链滴

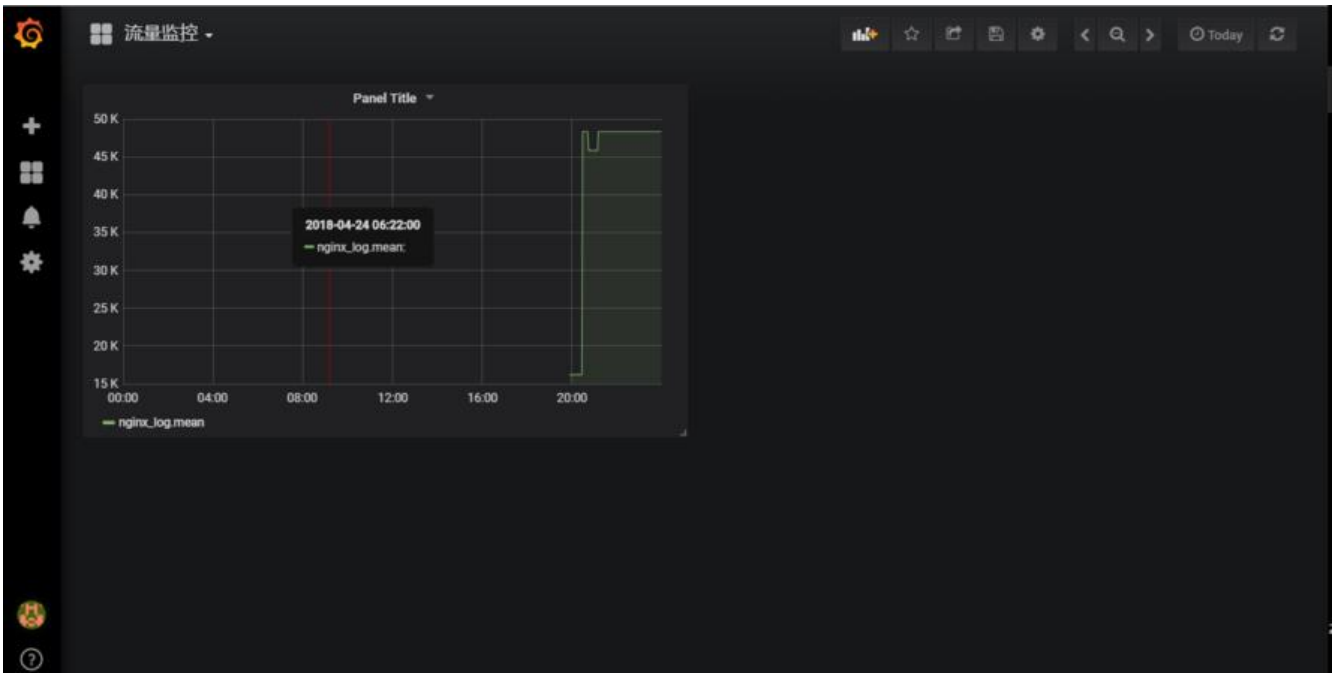
golang+influxdb+grafana 实现 nginx 日志流量、响应时间等监控系统

作者：xhaoxiong

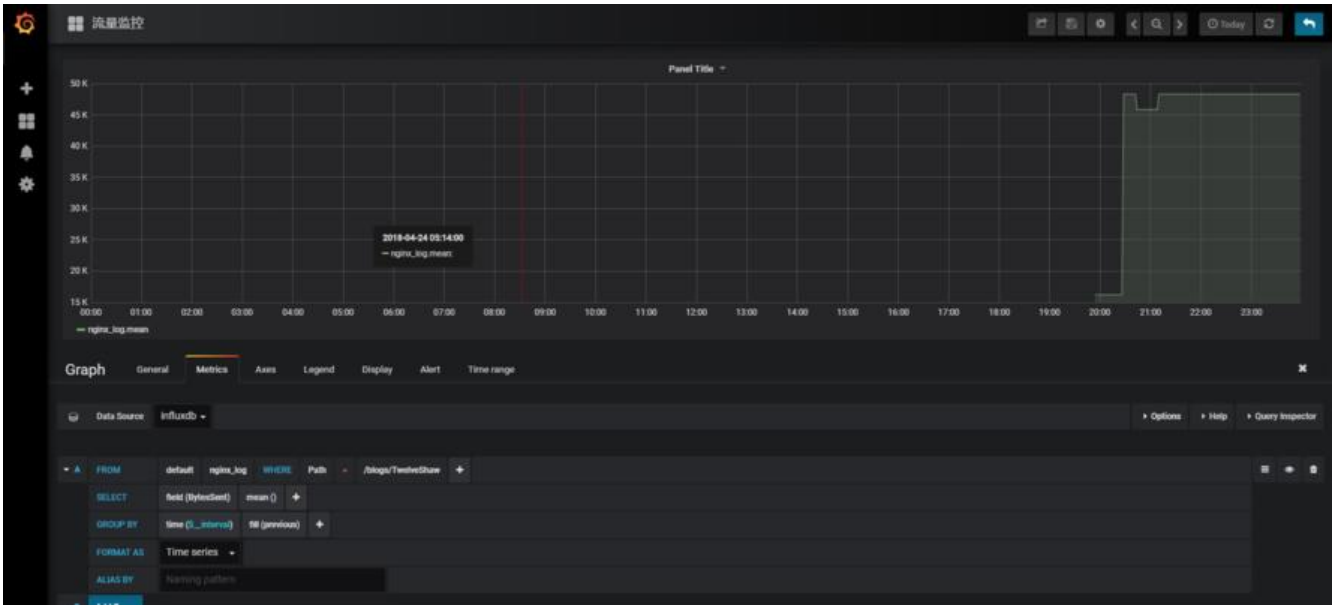
原文链接：https://ld246.com/article/1524579561076

来源网站：链滴

先看下具体效果图



主要是对自己博客服务器的流量的监控



主要思路

读取日志->正则解析->写入influxdb->grafana获取数据渲染

首先我们需要先将influxdb 和grafana 安装部署好 以便于后面使用

centos7安装influxdb

wget https://dl.influxdata.com/influxdb/releases/influxdb-0.13.0.x86_64.rpm

sudo yum localinstall influxdb-0.13.0.x86_64.rpm

service influxdb restart

如果遇到influxdb 8083端口访问不到web管理页面则需要到/etc/influxdb/influxdb.conf 修改参数

vim 命令模式下：/admin搜索到对应部分去掉注释#[admin] #enabled=true

```
 [admin]
  # Determines whether the admin service is enabled.
   enabled =true

  # The default bind address used by the admin service.
   bind-address = ":8083"

  # Whether the admin service should use HTTPS.
  # https-enabled = false

  # The SSL certificate used when HTTPS is enabled.
  # https-certificate = "/etc/ssl/influxdb.pem"
```

然后具体操作添加用户，建库什么均在百度，和mysql大同小异

centos7下安装grafana

在官方文档中有很多办法

我是通过配置yum 然后install的

Add the following to a new file at /etc/yum.repos.d/grafana.repo

```
[grafana]
name=grafana
baseurl=https://packagecloud.io/grafana/stable/el/7/$basearch
repo_gpgcheck=1
enabled=1
gpgcheck=1
gpgkey=https://packagecloud.io/gpg.key https://grafanarel.s3.amazonaws.com/RPM-GPG-K
Y-grafana
sslverify=1
sslcacert=/etc/pki/tls/certs/ca-bundle.crt
```

```
sudo yum install grafana

service grafana-server  restart
```

默认跑在3000端口
账号密码为
admin
admin

代码实现

```
type Reader interface {
    Read(rc chan []byte)
}
```

```
type Writer interface {
    Write(rc chan *Message)
}

//存入db的基本数据
type Message struct {
    Host string

    TimeLocal                 time.Time
    Method, Resource, Protocol string
    Status                    string
    BytesSent                 int
    Scheme                    string
    Url                       string
    UpstreamTime, RequestTime  float64
}

//扩展性实现接口的struct
type ReadFromFile struct {
    path string
}

type WriteToInfluxDB struct {
    influxDBDsn string
}

//日志解析中转
type LogProcess struct {
    rc    chan []byte
    wc    chan *Message
    read  Reader
    write Writer
}
```

从文件中读取数据（注释中的数据为我nginx日志中的一条数据）

```
/*14.215.176.15 - - [23/APR/2018:21:43:39 +0800]
"GET /CONSOLE/DIST/LAYOUTS/DEFAULT.F5E6C608DE637CAC3F50.JS HTTP/1.1"
200 5665 "HTTPS://WWW.XHXBLOG.CN/?B3ID=H9OXZSYM"
"MOZILLA/5.0 (WINDOWS NT 6.1; WOW64; RV:43.0) GECKO/20100101 FIREFOX/43.0" "-"*/

func (r *ReadFromFile) Read(rc chan []byte) {
    //读取数据

    f, err := os.Open(r.path)
    if err != nil {
        panic(fmt.Sprintf("open file fail:%s", err.Error()))
    }

    //跳到文件末尾
    f.Seek(0, 2)
```

```
    rd := bufio.NewReader(f)
    //循环读取数据将每行数据送入rc channel
    for {

        line, err := rd.ReadBytes('\n')
        if err == io.EOF {

            time.Sleep(500 * time.Millisecond)
            continue
        } else if err != nil {
            panic(fmt.Sprintf("ReadBytes error: %s", err.Error()))
        }
        TypeMonitorChan <- TypeHandleLine
        rc <- line[:len(line)-1]

    }

}
```

正则解析数据(参考正则表达解析,使用该工具调试)

正则测试模块没有单独写出来了，放在如下函数的注释中测试

其中method,resource,protocol等是通过正则解析后的字符串再次通过空格split分割获取的

```
func (l *LogProcess) Process() {
    //解析数据

    r := regexp.MustCompile(`([\d\.]+)\s+([^\[]+)\s+([^\[]+)\s+\[([^\]]+)\]\s+\"([^"]+)\"\s+(\d{3
)\s+(\d+)\s+\"([^"]+)\"\s+\"([^"]+)\"\s+`)

    /**
      测试flag
    */
    //flag := 0
    loc, _ := time.LoadLocation("Asia/Shanghai")
    for v := range l.rc {

        ret := r.FindStringSubmatch(string(v))

        /*测试**/
        /*
        if flag != 2 {
            sp := strings.Split(ret[5], " ")

            fmt.Println(ret)
            fmt.Println("Host:", ret[1])

            fmt.Println("LocalTime:", ret[4])

            fmt.Println("Method:", sp[0])
            uu, _ := url.Parse(sp[1])
            fmt.Println(uu.Path)
```

```
    fmt.Println("Path:", sp[1])
    fmt.Println("Protocol:", sp[2])
    fmt.Println("Status:", ret[6])
    fmt.Println("BytesSent:", ret[7])
    fmt.Println("Scheme:", ret[9])
    fmt.Println(ret[8])
    flag++
}
*/
if len(ret) != 10 {
    TypeMonitorChan <- TypeErrNum
    log.Println("FindStringSubmatch fail:", string(v))
    continue
}

t, err := time.ParseInLocation("02/Jan/2006:15:04:05 +0800", ret[4], loc)
if err != nil {
    log.Println("ParseInLocation fail:", err.Error(), ret[4])
}
message := &Message{}

//14.215.176.15
message.Host = ret[1]

//23/APR/2018:21:43:39 +0800
message.TimeLocal = t
//MOZILLA/5.0 (WINDOWS NT 6.1; WOW64; RV:43.0) GECKO/20100101 FIREFOX/43.0
message.Scheme = ret[9]

//GET /CONSOLE/DIST/LAYOUTS/DEFAULT.F5E6C608DE637CAC3F50.JS HTTP/1.1
sp := strings.Split(ret[5], " ")

if len(sp) != 3 {
    TypeMonitorChan <- TypeErrNum
    log.Println("strings.Split fail:", ret[5])
    continue
}
//请求方法
message.Method = sp[0]

//请求路径
u, err := url.Parse(sp[1])

if err != nil {
    TypeMonitorChan <- TypeErrNum
    log.Println("url parse fail:", err)
    continue
}

message.Resource = u.Path

//请求协议
message.Protocol = sp[2]
//200
```

```
        message.Status = ret[7]

        //HTTPS://WWW.XHXBLOG.CN/?B3ID=H9OXZSYM
        message.Url = ret[8]
        //5665
        message.BytesSent, _ = strconv.Atoi(ret[7])
        l.wc <- message
    }

}
```

写入influxdb客户端是用golang写的influxdb gay地址为：InfluxDB Client

```
func (w *WriteToInfluxDB) Write(wc chan *Message) {

    sp := strings.Split(w.influxDBDsn, "@")

    // Create a new HTTPClient
    c, err := client.NewHTTPClient(client.HTTPConfig{
        Addr:     sp[0],
        Username: sp[1],
        Password: sp[2],
    })
    if err != nil {
        log.Fatal(err)
    }
    defer c.Close()

    for v := range wc {
        // Create a new point batch
        bp, err := client.NewBatchPoints(client.BatchPointsConfig{
            Database:  sp[3],
            Precision: sp[4],
        })
        if err != nil {
            log.Fatal(err)
        }

        // Create a point and add to batch
        tags := map[string]string{"Path": v.Resource, "Method": v.Method, "Scheme": v.Scheme, "
tatus": v.Status, "Protocol": v.Protocol}
        fields := map[string]interface{}{
            "RequestTime": 2.0,
            "BytesSent":   v.BytesSent,
        }

        pt, err := client.NewPoint("nginx_log", tags, fields, v.TimeLocal)
        if err != nil {
            log.Fatal(err)
        }
        bp.AddPoint(pt)
```

```
        // Write the batch
        if err := c.Write(bp); err != nil {
            log.Fatal(err)
        }

        // Close client resources
        if err := c.Close(); err != nil {
            log.Fatal(err)
        }

        log.Println("write success")
    }

}



type SystemInfo struct {
    HandleLine   int     `json:"handleLine"`
    Tps          float64 `json:"tps"`
    ReadChanLen  int     `json:"readChanLen"`
    WriteChanLen int     `json:"writeChanLen"`
    RunTime      string  `json:"runTime"`
    ErrNum       int     `json:"errNum"`
}

type Monitor struct {
    startTime time.Time
    data      SystemInfo
    tpsSli    []int
}
```

此处主要监控运行时间，channel阻塞数量，处理条数监听一个端口在8999上

其次该监听将阻塞在main函数上

```
//专门接收内容的channel，错误数量和处理条数
var TypeMonitorChan = make(chan int, 200)

func (m *Monitor) start(lp *LogProcess) {
    go func() {
        for n := range TypeMonitorChan {
            switch n {
            case TypeErrNum:
                m.data.ErrNum += 1
            case TypeHandleLine:
                m.data.HandleLine += 1
            }
        }
    }()

    ticker := time.NewTicker(time.Second * 5)

    go func() {
```

```
        <-ticker.C
        m.tpsSli = append(m.tpsSli, m.data.HandleLine)
        //目的是为了通过两次读取的行数除以单位时间就能得到大概的吞吐量
        if len(m.tpsSli) > 2 {
            m.tpsSli = m.tpsSli[1:]
        }
    }()

    http.HandleFunc("/monitor", func(writer http.ResponseWriter, request *http.Request) {
        m.data.RunTime = time.Now().Sub(m.startTime).String()
        m.data.ReadChanLen = len(lp.rc)
        m.data.WriteChanLen = len(lp.wc)

        if len(m.tpsSli) >= 2 {
            m.data.Tps = float64(m.tpsSli[1]-m.tpsSli[0]) / 5
        }
        ret, _ := json.MarshalIndent(m.data, "", "\t")

        io.WriteString(writer, string(ret))
    })

    http.ListenAndServe(":8999", nil)

}

func main() {

    //通过命令行模式输入参数简单化，不输则为默认值
    var path, influxDsn string
    flag.StringVar(&path, "path", "/var/log/nginx/access.log", "read file path")
    flag.StringVar(&influxDsn, "influxDsn", "http://127.0.0.1:8086@haoxiong@8080@nginx_lo
@s", "influx data source")
    flag.Parse()
    r := &ReadFromFile{
        path: path,
    }

    w := &WriteToInfluxDB{
        influxDBDsn: influxDsn,
    }

    lp := &LogProcess{
        rc:    make(chan []byte, 200),
        wc:    make(chan *Message),
        read:  r,
        write: w,
    }

    go lp.read.Read(lp.rc)
    for i := 0; i < 2; i++ {
        go lp.Process()

    }
    for i := 0; i < 4; i++ {
```

```
        go lp.write.Write(lp.wc)
    }

    m := &Monitor{
        startTime: time.Now(),
        data:      SystemInfo{},
    }
    //监听一个端口阻塞main
    m.start(lp)
}
```

具体代码