为什么要做NGINX日志分析看板

Grafana官网的dashboards有NGINX日志采集到ES数据源的展示看板，也有采集到LOKI数据源的展示看板，唯独没有采集到ClickHouse数据源的展示看板。所以这个轮子是必须要造的。

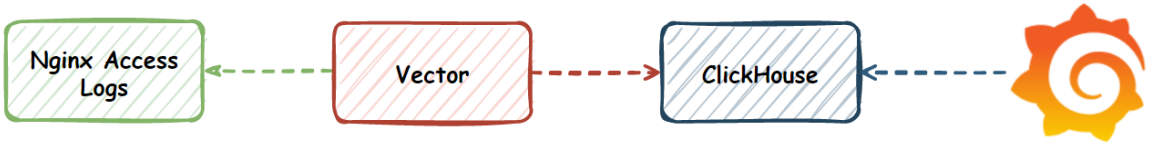
为什么不使用ES存储

ElasticSearch是全文检索引擎的文档数据库，对于业务日志、异常日志、多行日志这类，非结构化、半结构化的日志数据，经常需要做关键字查询，模糊匹配等操作，非常适合使用es，使用倒排索引实现快速全文搜索。

ClickHouse是一个列式存储数据库，尤其擅长处理结构化的大规模的SQL查询和聚合分析操作，所以针对NGINX这类结构化的请求日志，在处理多维分析、聚合查询、分组统计等操作速度极快，并且压缩比极高，存储成本比ES低10倍，CPU、内存的占用也有巨大优势。

NGINX日志采集架构

- 基础架构



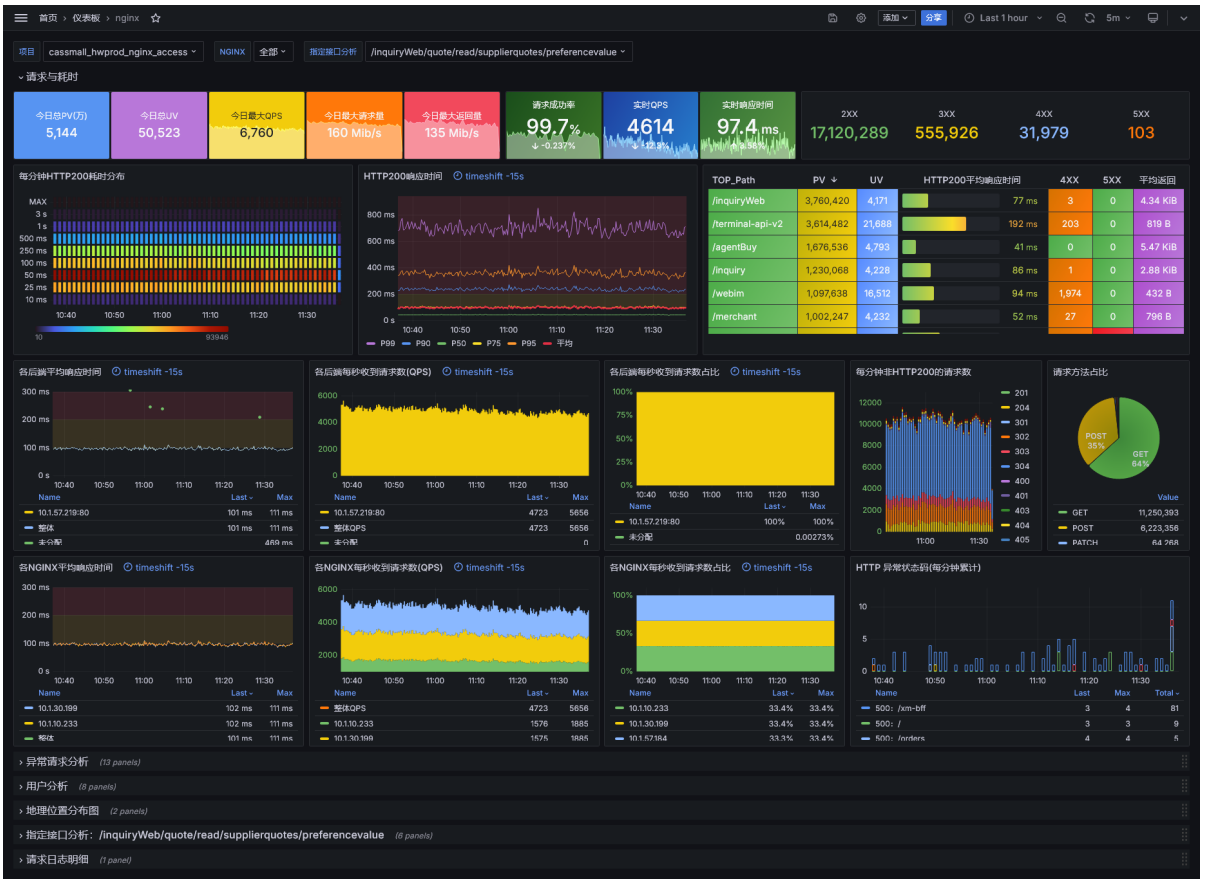
- 完整架构

Grafana请求日志分析看板预览

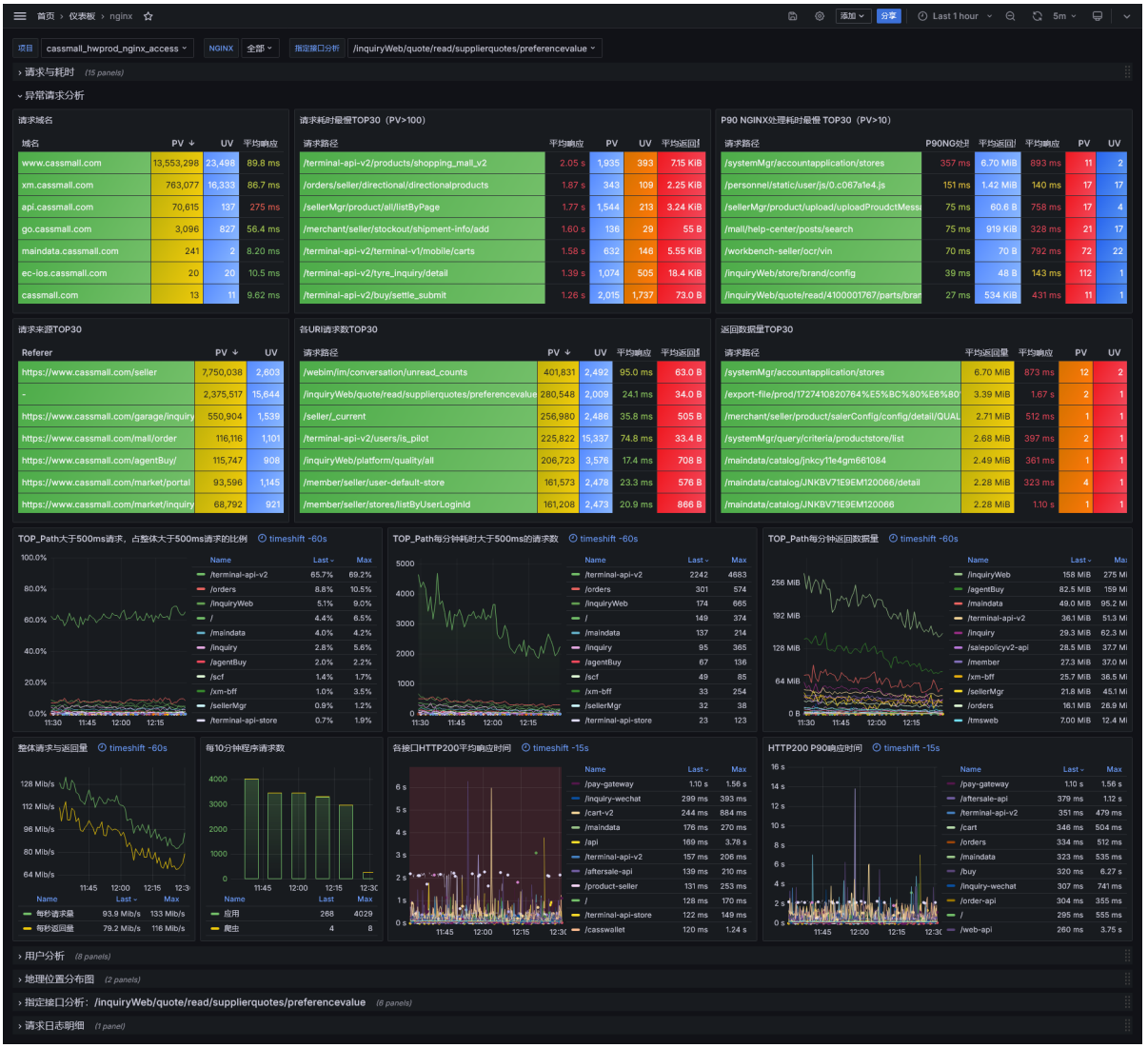
该看板是基于 ClickHouse + Vector 的NGINX请求日志分析看板。包括请求与耗时分析、异常请求分析、用户分析、地理位置分布图、指定接口分析、请求日志明细。

尤其在异常请求分析方面，总结多年异常请求分析经验，从各个角度设计大量异常请求的分析图表。

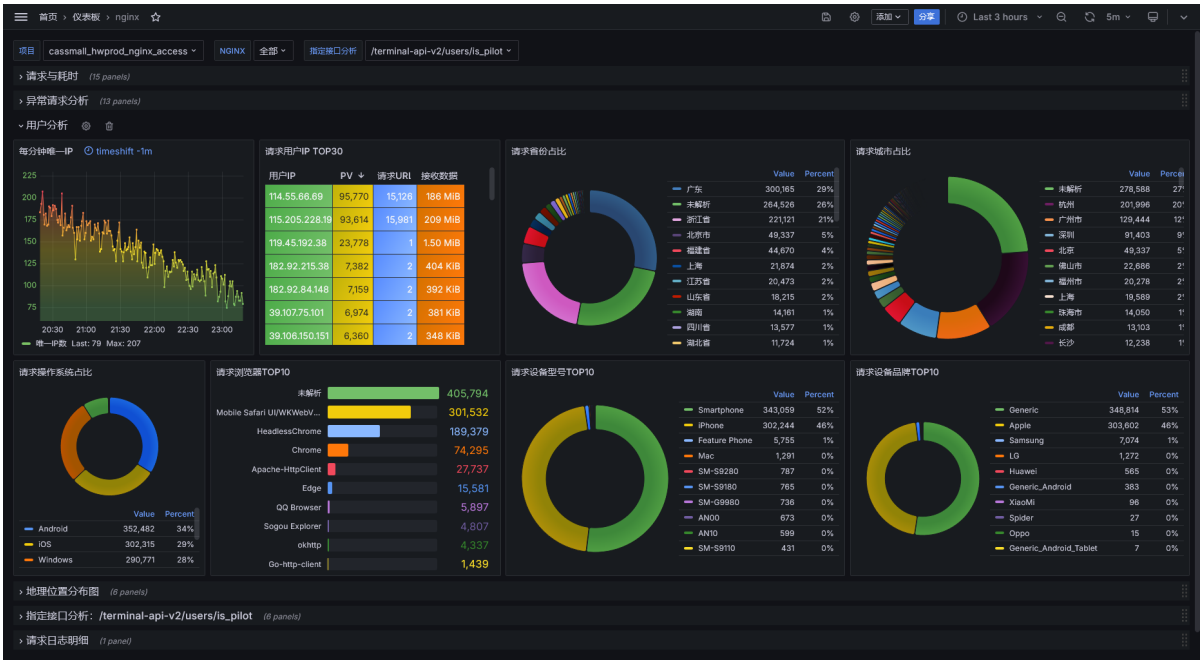
- 整体请求与耗时分析



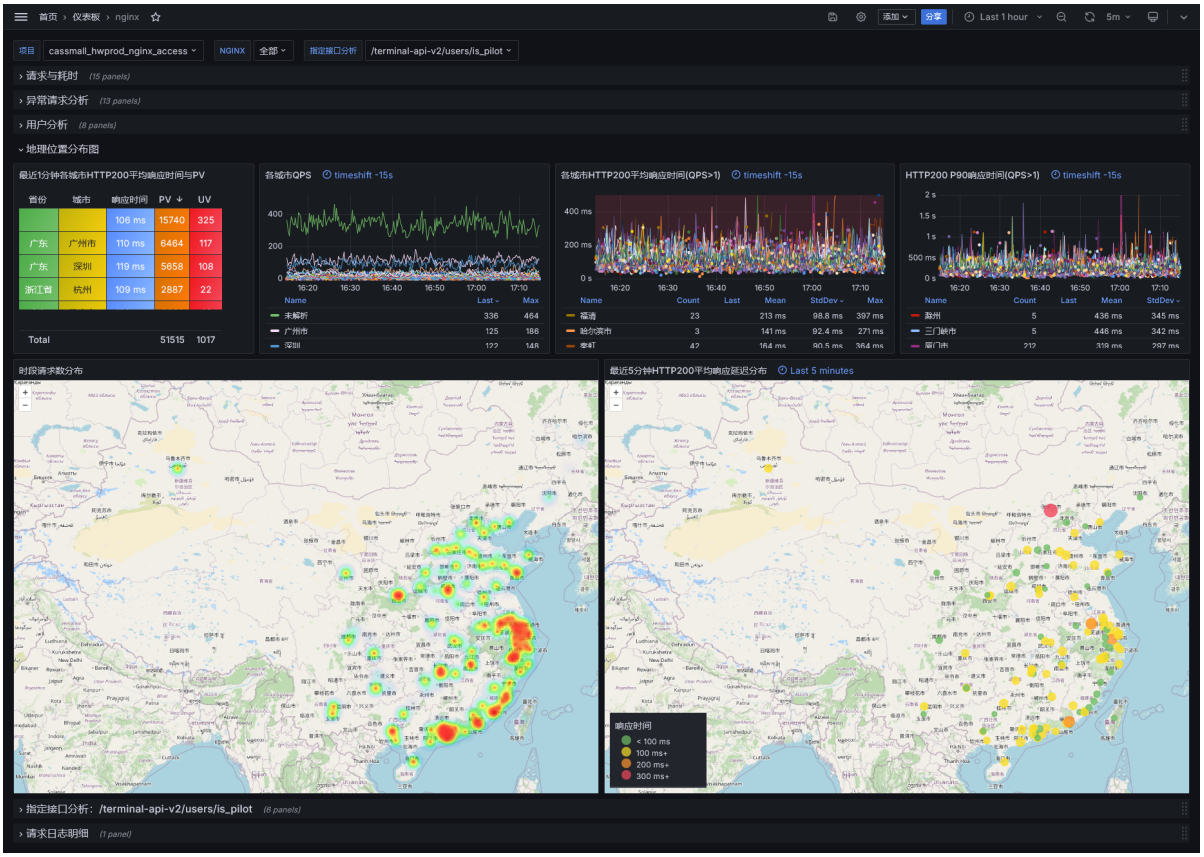
- NGINX异常请求分析



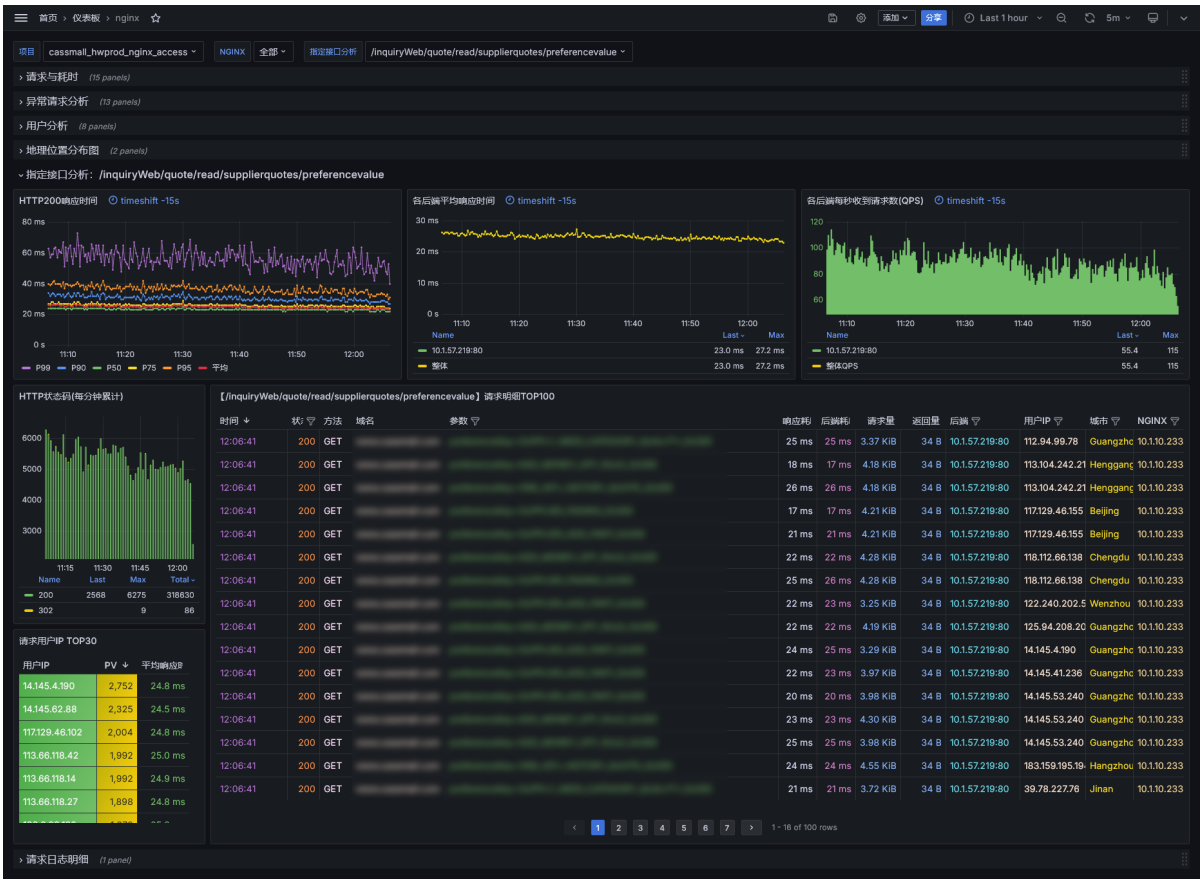
- 用户请求数据分析



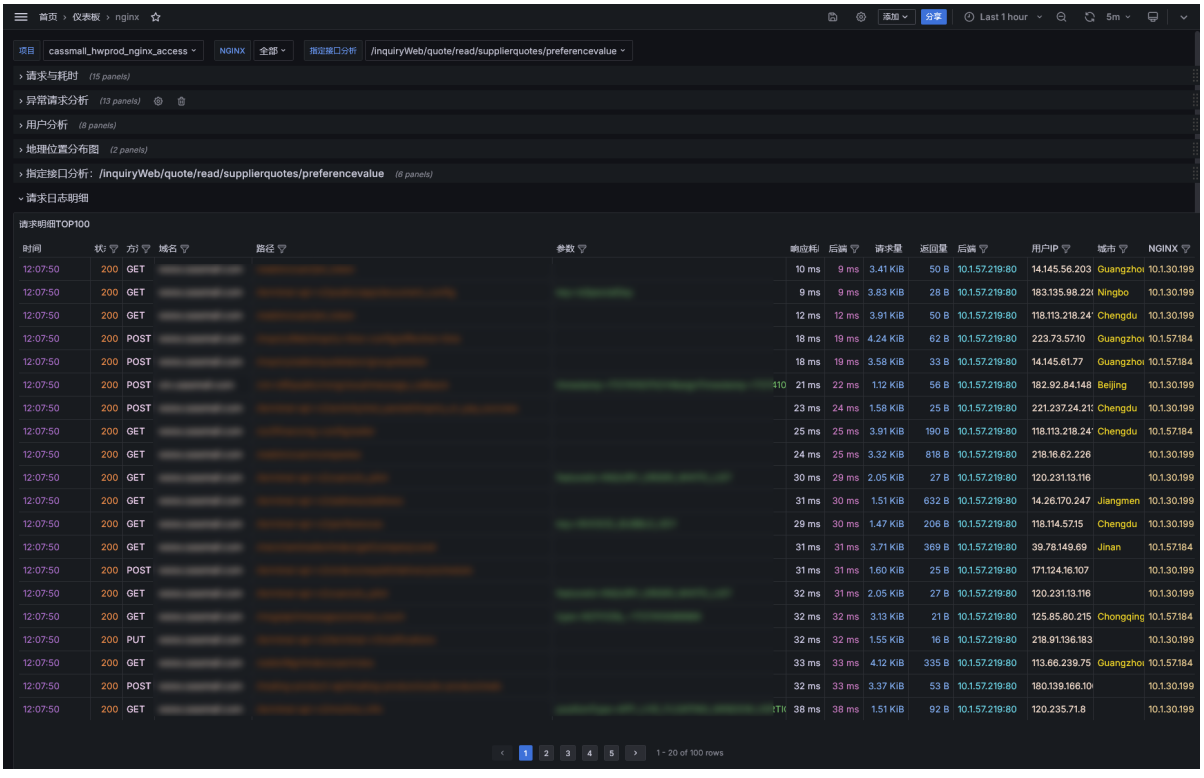
- 地理位置数据分析



- 指定接口明细分析



- 请求日志详情分析



修改NGINX日志格式

编辑NGINX配置文件`nginx.conf`（已配置为毫秒级别的时间格式）

```
    map "$time_iso8601 # $msec" $time_iso8601_ms { "~(^[^+]+)(\+[0-9:]+) # \d+\.(\d+)$" $1.$3$2; }
    log_format main
        '{"timestamp":"$time_iso8601_ms",'
```

```
        '"server_ip":"$server_addr",'
        '"remote_ip":"$remote_addr",'
        '"xff":"$http_x_forwarded_for",'
        '"remote_user":"$remote_user",'
        '"domain":"$host",'
        '"url":"$request_uri",'
        '"referer":"$http_referer",'
        '"upstreamtime":"$upstream_response_time",'
        '"responsetime":"$request_time",'
        '"request_method":"$request_method",'
        '"status":"$status",'
        '"response_length":"$bytes_sent",'
        '"request_length":"$request_length",'
        '"protocol":"$server_protocol",'
        '"upstreamhost":"$upstream_addr",'
        '"http_user_agent":"$http_user_agent"'
        '}';
```

```
# 配置完成后，重载NGINX
nginx -s reload
```

部署ClickHouse

```
# 创建部署目录和docker-compose.yaml
mkdir -p /opt/clickhouse/etc/clickhouse-server/{config.d,users.d}
cd /opt/clickhouse
cat <<-EOF > docker-compose.yaml
services:
  clickhouse:
    image: registry.cn-shenzhen.aliyuncs.com/starsl/clickhouse-server:23.4
    container_name: clickhouse
    hostname: clickhouse
    volumes:
      - /opt/clickhouse/logs:/var/log/clickhouse-server
      - /opt/clickhouse/data:/var/lib/clickhouse
      - /opt/clickhouse/etc/clickhouse-server/config.d/config.xml:/etc/clickhouse-server/config.d/config.xml
      - /opt/clickhouse/etc/clickhouse-server/users.d/users.xml:/etc/clickhouse-server/users.d/users.xml
      - /usr/share/zoneinfo/PRC:/etc/localtime
    ports:
      - 8123:8123
      - 9000:9000
EOF
```

```
# vi /opt/clickhouse/etc/clickhouse-server/config.d/config.xml
<clickhouse replace="true">
    <logger>
        <level>debug</level>
        <log>/var/log/clickhouse-server/clickhouse-server.log</log>
        <errorlog>/var/log/clickhouse-server/clickhouse-server.err.log</errorlog>
        <size>1000M</size>
```

```
        <count>3</count>
    </logger>
    <display_name>ch_accesslog</display_name>
    <listen_host>0.0.0.0</listen_host>
    <http_port>8123</http_port>
    <tcp_port>9000</tcp_port>
    <user_directories>
        <users_xml>
            <path>users.xml</path>
        </users_xml>
        <local_directory>
            <path>/var/lib/clickhouse/access/</path>
        </local_directory>
    </user_directories>
</clickhouse>
```

```
# 生成密码(返回的第一行是明文，第二行是密文)
PASSWORD=$(base64 < /dev/urandom | head -c8); echo "$PASSWORD"; echo -n "$PASSWORD" | sha256sum | tr -d '-'

# vi /opt/clickhouse/etc/clickhouse-server/users.d/users.xml
<?xml version="1.0"?>
<clickhouse replace="true">
    <profiles>
        <default>
            <max_memory_usage>10000000000</max_memory_usage>
            <use_uncompressed_cache>0</use_uncompressed_cache>
            <load_balancing>in_order</load_balancing>
            <log_queries>1</log_queries>
        </default>
    </profiles>
    <users>
        <default>
            <password remove='1' />
            <password_sha256_hex>填写生成的密码密文</password_sha256_hex>
            <access_management>1</access_management>
            <profile>default</profile>
            <networks>
                <ip>::/0</ip>
            </networks>
            <quota>default</quota>
            <access_management>1</access_management>
            <named_collection_control>1</named_collection_control>
            <show_named_collections>1</show_named_collections>

<show_named_collections_secrets>1</show_named_collections_secrets>
        </default>
    </users>
    <quotas>
        <default>
            <interval>
                <duration>3600</duration>
                <queries>0</queries>
                <errors>0</errors>
                <result_rows>0</result_rows>
```

```
                <read_rows>0</read_rows>
                <execution_time>0</execution_time>
            </interval>
        </default>
    </quotas>
</clickhouse>
```

```
# 启动
docker compose up -d
```

创建数据库与表

```
CREATE DATABASE IF NOT EXISTS nginxlogs ENGINE=Atomic;

CREATE TABLE nginxlogs.nginx_access
(
    `timestamp` DateTime64(3, 'Asia/Shanghai'),
    `server_ip` String,
    `domain` String,
    `request_method` String,
    `status` Int32,
    `top_path` String,
    `path` String,
    `query` String,
    `protocol` String,
    `referer` String,
    `upstreamhost` String,
    `responsetime` Float32,
    `upstreamtime` Float32,
    `duration` Float32,
    `request_length` Int32,
    `response_length` Int32,
    `client_ip` String,
    `client_latitude` Float32,
    `client_longitude` Float32,
    `remote_user` String,
    `remote_ip` String,
    `xff` String,
    `client_city` String,
    `client_region` String,
    `client_country` String,
    `http_user_agent` String,
    `client_browser_family` String,
    `client_browser_major` String,
    `client_os_family` String,
    `client_os_major` String,
    `client_device_brand` String,
    `client_device_model` String,
    `createdtime` DateTime64(3, 'Asia/Shanghai'),
)
ENGINE = MergeTree
PARTITION BY toYYYYMMDD(timestamp)
PRIMARY KEY (timestamp,
 server_ip,
```

```
 status,
 top_path,
 domain,
 upstreamhost,
 client_ip,
 remote_user,
 request_method,
 protocol,
 responsetime,
 upstreamtime,
 duration,
 request_length,
 response_length,
 path,
 referer,
 client_city,
 client_region,
 client_country,
 client_browser_family,
 client_browser_major,
 client_os_family,
 client_os_major,
 client_device_brand,
 client_device_model
)
TTL toDateTime(timestamp) + toIntervalDay(90)
SETTINGS index_granularity = 8192;
```

部署Vector采集日志

```
# 创建部署目录和docker-compose.yaml
mkdir -p /opt/vector/conf
cd /opt/vector
touch access_vector_error.log
wget GeoLite2-City.mmdb
cat <<-EOF > docker-compose.yaml
services:
  vector:
    image: registry.cn-shenzhen.aliyuncs.com/starsl/vector:0.41.1-alpine
    container_name: vector
    hostname: vector
    restart: always
    entrypoint: vector --config-dir /etc/vector/conf
    ports:
      - 8686:8686
    volumes:
      - /usr/local/openresty/nginx/logs:/nginx_logs  # 这是需要采集的日志的路径需要挂载到容器内
      - /opt/vector/access_vector_error.log:/tmp/access_vector_error.log
      - /opt/vector/GeoLite2-City.mmdb:/etc/vector/GeoLite2-City.mmdb
      - /opt/vector/conf:/etc/vector/conf
      - /usr/share/zoneinfo/PRC:/etc/localtime
EOF
```

```
# conf目录采集配置
cd /opt/vector/conf
cat <<-EOF > vector.yaml
timezone: "Asia/Shanghai"
api:
  enabled: true
  address: "0.0.0.0:8686"
EOF
```

```
# vi nginx-access.yaml
sources:
  01_file_nginx_access:
    type: file
    include:
      - /nginx_logs/access.log  #nginx请求日志路径(注意是挂载到容器内的路径)
transforms:
  02_parse_nginx_access:
    drop_on_error: true
    reroute_dropped: true
    type: remap
    inputs:
      - 01_file_nginx_access
    source: |
      .message = string!(.message)
      if contains(.message,"\\x") { .message = replace(.message, "\\x", "\\\\x") }
      . = parse_json!(.message)
      .createdtime = to_unix_timestamp(now(), unit: "milliseconds")
      .timestamp = to_unix_timestamp(parse_timestamp!(.timestamp , format: "%+"), unit: "milliseconds")
      .url_list = split!(.url, "?", 2)
      .path = .url_list[0]
      .query = .url_list[1]
      .path_list = split!(.path, "/", 3)
      if length(.path_list) > 2 {.top_path = join!(["/", .path_list[1]])} else {.top_path = "/"}
      .duration = round(((to_float(.responsetime) ?? 0) - (to_float(.upstreamtime) ?? 0)) ?? 0,3)
      if .xff == "-" { .xff = .remote_ip }
      .client_ip = split!(.xff, ",", 2)[0]
      .ua = parse_user_agent!(.http_user_agent , mode: "enriched")
      .client_browser_family = .ua.browser.family
      .client_browser_major = .ua.browser.major
      .client_os_family = .ua.os.family
      .client_os_major = .ua.os.major
      .client_device_brand = .ua.device.brand
      .client_device_model = .ua.device.model
      .geoip = get_enrichment_table_record("geoip_table", {"ip": .client_ip}) ?? {"city_name":"unknown","region_name":"unknown","country_name":"unknown"}
      .client_city = .geoip.city_name
      .client_region = .geoip.region_name
      .client_country = .geoip.country_name
      .client_latitude = .geoip.latitude
      .client_longitude = .geoip.longitude
```

```
      del(.path_list)
      del(.url_list)
      del(.ua)
      del(.geoip)
      del(.url)
sinks:
  03_ck_nginx_access:
    type: clickhouse
    inputs:
      - 02_parse_nginx_access
    endpoint: http://10.7.0.26:8123  #clickhouse http接口
    database: nginxlogs  #clickhouse 库
    table: nginx_access  #clickhouse 表
    auth:
      strategy: basic
      user: default  #clickhouse 库
      password: GlWszBQp  #clickhouse 密码
    compression: gzip
  04_out_nginx_dropped:
    type: file
    inputs:
      - 02_parse_nginx_access.dropped
    path: /tmp/access_vector_error.log  #解析异常的日志
    encoding:
      codec: json
enrichment_tables:
  geoip_table:
    path: "/etc/vector/GeoLite2-City.mmdb"
    type: geoip
    locale: "zh-CN"
```

```
# 启动
cd /opt/vector
docker compose up -d
```

Grafana新增ClickHouse数据源

在Grafana中增加ClickHouse数据源时，注意点开`Additional settings`右边的箭头，配置`Default database`为存放日志的默认库，如上的：`nginxlogs`。

导入NGINX请求日志分析的Grafana看板

Grafana看板ID：`22037`

下载地址：

https://grafana.com/grafana/dashboards/22037

注意：如果你保存日志的表名不是`access`结尾的，`项目`菜单会没有数据，需要点击看板右上角的`设置`-`变量`-`project`，在下方的`Regex`项，输入你需要展示的日志表的正则，或者留空，展示默认库的所有表。