metrics

172.31.2.182：
curl http://172.31.2.182:8480/metrics
curl http://172.31.2.182:8481/metrics
curl http://172.31.2.182:8482/metrics


172.31.2.183：
curl http://172.31.2.183:8480/metrics
curl http://172.31.2.183:8481/metrics
curl http://172.31.2.183:8482/metrics
```

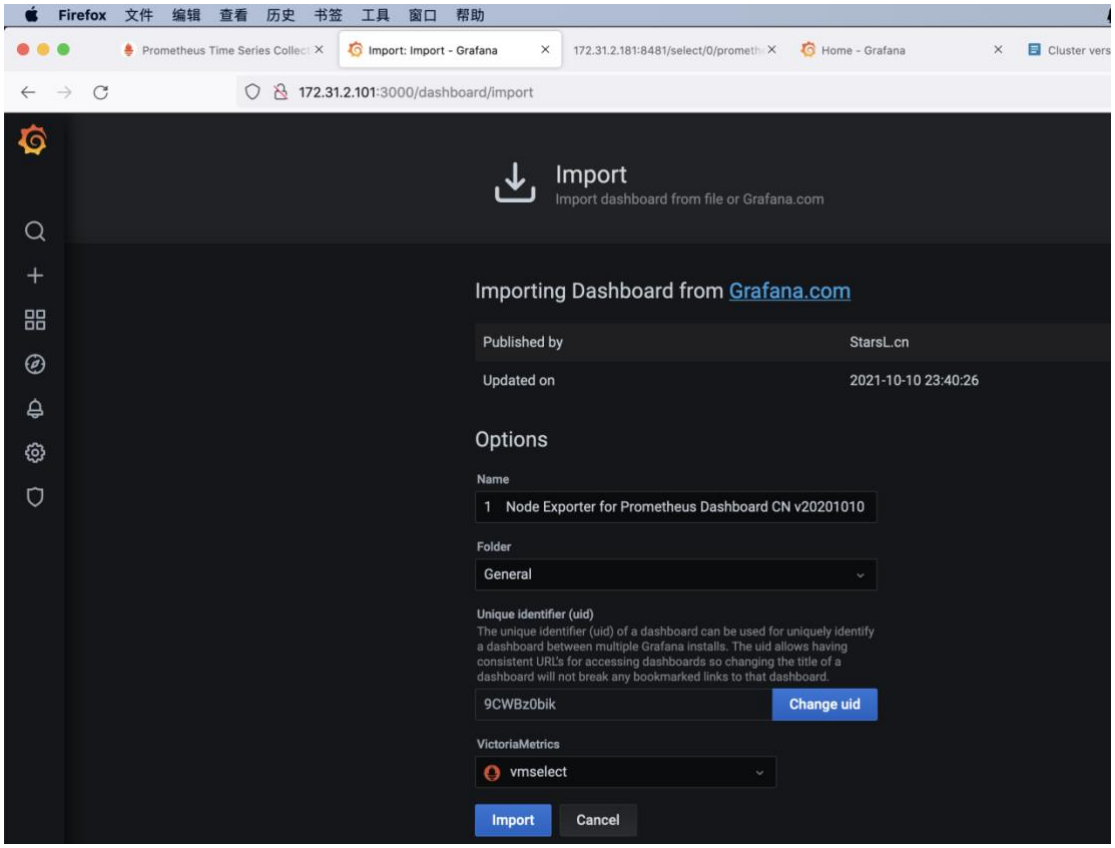
8.2.3.3：grafana 数据源配置：
https://github.com/VictoriaMetrics/VictoriaMetrics#grafana-setup

8.2.3.4：添加数据源：
http://172.31.2.182:8481/select/1/prometheus

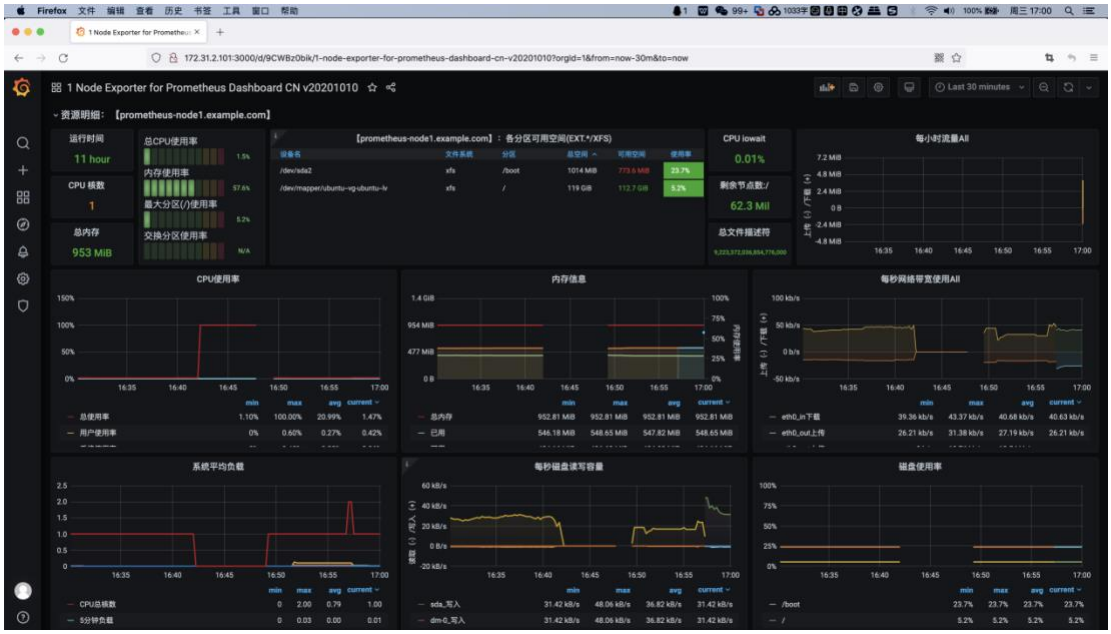


导入指定模板：

13824



验证数据：



8.2.4：开启数据复制：

https://docs.victoriametrics.com/Cluster-VictoriaMetrics.html#replication-and-data-safety

默认情况下，数据被 vminsert 的组件基于 hash 算法分别将数据持久化到不同的 vmstorage 节点，可以启用 vminsert 组件支持的-replicationFactor=N 复制功能，将数据分别在各节点保存一份完整的副本以实现数据的高可用。