

实验6report

问答题

- Namespace技术

用来修改进程视图的主要方法。如**PID Namespace**它是 Linux 创建新进程的一个可选参数，当我们用 clone() 系统调用创建一个新进程时，就可以在参数中指定 CLONE_NEWPID 参数，这时，新创建的这个进程将会“看到”一个全新的进程空间，在这个进程空间里，它的 PID 是 1。

- Cgroups

Cgroups 就是内核中用来为进程设置资源限制的一个重要功能。它的最主要的作用，就是限制一个进程组能够使用的资源上限，包括 CPU、内存、磁盘、网络带宽等等。此外，Cgroups 还能够对进程进行优先级设置、审计，以及将进程挂起和恢复等操作。

- 容器

容器是一个“单进程”模型。容器的本质就是一个进程

- 镜像

传统虚拟机的镜像大多是一个磁盘的“快照”，磁盘有多大，镜像就至少有多大。容器镜像只是一个操作系统的所有文件和目录，并不包含内核，最多也就几百兆。

- service及其作用

Service是由于Kubernetes中一个应用服务会有一个或多个实例（Pod）,每个实例（Pod）的IP地址由网络插件动态随机分配（Pod重启后IP地址会改变），为屏蔽这些后端实例的动态变化和对多实例的负载均衡而引入的资源对象。Service是一组逻辑pod的抽象，为一组pod提供统一入口，用户只需与service打交道，service提供DNS解析名称，负责追踪pod动态变化并更新转发表，通过负载均衡算法最终将流量转发到后端的pod。

- Prometheus工作流程

1. Prometheus server 定期从配置好的 jobs 或者 exporters 中拉 metrics，或者接收来自 Pushgateway 发过来的 metrics，或者从其他的 Prometheus server 中拉 metrics。
2. Prometheus server 在本地存储收集到的 metrics，并运行已定义好的 alert.rules，记录新的时间序列或者向 Alertmanager 推送警报。
3. Alertmanager 根据配置文件，对接收到的警报进行处理，发出告警。
4. 在图形界面中，可视化采集数据

实验题

- service配置成功

```
root@ubuntu:/usr/local/example/deploy# kubectl get svc
NAME                TYPE        CLUSTER-IP      EXTERNAL-IP  PORT(S)          AGE
example-service     ClusterIP   10.101.247.90    <none>       80/TCP           33h
kubernetes           ClusterIP   10.96.0.1        <none>       443/TCP          6d2
3h
prometheus          NodePort    10.105.4.204     <none>       9090:30177/TCP   13m
root@ubuntu:/usr/local/example/deploy# curl 10.101.247.90/abc
there is env NUM. Computation succeeded
```

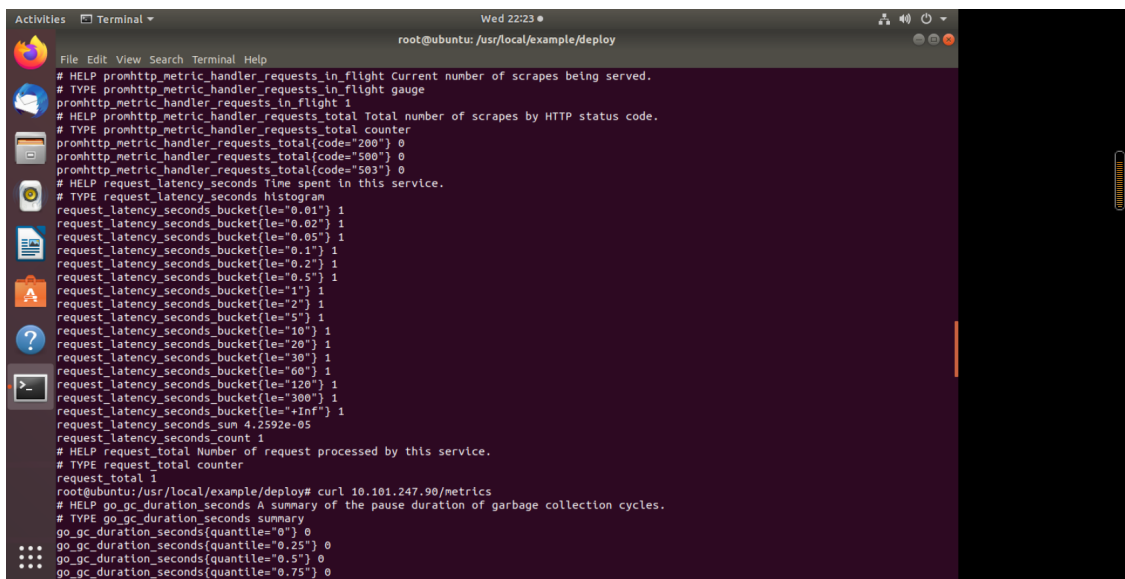
- 拉取metrics成功

```

root@ubuntu:/usr/local/example/metrics_version# kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE   NOMINATED NODE   READINESS GATES
example-service-7874558cf7-5kdrq    1/1     Running   0           3m38s  172.16.0.184    ubuntu <none>      <none>
example-service-7874558cf7-x6cwb    1/1     Running   0           3m40s  172.16.0.183    ubuntu <none>      <none>
root@ubuntu:/usr/local/example/metrics_version# curl 172.16.0.184/metrics
curl: (7) Failed to connect to 172.16.0.184 port 80: Connection refused
root@ubuntu:/usr/local/example/metrics_version# curl 172.16.0.184:5565/metrics
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 0
go_gc_duration_seconds{quantile="0.25"} 0
go_gc_duration_seconds{quantile="0.5"} 0
go_gc_duration_seconds{quantile="0.75"} 0
go_gc_duration_seconds{quantile="1"} 0
go_gc_duration_seconds_sum 0
go_gc_duration_seconds_count 0
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 8
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.12.6"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 411488
# HELP go_memstats_alloc_bytes_total Total number of bytes allocated, even if freed.
# TYPE go_memstats_alloc_bytes_total counter
go_memstats_alloc_bytes_total 411488
# HELP go_memstats_buck_hash_sys_bytes Number of bytes used by the profiling bucket hash table.
# TYPE go_memstats_buck_hash_sys_bytes gauge
go_memstats_buck_hash_sys_bytes 1.442976e+06
# HELP go_memstats_frees_total Total number of frees.
# TYPE go_memstats_frees_total counter

```

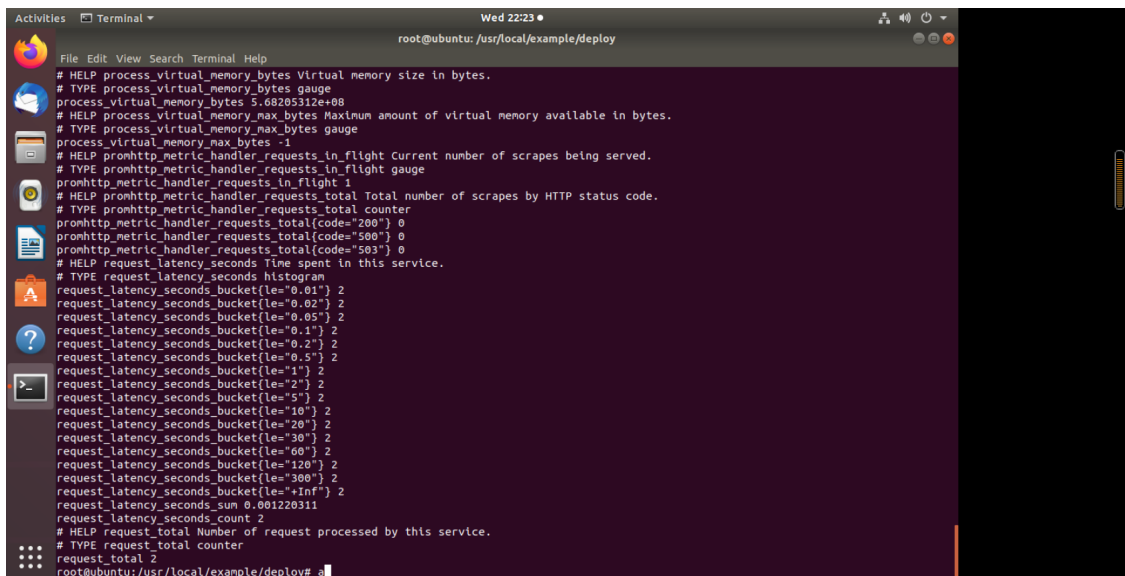
- metrics_version版本编写并发布成功并返回pod的请求次数和时延。



```

root@ubuntu:/usr/local/example/deploy
# HELP promhttp_metric_handler_requests_in_flight Current number of scrapes being served.
# TYPE promhttp_metric_handler_requests_in_flight gauge
promhttp_metric_handler_requests_in_flight 1
# HELP promhttp_metric_handler_requests_total Total number of scrapes by HTTP status code.
# TYPE promhttp_metric_handler_requests_total counter
promhttp_metric_handler_requests_total{code="200"} 0
promhttp_metric_handler_requests_total{code="500"} 0
promhttp_metric_handler_requests_total{code="503"} 0
# HELP request_latency_seconds Time spent in this service.
# TYPE request_latency_seconds histogram
request_latency_seconds_bucket{le="0.01"} 1
request_latency_seconds_bucket{le="0.02"} 1
request_latency_seconds_bucket{le="0.05"} 1
request_latency_seconds_bucket{le="0.1"} 1
request_latency_seconds_bucket{le="0.2"} 1
request_latency_seconds_bucket{le="0.5"} 1
request_latency_seconds_bucket{le="1"} 1
request_latency_seconds_bucket{le="2"} 1
request_latency_seconds_bucket{le="5"} 1
request_latency_seconds_bucket{le="10"} 1
request_latency_seconds_bucket{le="20"} 1
request_latency_seconds_bucket{le="30"} 1
request_latency_seconds_bucket{le="60"} 1
request_latency_seconds_bucket{le="120"} 1
request_latency_seconds_bucket{le="300"} 1
request_latency_seconds_bucket{le="+Inf"} 1
request_latency_seconds_sum 4.2592e-05
request_latency_seconds_count 1
# HELP request_total Number of request processed by this service.
# TYPE request_total counter
request_total 1
root@ubuntu:/usr/local/example/deploy# curl 10.101.247.90/metrics
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 0
go_gc_duration_seconds{quantile="0.25"} 0
go_gc_duration_seconds{quantile="0.5"} 0
go_gc_duration_seconds{quantile="0.75"} 0

```

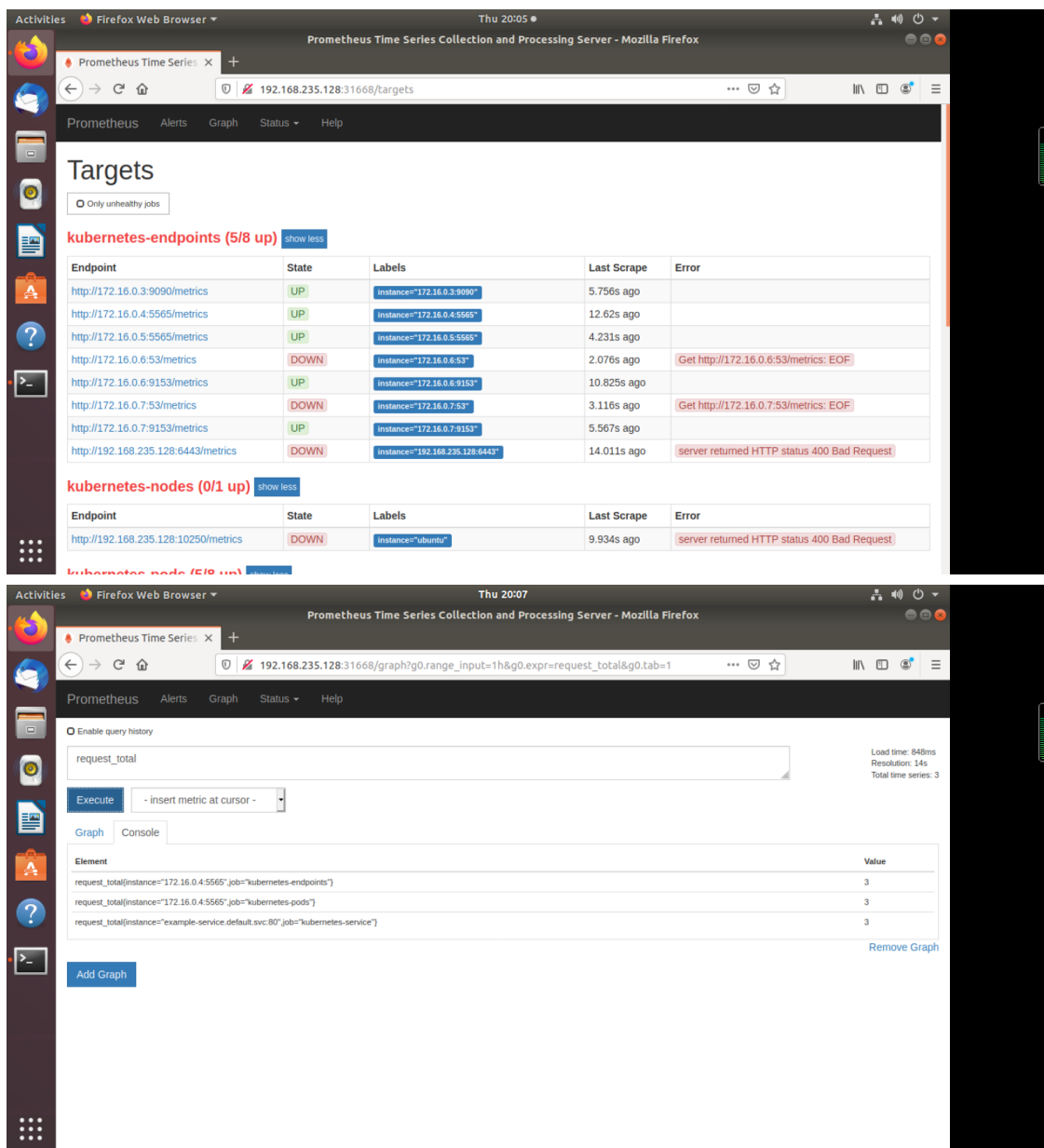


```

root@ubuntu:/usr/local/example/deploy
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 5.68205312e+08
# HELP process_virtual_memory_max_bytes Maximum amount of virtual memory available in bytes.
# TYPE process_virtual_memory_max_bytes gauge
process_virtual_memory_max_bytes -1
# HELP promhttp_metric_handler_requests_in_flight Current number of scrapes being served.
# TYPE promhttp_metric_handler_requests_in_flight gauge
promhttp_metric_handler_requests_in_flight 1
# HELP promhttp_metric_handler_requests_total Total number of scrapes by HTTP status code.
# TYPE promhttp_metric_handler_requests_total counter
promhttp_metric_handler_requests_total{code="200"} 0
promhttp_metric_handler_requests_total{code="500"} 0
promhttp_metric_handler_requests_total{code="503"} 0
# HELP request_latency_seconds Time spent in this service.
# TYPE request_latency_seconds histogram
request_latency_seconds_bucket{le="0.01"} 2
request_latency_seconds_bucket{le="0.02"} 2
request_latency_seconds_bucket{le="0.05"} 2
request_latency_seconds_bucket{le="0.1"} 2
request_latency_seconds_bucket{le="0.2"} 2
request_latency_seconds_bucket{le="0.5"} 2
request_latency_seconds_bucket{le="1"} 2
request_latency_seconds_bucket{le="2"} 2
request_latency_seconds_bucket{le="5"} 2
request_latency_seconds_bucket{le="10"} 2
request_latency_seconds_bucket{le="20"} 2
request_latency_seconds_bucket{le="30"} 2
request_latency_seconds_bucket{le="60"} 2
request_latency_seconds_bucket{le="120"} 2
request_latency_seconds_bucket{le="300"} 2
request_latency_seconds_bucket{le="+Inf"} 2
request_latency_seconds_sum 0.001220311
request_latency_seconds_count 2
# HELP request_total Number of request processed by this service.
# TYPE request_total counter
request_total 2
root@ubuntu:/usr/local/example/deploy#

```

- 运行prometheus



编程题

- 修改业务逻辑

如图，在示例代码基础上，新增定义了两个指标数据，一个是Gauge类型，一个是Counter类型。分别代表了CPU温度和磁盘失败次数统计。

```

cpuTemp = prometheus.NewGauge(
    prometheus.GaugeOpts{
        Name: "cpu_temperature_celsius",
        Help: "Current temperature of the CPU.",
    })
hdFailures = prometheus.NewCounterVec(
    prometheus.CounterOpts{
        Name: "hd_errors_total",
        Help: "Number of hard-disk errors.",
    }, []string{"device"},
)
requestCount = prometheus.NewCounterVec(
    prometheus.CounterOpts{
        Name: "request_total",
        Help: "Number of request processed by this service.",
    }, []string{},
)
requestLatency = prometheus.NewHistogramVec(
    prometheus.HistogramOpts{
        Name: "request_latency_seconds",
        Help: "Time spent in this service.",
        Buckets: []float64{0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1.0, 2.0, 5.0, 10.0, 20.0, 30.0, 60.0, 120.0, 300.0},
    }, []string{},
)

```

为了方便编写和测试，在metrics.go的Register中直接声明与赋予静态数值和设备名给这两个指标。

```
root@ubuntu:/usr/local/example/metrics
File Edit View Search Terminal Help
)
// AdmissionLatency measures latency / execution time of Admission Control execution
// usual usage pattern is: timer := NewAdmissionLatency(); compute; timer.Observe()
type RequestLatency struct {
    histo *prometheus.HistogramVec
    start time.Time
}

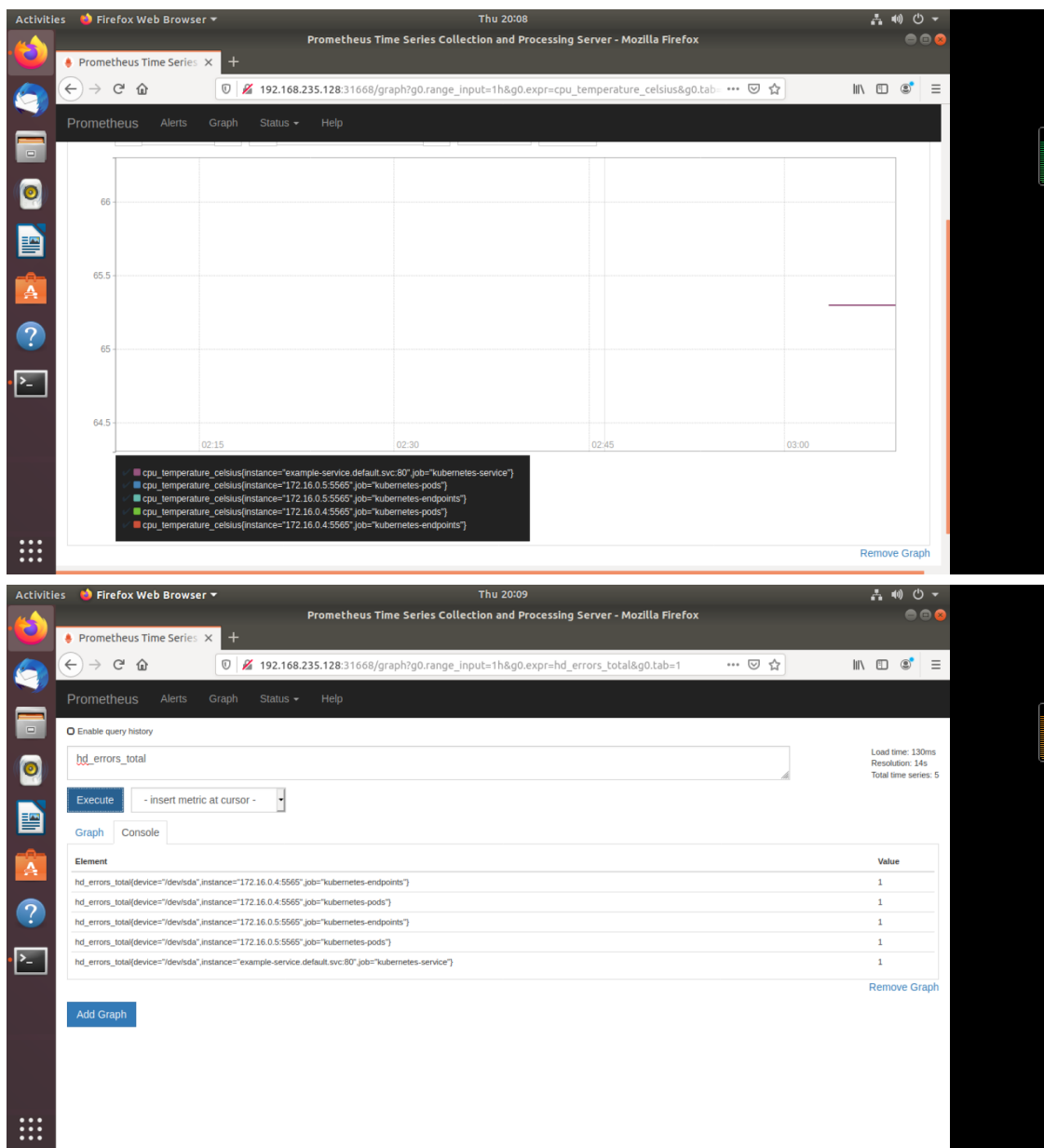
func Register() {
    prometheus.MustRegister(requestCount)
    prometheus.MustRegister(requestLatency)
    prometheus.MustRegister(cpuTemp)
    cpuTemp.Set(65.3)
    prometheus.MustRegister(hdFailures)
    hdFailures.With(prometheus.Labels{"device":"/dev/sda"}).Inc()
}

// NewAdmissionLatency provides a timer for admission latency; call Observe() on it to measure
func NewAdmissionLatency() *RequestLatency {
    return &RequestLatency{
        histo: requestLatency,
        start: time.Now(),
    }
}
}
```

- 结果截图

```
root@ubuntu:/usr/local/example/metrics_version# curl 172.16.0.210/metrics
curl: (7) Failed to connect to 172.16.0.210 port 80: Connection refused
root@ubuntu:/usr/local/example/metrics_version# curl 172.16.0.210:5565/metrics
# HELP cpu_temperature_celsius Current temperature of the CPU.
# TYPE cpu_temperature_celsius gauge
cpu_temperature_celsius 65.3
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 0
go_gc_duration_seconds{quantile="0.25"} 0
go_gc_duration_seconds{quantile="0.5"} 0
go_gc_duration_seconds{quantile="0.75"} 0
go_gc_duration_seconds{quantile="1"} 0
go_gc_duration_seconds_sum 0
go_gc_duration_seconds_count 0
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 7
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.12.6"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 539512
# HELP go_memstats_alloc_bytes_total Total number of bytes allocated, even if freed.
# TYPE go_memstats_alloc_bytes_total counter
go_memstats_alloc_bytes_total 539512
# HELP go_memstats_buck_hash_sys_bytes Number of bytes used by the profiling bucket hash table.
# TYPE go_memstats_buck_hash_sys_bytes gauge
go_memstats_buck_hash_sys_bytes 1
# TYPE go_memstats_sys_bytes gauge
go_memstats_sys_bytes 7.1631096e+07
# HELP go_threads Number of OS threads created.
# TYPE go_threads gauge
go_threads 7
# HELP hd_errors_total Number of hard-disk errors.
# TYPE hd_errors_total counter
hd_errors_total{device="/dev/sda"} 1
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 0.03
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 1.048576e+06
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 7
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 4.374528e+06
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.59250735337e+09
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 4.93883392e+08
# HELP process_virtual_memory_max_bytes Maximum amount of virtual memory available in bytes.
# TYPE process_virtual_memory_max_bytes gauge
process_virtual_memory_max_bytes -1
# HELP promhttp_metric_handler_requests_in_flight Current number of scrapes being served.
# TYPE promhttp_metric_handler_requests_in_flight gauge
promhttp_metric_handler_requests_in_flight 1
# HELP promhttp_metric_handler_requests_total Total number of scrapes by HTTP status code.
# TYPE promhttp_metric_handler_requests_total counter
promhttp_metric_handler_requests_total{code="200"} 0
promhttp_metric_handler_requests_total{code="500"} 0
promhttp_metric_handler_requests_total{code="503"} 0
root@ubuntu:/usr/local/example/metrics_version#
```

- 在prometheus上查看



如上图，两个指标都与设置的一致，且能被prometheus拉取到