

开源软件供应链点亮计划结题报告

项目名称： 基于 Grafana 与 Prometheus 改善 Serverless 计算引擎 rusty-workers 的可观测性

项目编号： 210330562

指导老师： 潘政、周鹤洋

参与学生： 孔可青

目录

一、	项目信息.....	3
1.1	项目名称.....	3
1.2	方案描述.....	3
1.2.1	Rusty-workers	3
1.2.2	Prometheus.....	3
1.2.3	Grafana.....	4
1.3	实施方案.....	4
1.3.1	问题.....	4
1.3.2	设计原则.....	5
1.2.3	详细设计要点.....	5
1.4	项目开发时间计划.....	5
二、	项目情况.....	6
2.1	基础配置.....	6
2.2	功能实现.....	6
2.2.1	跟踪服务器相关信息.....	6
2.2.2	rust 编写的 Prometheus exporter 库	9
2.2.3	将收集的信息进行可视化.....	10
2.3	问题与解决方案.....	12
2.4.1	环境部署.....	12
2.4.2	rust 语言	12
2.4.3	Prometheus 启动问题.....	12
2.4	程序运行情况.....	14
三、	心得体会.....	16

一、 项目信息

1.1 项目名称

基于 Grafana 与 Prometheus 改善 Serverless 计算引擎 rusty-workers 的可观测性

1.2 方案描述

1.2.1 Rusty-workers

类似于编程语言从汇编语言到高级语言的自然进化，云计算也正在进行着裸机→虚拟机→容器→Serverless 计算的演进。Serverless 计算允许开发者在无需了解服务器配置的情况下运行应用程序，开发者将代码上传至平台后，即可以任意规模运行应用，且只需为实际使用的资源付费。

rusty-workers 是一个由 Rust 编写的云原生分布式 Serverless 计算引擎，基于 V8 实现轻量级多应用隔离，并兼容 Cloudflare Workers 的 JS API.

1.2.2 Prometheus

Prometheus 使用 Go 语言开发，与 Google BorgMon 监控系统的实现相似，相比 Heapster 功能更完善、更全面；性能足够支撑上万台规模的集群；是为数不多的适合 Docker、Mesos、Kubernetes 环境的监控系统之一。

(1) 系统架构

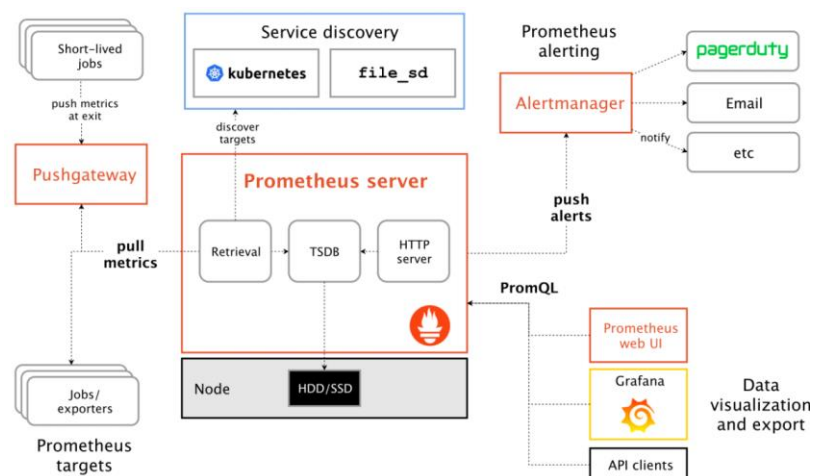


图 1 Prometheus 系统架构图

(2) 工作流程

Prometheus server 定期从配置的 Jobs、exporters 中或其他 Prometheus

server 拉取 metrics，也接收来自 Pushgateway 发过来的 metrics。然后在本地或其他数据库存储收集到的 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。Alertmanager 根据配置文件，对接收到的警报进行处理，发出告警。数据展示组件展示数据或者导出数据。

1.2.3 Grafana

Grafana 是一个由 Go 语言编写的用于可视化大型测量数据的开源系统，功能强大，界面精美，使用它可以创建自定义的控制面板，在面板中配置要显示的数据和显示方式。它 Grafana 支持很多不同的数据源，比如:Graphite、InfluxDB、OpenTSDB、Elasticsearch、Prometheus 等，而且它也支持众多的插件。



图 2 Grafana 的精美可视化展示

虽然 Prometheus 提供的 Web UI 也可以很好的查看不同指标的视图，但是这个功能非常简单，只适合用来调试。要实现一个强大的监控系统，还需要一个能定制展示不同指标的面板，能支持不同类型的展现方式（曲线图、饼状图、热点图、TopN 等），这就是仪表盘（Dashboard）功能。Prometheus 官方推荐使用 Grafana 来对 Prometheus 的指标数据进行可视化，这不仅是因为 Grafana 的功能非常强大，而且它和 Prometheus 可以完美的无缝融合。

1.3 实施方案

1.3.1 问题

目前 rusty-workers 的内部状态对外不可见，给性能优化和错误排查带来了诸多不便。希望基于 Grafana 与 Prometheus 实现 rusty-workers 运行指标的监控和可视化，改善系统的可观测性。

1.3.2 设计原则

尽量降低对 rusty-workers 的侵入，保证低耦合性。贴合 Serverless 场景工作负载特点，保证 rusty-workers 框架的服务质量。

1.2.3 详细设计要点

Prometheus 是使用 Pull 的方式来获取指标数据的，要让 Prometheus 从目标处获得数据，首先必须在目标上安装指标收集的程序，并暴露出 HTTP 接口供 Prometheus 查询。因此要在 rusty-workers 框架形成 HTTP 接口。

1.4 项目开发时间计划

时间	工作内容
2021. 7. 1-2021. 7. 20	搭建工作环境；深入探寻源码，了解工作机制与模块分布
2021. 7. 20-2021. 8. 10	探寻如何获取监控指标的数据
2021. 8. 10-2021. 8. 31	在 rusty-workers 框架形成 HTTP 接口，提供 Prometheus 需要的数据格式
2021. 9. 1-2021. 9. 15	实现可视化展示数据
2021. 9. 15-2021. 9. 30	优化、调整整个项目

二、项目情况

2.1 基础配置

1) 系统环境: Ubuntu 20.04 LTS

```
kkq@kkq-virtual-machine:~$ cat /proc/version
Linux version 5.11.0-36-generic (buildd@lcy01-amd64-004) (gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0, GNU ld (GNU Binutils for Ubuntu) 2.34) #40~20.04.1-Ubuntu SMP Sat Sep 18 02:14:19 UTC 2021
```

2) 数据库 TiDB:

```
kkq@kkq-virtual-machine:~$ tiup list --installed --verbose
Available components:
Name           Owner      Installed   Platforms
----           -
client         pingcap   v1.5.4      darwin/amd64,darwin/arm64,linux/amd64,linux/arm64
grafana        pingcap   v5.1.1,v5.2.1 darwin/amd64,darwin/arm64,linux/amd64,linux/arm64
pd             pingcap   v5.1.1,v5.2.1 darwin/arm64,linux/amd64,linux/arm64,darwin/amd64
playground     pingcap   v1.5.4      darwin/amd64,darwin/arm64,linux/amd64,linux/arm64
prometheus     pingcap   v5.1.1,v5.2.1 darwin/amd64,darwin/arm64,linux/amd64,linux/arm64
tidb           pingcap   v5.1.1,v5.2.1 darwin/amd64,darwin/arm64,linux/amd64,linux/arm64
tidb-lightning pingcap   v5.1.1      darwin/amd64,darwin/arm64,linux/amd64,linux/arm64
tiflash        pingcap   v5.1.1,v5.2.1 darwin/amd64,darwin/arm64,linux/amd64,linux/arm64
tikv           pingcap   v5.1.1,v5.2.1 darwin/amd64,darwin/arm64,linux/amd64,linux/arm64
```

TiDB 是一个分布式系统。最基础的 TiDB 测试集群通常由 2 个 TiDB 实例、3 个 TiKV 实例、3 个 PD 实例和可选的 TiFlash 实例构成。通过 TiUP Playground, 可以快速搭建出一套基础测试集群。

配置文档: <https://www.notion.so/kkqqqqqq/TiDB-8a94de0620854c2b8e5e2d9ddbfeff49e>

3) Prometheus(V 2.28.1) + Grafana

配置文档: <https://www.notion.so/kkqqqqqq/Prometheus-Grafana-ab5e249a748b49cfce84ecd8bbd871c>

4) rusty_workers

配置文档: <https://www.notion.so/kkqqqqqq/rusty-workers-762b445d4441460f83e2529b82a2f35f>

2.2 功能实现

2.2.1 跟踪服务器相关信息

1) app 数量

rusty-workers-proxy 主要负责接收来自 rusty_workers clients 端发来的请求, 并调用 runtime 来完成请求。

这部分中最重要的组件是调度器 SCHEDULER, 负责处理请求, 查询 runtime、发现新的 runtime、将代码加载至内存等。他维护了一个结构体:

```
pub struct Scheduler {
    local_config: LocalConfig,
```

```

worker_config: WorkerConfiguration,
pub clients: AsyncRwLock<BTreeMap<RuntimeId, RtState>>,
pub apps: AsyncMutex<LruCache<AppId, Arc<AppState>>>,
route_cache: AsyncMutex<LruCache<String, BTreeMap<String, AppId>>>,
terminate_queue: tokio::sync::mpsc::Sender<ReadyInstance>,
kv_client: DataClient,
lookup_route_tx: Sender<((String, String), oneshot::Sender<()>>>,
lookup_app_tx: Sender<(AppId, oneshot::Sender<()>>>,
}

```

在此结构体中，成员 `apps` 记录了有关 `app` 的信息，可以通过其长度得知有多少 `app` 已经在 `rusty_worker` 上运行过。

2) `app` 运行时间

关于 `app` 的状态，`rusty_workers` 维护了一个结构体：

```

pub struct AppState {
    id: AppId,
    config: WorkerConfiguration,
    bundle_id: String,
    bundle: Vec<u8>,
    pub ready_instances: AsyncMutex<VecDeque<ReadyInstance>>,
    pub start_time: Instant,
}

```

本项目新增一个成员，`start_time`，用于记录 `app` 第一次加载到内存的时间。每当这个 `app` 的函数被调用时，他都会进行加载到内存的检查，在此处更新时间。

3) `app` 已准备好的实例数量

关于 `app` 的状态，`rusty_workers` 维护了一个结构体：

```

pub struct AppState {
    id: AppId,
    config: WorkerConfiguration,
    bundle_id: String,
    bundle: Vec<u8>,
    pub ready_instances: AsyncMutex<VecDeque<ReadyInstance>>,
    pub start_time: Instant,
}

```

其中成员 `ready_instances` 的长度代表了该 `app` 的实例数量。

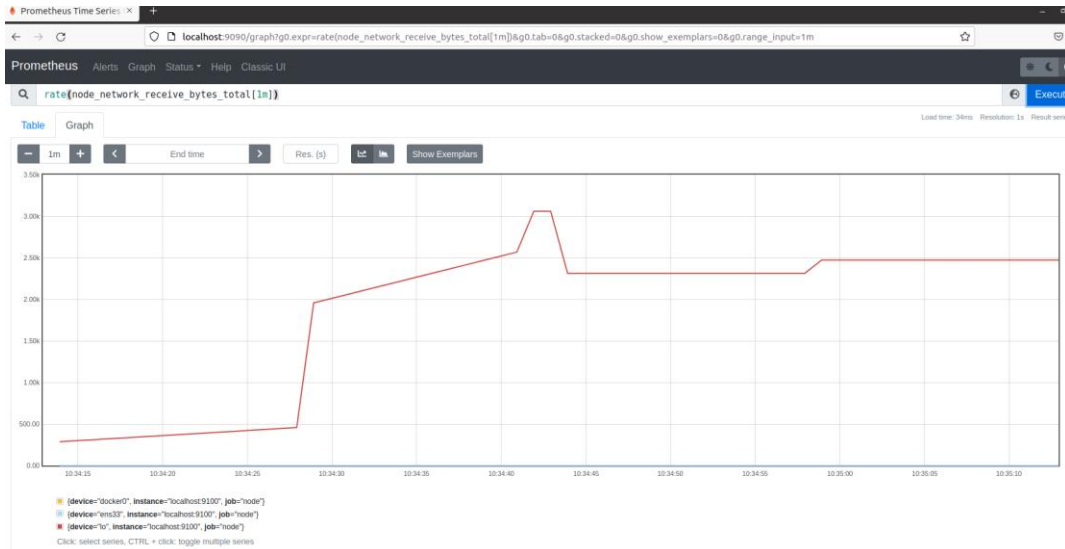
4) `node_exporter` 抓取本机相关数据

`Node_exporter` 是 Prometheus 官方提供的能够公开 Linux 内核硬件和操

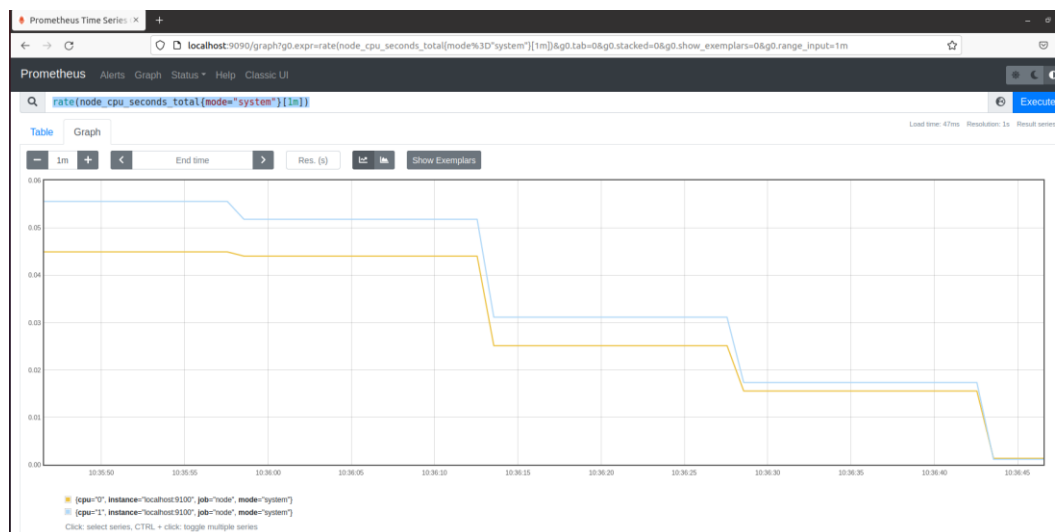
作系统指标的 exporter，用 Go 编写，带有可插入的指标收集器。

例如：以下是 Prometheus 视图（以某信息为例）

过去一分钟内每秒接收的平均网络流量（以字节为单位）



过去一分钟内每秒在系统模式下花费的平均 CPU 时间（以秒为单位）



采用 grafana 进行展示

`node_memory_Inactive_file_bytes{}`



2.2.2 rust 编写的 Prometheus exporter 库

Rusty-worker 项目是 Rust 语言编写的，而 rust 语言简单而言具有以下几点特性：

- 1) 高性能 – Rust 速度惊人且内存利用率极高。由于没有运行时和垃圾回收，它能够胜任对性能要求特别高的服务，可以在嵌入式设备上运行，还能轻松和其他语言集成。
- 2) 可靠性 – Rust 丰富的类型系统和所有权模型保证了内存安全和线程安全，在编译期就能够消除各种各样的错误。
- 3) 生产力 – Rust 拥有出色的文档、友好的编译器和清晰的错误提示信息，还集成了一流的工具：包管理器和构建工具，智能地自动补全和类型检验的多编辑器支持，以及自动格式化代码等等。

官方提供的 exporter 客户端库语言为 Go, JAV, Scala, python, Ruby。因此，采用 rust 语言来写一个 Exporter 库。

1) 大致结构

关键类是 Registry。它主要有三个方法：register、unregister、gather。Register 将一个新的 collector 纳入 metric collection 中；gather 方法，他会返回 EFAULT_REGISTRY 中的 MetricFamily。

大多数用户交互的接口是 Counter、Gauge、Summary 和 Histogram Collectors。这些代表了一个单一的 metric。Metric 代表了监控的系统的一些信息。

Encoder 是一个接口，会将 metric families 编码到底层连线协议。

2) Metrics

➤ counter

单调递增的计数器，它不能允许值减小，但是它可以重置为 0（例如通过服务器重新启动）。

它具有以下方法：

- `inc()` 将 counter 加 1
- `inc(double v)` 将 counter 增加给定的数量。Check $v \geq 0$ 。

Counter 必须从 0 开始。

➤ Gauge

该数值可以上下波动。这个可以适用于一种在某些代码/函数中跟踪正在进行的请求的方法。类似于 python 中的 `track_inprogress`。

具有以下方法：

- `inc()` 增加 1
- `inc(double v)`：按给定的量增加
- `dec()` 将仪表减 1
- `dec(double v)`：按给定的量减少
- `set(double v)`：设置为给给定值

必须从 0 开始，也可以更改代码，让他从其他的数字开始

➤ summary

`summary` 对滑动时间窗口(比如说请求持续时间)进行抽样，并提供对它们的分布、频率和总和等等的数值结果。

➤ Histogram

计算来自事件或可配置桶中的样本流的单个观察值。与 `Summary` 类似，它还提供观察值和观察计数的总和。

2.2.3 将收集的信息进行可视化

Prometheus 通常有两种方式收集数据：

1) Pull 方式，该方式要让 Prometheus 从目标处获得数据，首先必须在目标上安装指标收集的程序，并暴露出 HTTP 接口供 Prometheus 查询。

2) push 方式。Pushgateway 是一种中介服务，允许用户从无法抓取的作业中推送指标。盲目使用 Pushgateway 而不是 Prometheus 通常的 pull 模型进行一般指标收集时，有几个陷阱：

- 当通过单个 Pushgateway 监控多个实例时，Pushgateway 会成为单点故障和潜在瓶颈。
- 失去 Prometheus 的自动实例健康监控。
- Pushgateway 永远不会忘记推送给它的 metrics，并且会永远向 Prometheus 公开它们，除非通过 Pushgateway 的 API 手动删除。

因此，通常 Pushgateway 的唯一有效用例是捕获服务级别批处理作业的结果。所以，我们采用 pull 的方式手机信息，传递信息。

其他语言（例如 Python/Java）实现的 exporter 一般会支持自动公开 metrics 端点，同样的，我们应该：

1. 必须自己公开一个 metrics 端口。
2. 必须调用 gather，encode 并从端点返回字符串

这里应该注意分别的两点概念的是：Prometheus exporter 是一个与监控目标（例如数据库）一起运行的小型二进制文件，用于公开有关该目标的指标。

Prometheus 端口是监控目标本身上的 (HTTP) 端点，可导出有关目标的指标。

因此，他的流程大致如下：

- 1) 创建 metrics
- 2) 注册 metrics
- 3) 通过 encoder 将收集到的指标进行编码
- 4) 指定端口，将指标公布在端口上
- 5) Prometheus 通过该端口的数据进型可视化

需要修改 prometheus 的配置文件 Prometheus.yml，来指定 Prometheus 监控哪几个端口的数据。

```
global:
  scrape_interval: 2s
  evaluation_interval: 2s
scrape_configs:
  - job_name: 'rusty_workers'
    static_configs:
      - targets: ['localhost:9898']
  - job_name: 'node'
```

```
static_configs:  
  - targets: ['localhost:9100']  
  
- job_name: 'tikv'  
  static_configs:  
    - targets: ['localhost:9090']
```

2.3 问题与解决方案

2.4.1 环境部署

1) 包下载

问题：在搭载环境的时候，由于网络的问题，部分依赖包下载失败。尝试采用手动下载该包并放入目录中会编译失败。

解决方案：将代理调整至更新 SSR 服务器订阅（不通过代理）模式。

2) 虚拟机内存

问题：虚拟机初始只分配 20G 的内存，内存紧张，以致无法启动数据库。

解决方案：虚拟机添加 20G 的硬盘并重新分区。

参考链接：<https://blog.csdn.net/konroy/article/details/79832448>

2.4.2 rust 语言

Rust 语言难以上手，学习成本较高。其中类型的转换、引用借用需要花时间理解。

2.4.3 Prometheus 启动问题

启动 promethues 时报错 9090 端口被占用：

```

kkq@kkq-virtual-machine:~/prometheus-2.28.1.linux-amd64$ ./prometheus --config
.file=./prometheus.yml
level=info ts=2021-09-26T10:55:28.757Z caller=main.go:389 msg="No time or size
retention was set so using the default time retention" duration=15d
level=info ts=2021-09-26T10:55:28.757Z caller=main.go:443 msg="Starting Promet
heus" version="(version=2.28.1, branch=HEAD, revision=b0944590a1c9a6b35dc5a696
869f75f422b107a1)"
level=info ts=2021-09-26T10:55:28.757Z caller=main.go:448 build_context="(go=g
o1.16.5, user=root@2915dd495090, date=20210701-15:20:10)"
level=info ts=2021-09-26T10:55:28.757Z caller=main.go:449 host_details="(Linux
5.11.0-36-generic #40~20.04.1-Ubuntu SMP Sat Sep 18 02:14:19 UTC 2021 x86_64
kkq-virtual-machine (none))"
level=info ts=2021-09-26T10:55:28.757Z caller=main.go:450 fd_limits="(soft=102
4, hard=1048576)"
level=info ts=2021-09-26T10:55:28.757Z caller=main.go:451 vm_limits="(soft=unl
imited, hard=unlimited)"
level=info ts=2021-09-26T10:55:28.759Z caller=web.go:541 component=web msg="St
art listening for connections" address=0.0.0.0:9090
level=error ts=2021-09-26T10:55:28.759Z caller=main.go:653 msg="Unable to star
t web listener" err="listen tcp 0.0.0.0:9090: bind: address already in use"

```

解决方案：使用 `netstat -antup` 命令查看本机端口是否处于被占用状态

```

kkq@kkq-virtual-machine:~/prometheus-2.28.1.linux-amd64$ netstat -antup
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:3200         0.0.0.0:*                LISTEN      31040/./target/rele
tcp        0      0 127.0.0.1:4000         0.0.0.0:*                LISTEN      3512/tidb-server
tcp        0      0 127.0.0.1:3201         0.0.0.0:*                LISTEN      31041/./target/rele
tcp        0      0 127.0.0.1:9090         0.0.0.0:*                LISTEN      3482/prometheus
tcp        0      0 0.0.0.0:20292          0.0.0.0:*                LISTEN      3643/tiflash
tcp        0      0 0.0.0.0:9000           0.0.0.0:*                LISTEN      3643/tiflash
tcp        0      0 127.0.0.1:9898         0.0.0.0:*                LISTEN      31042/./target/rele
tcp        0      0 0.0.0.0:8234           0.0.0.0:*                LISTEN      3643/tiflash
tcp        0      0 127.0.0.1:2379         0.0.0.0:*                LISTEN      3502/pd-server
tcp        0      0 127.0.0.1:2380         0.0.0.0:*                LISTEN      3502/pd-server
tcp        0      0 0.0.0.0:3280           0.0.0.0:*                LISTEN      31042/./target/rele
tcp        0      0 127.0.0.1:38289        0.0.0.0:*                LISTEN      3502/pd-server
tcp        0      0 127.0.0.1:20180        0.0.0.0:*                LISTEN      3510/tikv-server
tcp        0      0 127.0.0.53:53          0.0.0.0:*                LISTEN      -
tcp        0      0 127.0.0.1:631          0.0.0.0:*                LISTEN      -
tcp        0      0 127.0.0.1:3000         0.0.0.0:*                LISTEN      3493/grafana-server
tcp        0      0 127.0.0.1:44953        0.0.0.0:*                LISTEN      3502/pd-server
tcp        0      0 0.0.0.0:8123           0.0.0.0:*                LISTEN      3643/tiflash
tcp        0      0 127.0.0.1:48168        127.0.0.1:4000         TIME_WAIT   -

```

kill 此进程即可。重新启动 prometheus，正确启动时应得到如下输出：

```

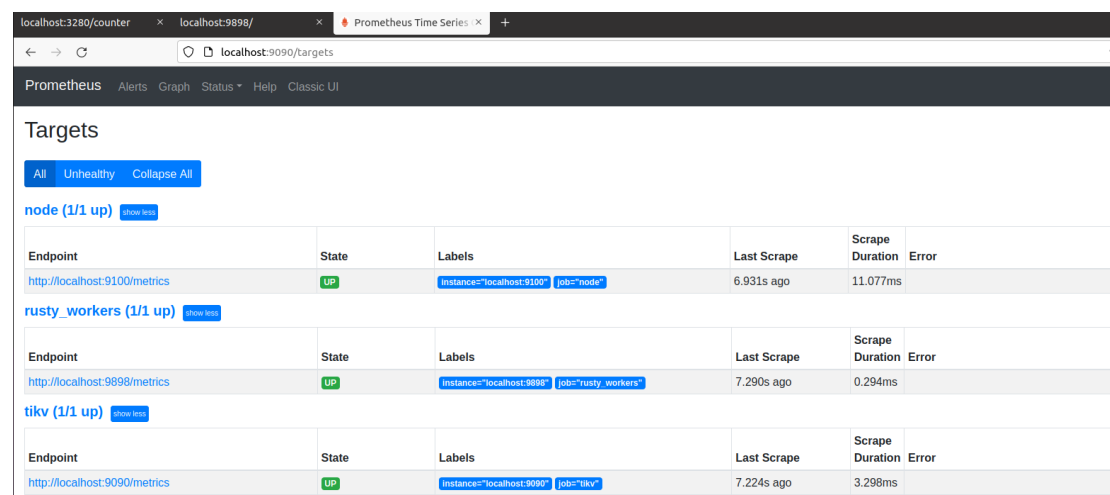
kkq@kkq-virtual-machine:~/prometheus-2.28.1.linux-amd64$ ./prometheus --config.file=./prometheus.yml
level=info ts=2021-09-26T11:25:35.508Z caller=main.go:389 msg="No time or size retention was set so using the default time retention" duration=15d
level=info ts=2021-09-26T11:25:35.508Z caller=main.go:443 msg="Starting Prometheus" version="(version=2.28.1, branch=HEAD, revision=b0944590a1c9a6b3
5dc5a696869f75f422b107a1)"
level=info ts=2021-09-26T11:25:35.508Z caller=main.go:448 build_context="(go=go1.16.5, user=root@2915dd495090, date=20210701-15:20:10)"
level=info ts=2021-09-26T11:25:35.508Z caller=main.go:449 host_details="(Linux 5.11.0-36-generic #40~20.04.1-Ubuntu SMP Sat Sep 18 02:14:19 UTC 2021
x86_64 kkq-virtual-machine (none))"
level=info ts=2021-09-26T11:25:35.508Z caller=main.go:450 fd_limits="(soft=1024, hard=1048576)"
level=info ts=2021-09-26T11:25:35.509Z caller=main.go:451 vm_limits="(soft=unlimited, hard=unlimited)"
level=info ts=2021-09-26T11:25:35.510Z caller=web.go:541 component=web msg="Start listening for connections" address=0.0.0.0:9090
level=info ts=2021-09-26T11:25:35.511Z caller=main.go:824 msg="Starting TSDB ..."
level=info ts=2021-09-26T11:25:35.513Z caller=tsdb/config.go:191 component=tsdb msg="TLS is disabled." http2=false
level=info ts=2021-09-26T11:25:35.517Z caller=repair.go:57 component=tsdb msg="Found healthy block" mint=1628151391833 maxt=1628157600000 ulid=01FCF
VHMHKK6MC4Q3FCVX05BV
level=info ts=2021-09-26T11:25:35.528Z caller=head.go:780 component=tsdb msg="Replaying on-disk memory mappable chunks if any"
level=info ts=2021-09-26T11:25:35.528Z caller=head.go:794 component=tsdb msg="On-disk memory mappable chunks replay completed" duration=3.656µs
level=info ts=2021-09-26T11:25:35.528Z caller=head.go:800 component=tsdb msg="Replaying WAL, this may take a while"
level=info ts=2021-09-26T11:25:35.534Z caller=head.go:854 component=tsdb msg="WAL segment loaded" segment=0 maxSegment=3
level=info ts=2021-09-26T11:25:35.544Z caller=head.go:854 component=tsdb msg="WAL segment loaded" segment=1 maxSegment=3
level=info ts=2021-09-26T11:25:35.561Z caller=head.go:854 component=tsdb msg="WAL segment loaded" segment=2 maxSegment=3
level=info ts=2021-09-26T11:25:35.561Z caller=head.go:854 component=tsdb msg="WAL segment loaded" segment=3 maxSegment=3
level=info ts=2021-09-26T11:25:35.561Z caller=head.go:860 component=tsdb msg="WAL replay completed" checkpoint_replay_duration=122.541µs wal_replay_
duration=33.042503ms total_replay_duration=33.18421ms
level=info ts=2021-09-26T11:25:35.563Z caller=main.go:851 fs_type=EXT4_SUPER_MAGIC
level=info ts=2021-09-26T11:25:35.563Z caller=main.go:854 msg="TSDB started"
level=info ts=2021-09-26T11:25:35.563Z caller=main.go:981 msg="Loading configuration file" filename=./prometheus.yml
level=info ts=2021-09-26T11:25:35.564Z caller=main.go:1012 msg="Completed loading of configuration file" filename=./prometheus.yml totalDuration=792
.678µs remote_storage=1.287µs web_handler=389ns query_engine=34.857µs scrape=368.429µs scrape_sd=22.82µs notify=590ns notify_sd=1.11µs rules=907ns
level=info ts=2021-09-26T11:25:35.564Z caller=main.go:796 msg="Server is ready to receive web requests."
level=info ts=2021-09-26T11:25:41.381Z caller=compact.go:509 component=tsdb msg="write block resulted in empty block" mint=1628157600000 maxt=162816
4800000 duration=9.675792ms
level=info ts=2021-09-26T11:25:41.382Z caller=head.go:967 component=tsdb msg="Head GC completed" duration=907.556µs
level=info ts=2021-09-26T11:25:41.416Z caller=compact.go:518 component=tsdb msg="write block" mint=162856838051 maxt=1628568000000 ulid=01FCGXN146H
52PF3XRF8XCQHQ2 duration=34.485355ms
level=info ts=2021-09-26T11:25:41.418Z caller=head.go:967 component=tsdb msg="Head GC completed" duration=1.367706ms
level=info ts=2021-09-26T11:25:41.418Z caller=checkpoint.go:97 component=tsdb msg="Creating checkpoint" from_segment=0 to_segment=1 mint=16285680000
00
level=info ts=2021-09-26T11:25:41.427Z caller=head.go:1064 component=tsdb msg="WAL checkpoint complete" first=0 last=1 duration=8.518636ms

```

2.4 程序运行情况

程序运行时，首先要保证 `rusty_workers`、`node_exporter` 正确运行。并且修改好 Prometheus 的配置文件。

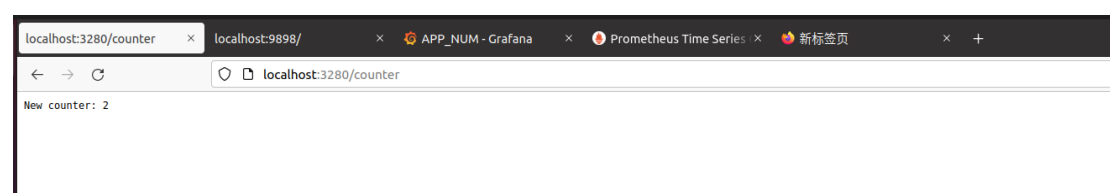
配置正确时，在 `localhost:9090` 端口的 `target` 选项卡中应该看到以下情况：



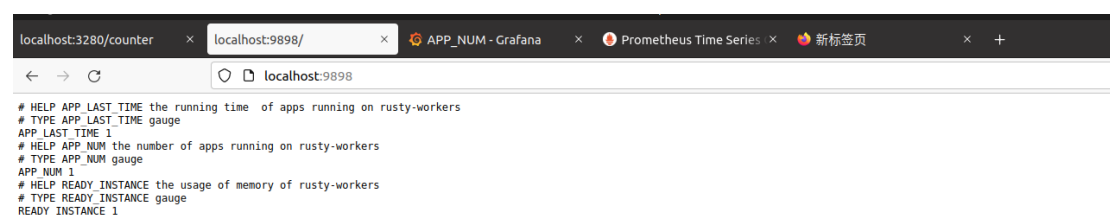
The screenshot shows the Prometheus Targets page with three targets listed. Each target has a table with columns: Endpoint, State, Labels, Last Scrape, Scrape Duration, and Error.

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
node (1/1 up)					
http://localhost:9100/metrics	UP	instance="localhost:9100" job="node"	6.931s ago	11.077ms	
rusty_workers (1/1 up)					
http://localhost:9898/metrics	UP	instance="localhost:9898" job="rusty_workers"	7.290s ago	0.294ms	
tikv (1/1 up)					
http://localhost:9090/metrics	UP	instance="localhost:9090" job="tikv"	7.224s ago	3.298ms	

触发 样例 app 时的网页正确显示：



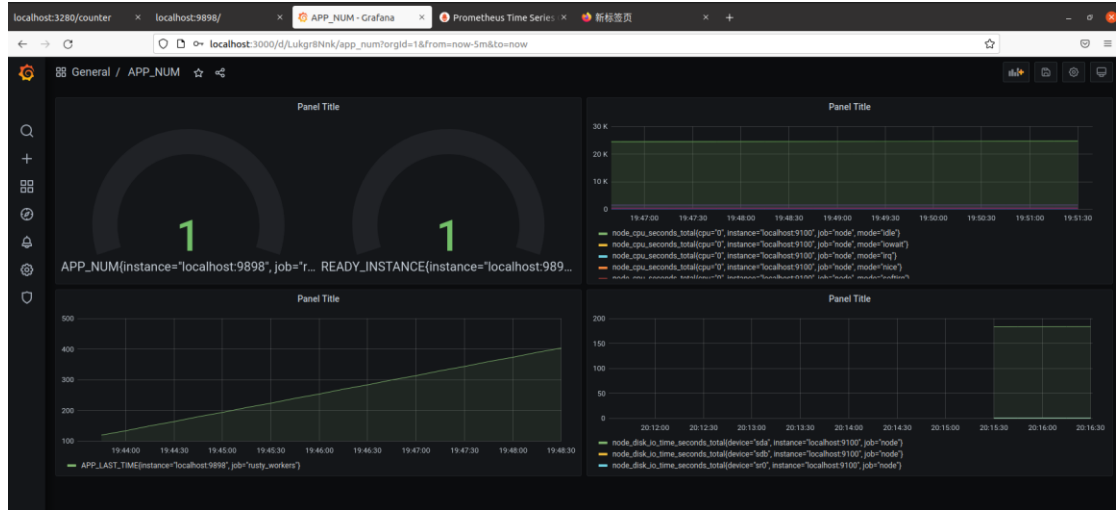
通过 `localhost:9898` 端口暴露的跟踪指标信息：



The screenshot shows a web browser window with the URL `localhost:9898`. The page content displays the following Prometheus metrics:

```
# HELP APP_LAST_TIME the running time of apps running on rusty-workers
# TYPE APP_LAST_TIME gauge
APP_LAST_TIME 1
# HELP APP_NUM the number of apps running on rusty-workers
# TYPE APP_NUM gauge
APP_NUM 1
# HELP READY_INSTANCE the usage of memory of rusty-workers
# TYPE READY_INSTANCE gauge
READY_INSTANCE 1
```

采用 Grafana 进行展示：左上侧两个仪表盘分别是 app 数量，该 app 的 `ready_instance` 数量。左下侧是 app 运行时间记录。右侧分别是主机 cpu 使用情况与网络 IO 情况。



三. 心得体会

此次项目到这里就要结束了。在此期间收获满满呀。

中期检查前的工作主要围绕着环境部署，文档完善，系统调研而展开。后一阶段，则展开了切实的代码，调试工作。

六月选择项目时，此项目吸引我的主要有两点：一个是此项目是关于 Serverless 的计算引擎，与我的研究方向比较贴近；另一个是此项目是 rust 编写的。rust 的安全性、高度并发很是吸引人。抱着这样的想法，我选择了这个项目。

主要收获有以下几点：首先是参阅官方文档的能力。以前大多是在网上参考他人写的博客，但是这个可能不适用于自己的环境并且具有滞后性。而很多软件和服务其实官方都提供了很详细的教程和文档，足以解决使用过程中的大部分问题。官方文档比转述或者翻译的更具有时效性，并且会随着产品的升级会对文档进行及时的修改。因此学会查看官方文档很重要。

其次，学会自己检查错误。当遇到项目报错时，有时候报错提示会给出可能的错误来源以及可能的解决方案，可能来的比网上资料更加直接。但是也要学会看看其他人的错误的和解决方法，有助于解决自己的问题。

然后，一个大型的项目会用到很多依赖，如何有效的管理项目的依赖是一个很重要的问题。

另外则是如何将一个大型项目划分为不同的模块，各模块之间有效协作。例如我起初试图将有关 Prometheus 相关部分的内容单独写作一个进程以降低模块的耦合性，但不同进程之间相互通信又会带来复杂性，因此，还是选择将其与 proxy 集成。

最后，感谢周鹤洋导师，在我进行设计，debug 时给予我及时有效的帮助，让我少走了许多弯路~ 😊