云压测

实践教程

腾讯云

【版权声明】

【商标声明】

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

实践教程

使用 Prometheus 观测性能压测指标

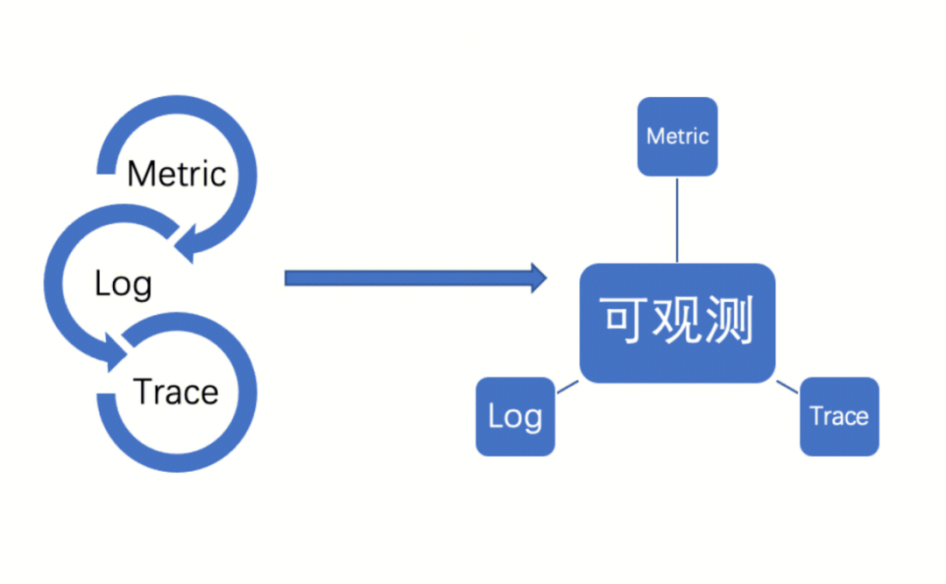
最近更新时间：2024-04-02 11:10:41

性能压测是一个充满挑战的领域，性能评估和优化贯穿开发、测试、部署、上线等各个阶段，直接影响系统运行效率和用户使用体验。本文将为您介绍在压测过程中 Prometheus 存储指标，并使用 Grafana 将指标可视化。通过观测指标的动态变化，发现系统瓶颈。

性能压测可观测

可观测体系主要包含3类指标：Metrics、Logs、Traces。三者既可独立工作，也可相辅相成，推导出系统整体的状况。

- Metrics：是一种聚合的度量数值，能够量化系统各个维度指标，常用于提供系统全局视图，一般包括 Counter、Gauge、Histogram 等指标类型。
- Logs：应用程序运行过程中产生的日志或者事件，提供系统运行的上下文信息，例如：某个变量值、发生错误详情等。
- Traces：提供请求从发送到完成响应整个链路。在分布式系统中，一个请求完成需要经过多个服务，Trace 提供请求在链路各个环节的响应时间、响应体、是否报错等。通过 Trace 更方便分析出请求中的异常环节。



指标概述

Counter：只增不减的计数器

Counter 是一种累积度量指标，用于表示一个只能增加的值，如果重置系统或服务，Counter 可能会从0开始重新计数。Counter 最常见的用途是计数器，用于记录发生的事件数量，例如请求的数量、完成的任务数、错误的数量等。例如 Counter 类型指标：http_requests_total，用于记录请求次数。

通过 rate()函数获取请求 QPS:

```
rate(http_requests_total[5m])
```

查询系统访问量前10的 http 请求:

```
topk(10, http_requests_total)
```

Gauge：有增有减的度量衡

Gauge 与 Counter 不同，Counter 用来反映事件发生次数，而 Gauge 用来反映系统当前的状态，例如当前的温度、服务器 CPU/内存使用率、剩余可用内存等。例如：通过 Gauge 指标，查看节点剩余可用内存:

```
# HELP node_memory_MemAvailable_bytes Memory information field
MemAvailable_bytes.
# TYPE node_memory_MemAvailable_bytes gauge
node_memory_MemAvailable_bytes
```

查看剩余可用内存比例（这个语句通常可用来设置警报，当剩余可用内存低于某个阈值时进行通知，这样您就可以采取行动避免系统出现内存不足的情况）:

```
(node_memory_MemFree_bytes/node_memory_MemTotal_bytes)*100
```

Histogram/Summary：分析数据分布

Histogram 和 Summary 主要用于统计和分析样本的分布情况。
以 Histogram 为例，通常使用 histogram_quantile 函数计算百分位数。例如，要计算请求响应时间的第90百分位数:

```
histogram_quantile(0.9, rate(http_response_duration_seconds_bucket[5m]))
```

这个查询会返回过去5分钟内所有记录的请求持续时间的第90百分位数。通过这种方式，Histogram 为您提供了强大的工具来监控和理解您的系统性能。

设计观测指标

可以通过性能压测暴露系统瓶颈，通过选择合适可观测工具、设计科学的指标监控，根据指标变化来帮助您定位系统瓶颈。

核心指标

压测过程中，从施压方看主要需要关注以下核心指标：

- 请求响应时间：包括平均响应时间、响应时间百分比水位。
- 请求 RPS
- 请求成功率，失败率。
- 请求错误原因
- 压测并发数
- 检查点成功数，失败数。
- 请求发送接收字节数
- 施压机内存/CPU 使用率等

指标维度

每个指标最好能够区分不同的维度，例如：

1. 按返回码统计不同请求的 QPS，例如返回码为200的请求的 QPS 是多少，返回码为500的请求的 QPS 是多少?
2. 统计发往不同服务的请求的 QPS，例如压测 `www.test1.com` 和 `www.ok1.com` 的请求的 QPS 分别是多少?

以 http 请求为例，常见的维度划分如下：

- job：压测任务标识，每一次压测都是一个不同的 job。
- method：请求方法，例如 GET、POST、HEAD 等。
- proto：请求协议，例如 http、https、http2。
- service：请求地址，例如 `https://www.test1.com` 。
- status：请求响应码，例如200、404、500等。
- result：标识请求响应码，例如200或者其他小于400对应 OK，404对应 Not Found，500对应 Internal Error 。
- check：检查点/断言名字，一般用于简单描述该断言的作用。

指标设计

基于此我们设计了一套最基本、常用的指标体系，能够覆盖绝大部分用户的使用需求，详情请参见下表。您也可以基于如下指标体系进行拓展，以满足不同的需求。

Metric	Type	Labels	Description
req_total	Counter	job,method, proto,service, status, result	请求次数
req_duration_seconds	Histogra	job,method, proto,service, status,	请求耗时

	m	result	
checks_total	Counter	job, check, result	检查点
send_bytes_total	Counter	job,method, proto,service, status, result	发送字节数
receive_bytes_total	Counter	job,method, proto,service, status, result	接收字节数
num_vus	Gauge	job	并发数，可用于指代虚拟用户数量。如果是JMeter压测，也可用于指代线程数

示例

查看实时的并发用户数：

```
sum(num_vus{job=~"$job"})
```

查看请求的 QPS：

```
# 查看不同请求的qps
sum(rate(req_total{job=~"$job"}[1m])) by (service)
# 查看成功请求的qps
sum(rate(req_total{job=~"$job",result="ok"}[1m]))
# 查看指定http://www.test1.com服务的qps
sum(rate(req_total{job=~"$job",service="http://www.test1.com"}[1m]))
```

查看请求失败率：

```
# 请求总体失败率
sum(rate(req_total{job=~"$job",result!="ok"}
[1m]))/sum(rate(req_total{job=~"$job"}[1m]))
# 基于请求维度，查看各请求的失败率
sum(rate(req_total{job=~"$job",result!="ok"}[1m])) by
(service)/sum(rate(req_total{job=~"$job"}[1m])) by (service)
```

查询请求响应时间：

```
# 查询请求平均响应时间
```

```
sum(rate(req_duration_seconds_sum{job=~"$job"}
[15s]))/sum(rate(req_duration_seconds_count{job=~"$job"}[15s]))
# 查询请求中位数（50百分位）响应时间
histogram_quantile(0.50, sum(rate(req_duration_seconds_bucket{job=~"$job"}[1m]))
by (le))
# 查询请求90百分位响应时间
histogram_quantile(0.90, sum(rate(req_duration_seconds_bucket{job=~"$job"}[1m]))
by (le))
```

查询请求出入带宽：

```
# 请求出带宽汇总
sum(rate(send_bytes_total{job=~"$job",region=~"$region"}[1m]))
# 请求入带宽汇总
sum(rate(pts_engine_receive_bytes_total{job=~"$job",region=~"$region"}[1m]))
```

检查点成功率（用户自定义的 test 语句，在JMeter 中对应断言）：

```
# 检查点成功率
sum(rate(checks_total{job=~"$job", result="ok"}
[1m]))/sum(rate(checks_total{job=~"$job"}[1m]))
# 指定检查点成功率
sum(rate(checks_total{job=~"$job", result="ok",check="response contains hello"}
[1m]))/sum(rate(checks_total{job=~"$job", check="response contains hello"}[1m]))
```

说明：

- 用户也可以基于以上 demo 进行灵活扩充，对被压测服务进行同样的监控。在压测过程中，对比查看压测平台生成的指标报告与用户服务的指标报告，综合分析排查问题。
- 使用 Prometheus 存储以上指标，使用查询语句在 Prometheus 中查询数据，最后通过 Grafana 可视化展示以上数据。一边压测，一边实时观测服务性能指标变化。

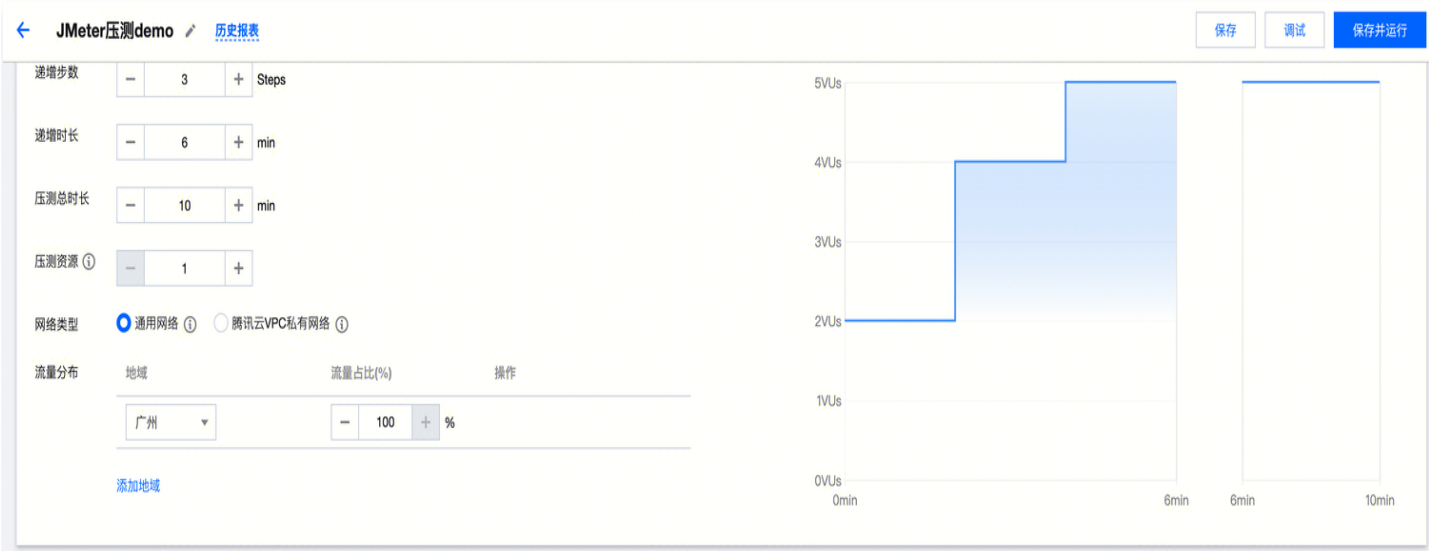
操作步骤

1. 登录 云压测控制台 。
2. 在左侧菜单栏中单击**测试场景**。
3. 在测试场景页面单击**新建场景**。
4. 在创建测试场景页面选择“JMeter”压测类型，并单击**开始**，创建压测场景。

上传 jmx 脚本

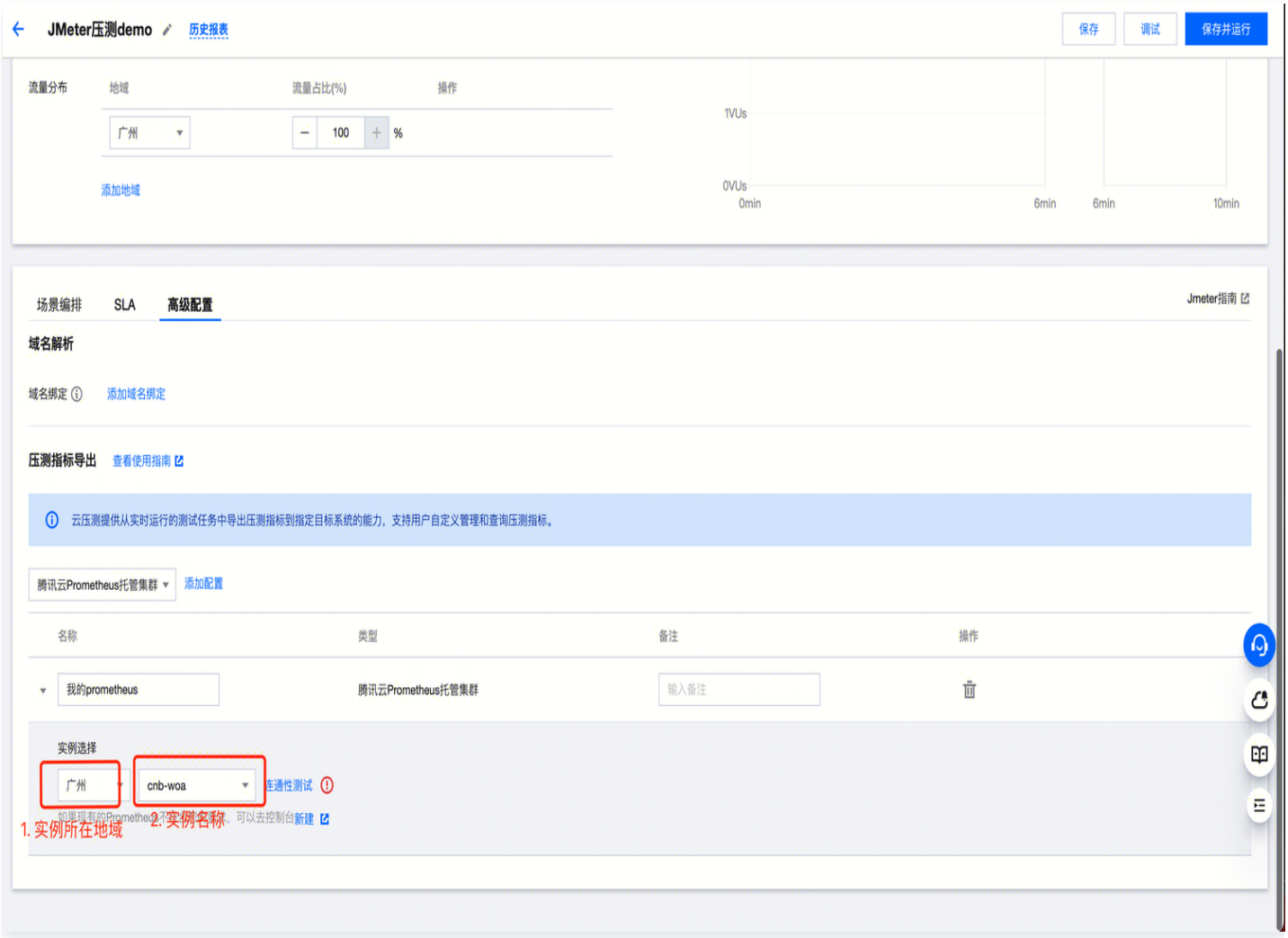
- 必选：上传 jmx 脚本

- 可选：
  - 上传 Jar 包：如果您的脚本中使用了 JMeter 三方插件，您可以上传对应 jar 包，来拓展 JMeter 功能。
  - 上传 properties 文件：在原生 jmeter.properties 文件基础上，自定义 JMeter 属性。
  - csv 文件：在 Jmeter 中读取 csv 文件中的数据，作为变量在脚本中引用。
  - 其他文件：任何在 jmx 中脚本引用的其他文件



将报告导出到腾讯云 Prometheus

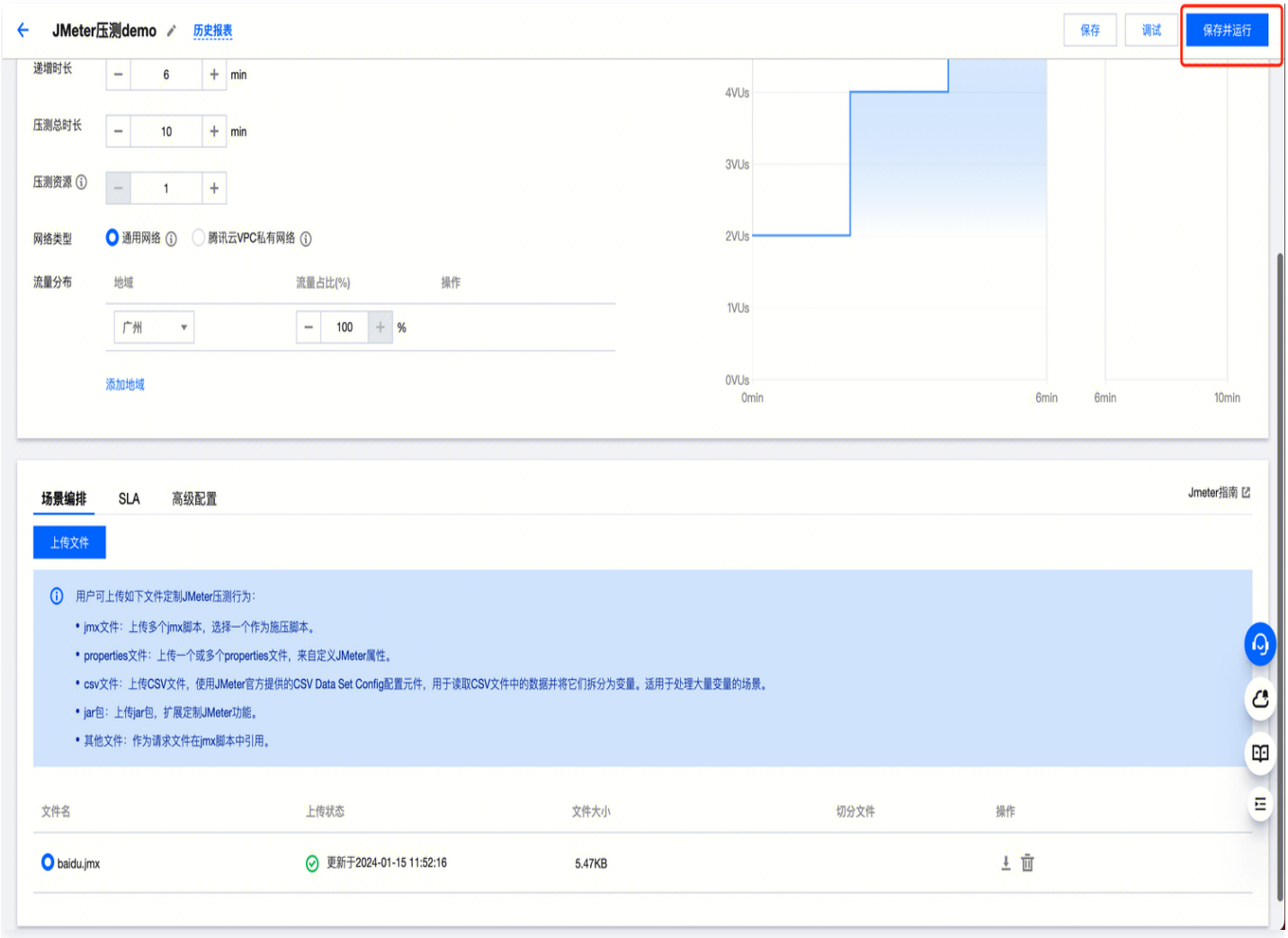
在**高级配置 > 压测指标导出 > 腾讯云 Prometheus 托管集群**中，点击**添加配置**， 选择您的实例：



如果您在该地域没有实例，您也可以点击**新建**。

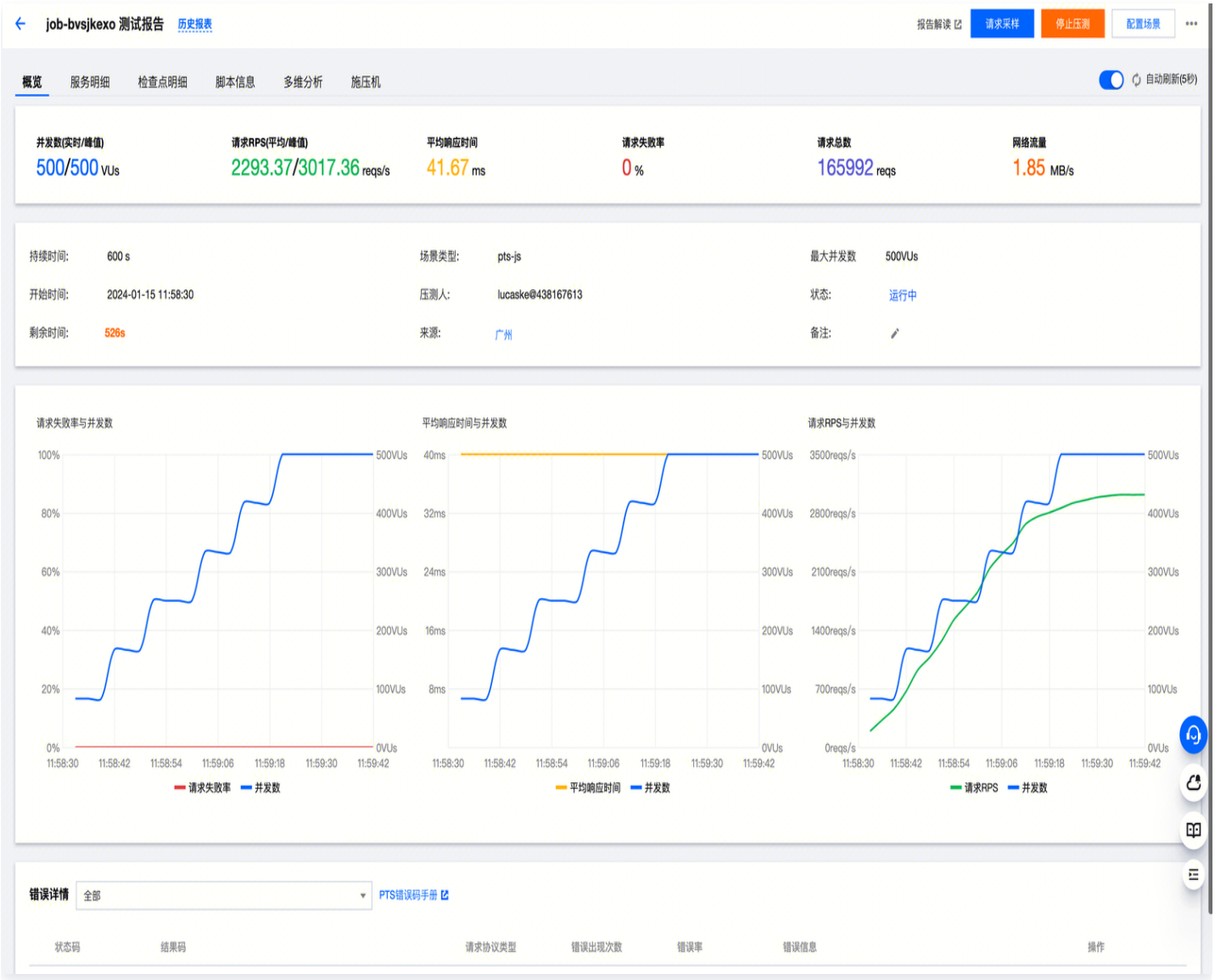
运行压测脚本

点击**保存并运行**即可开始压测。



查看实时报告

您可以选择不同的页签，查看不同维度的报告详情。

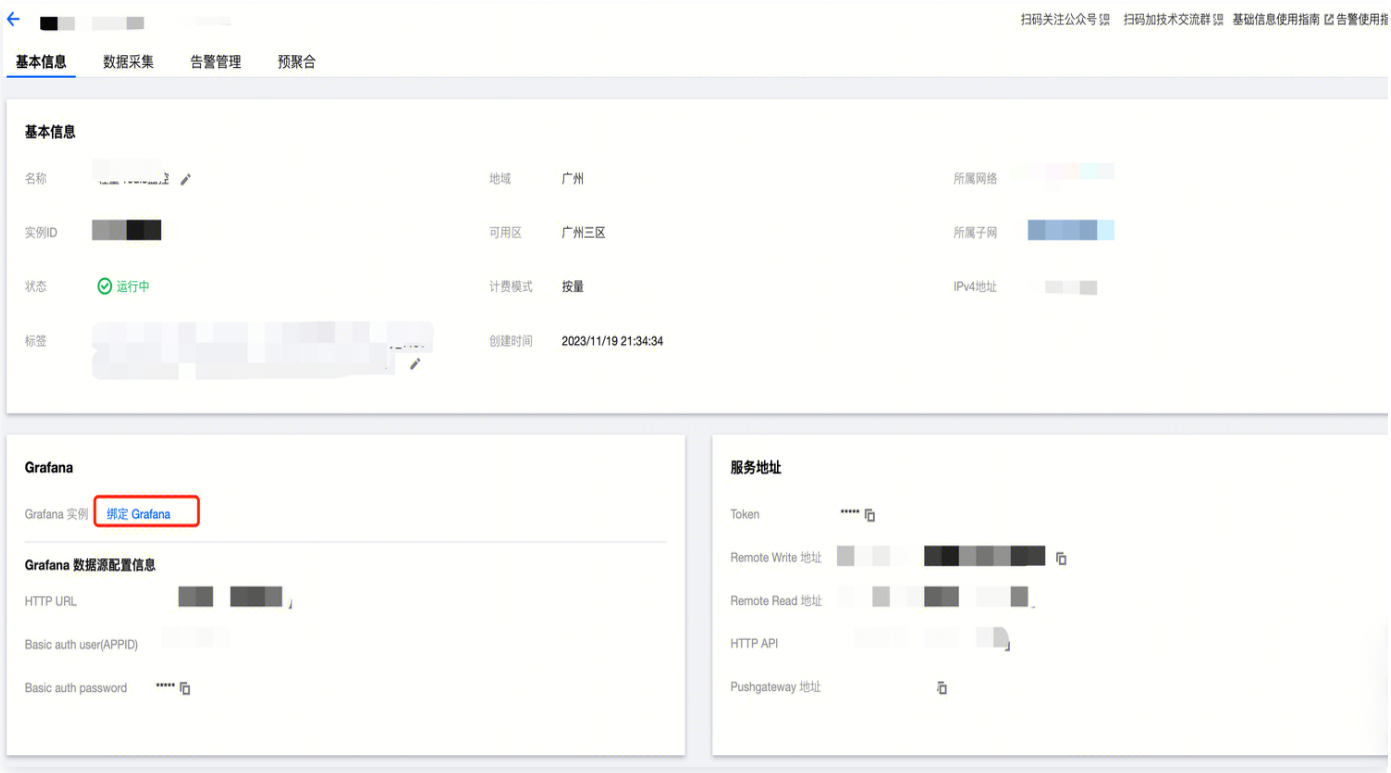


在 Prometheus 中查看压测报告

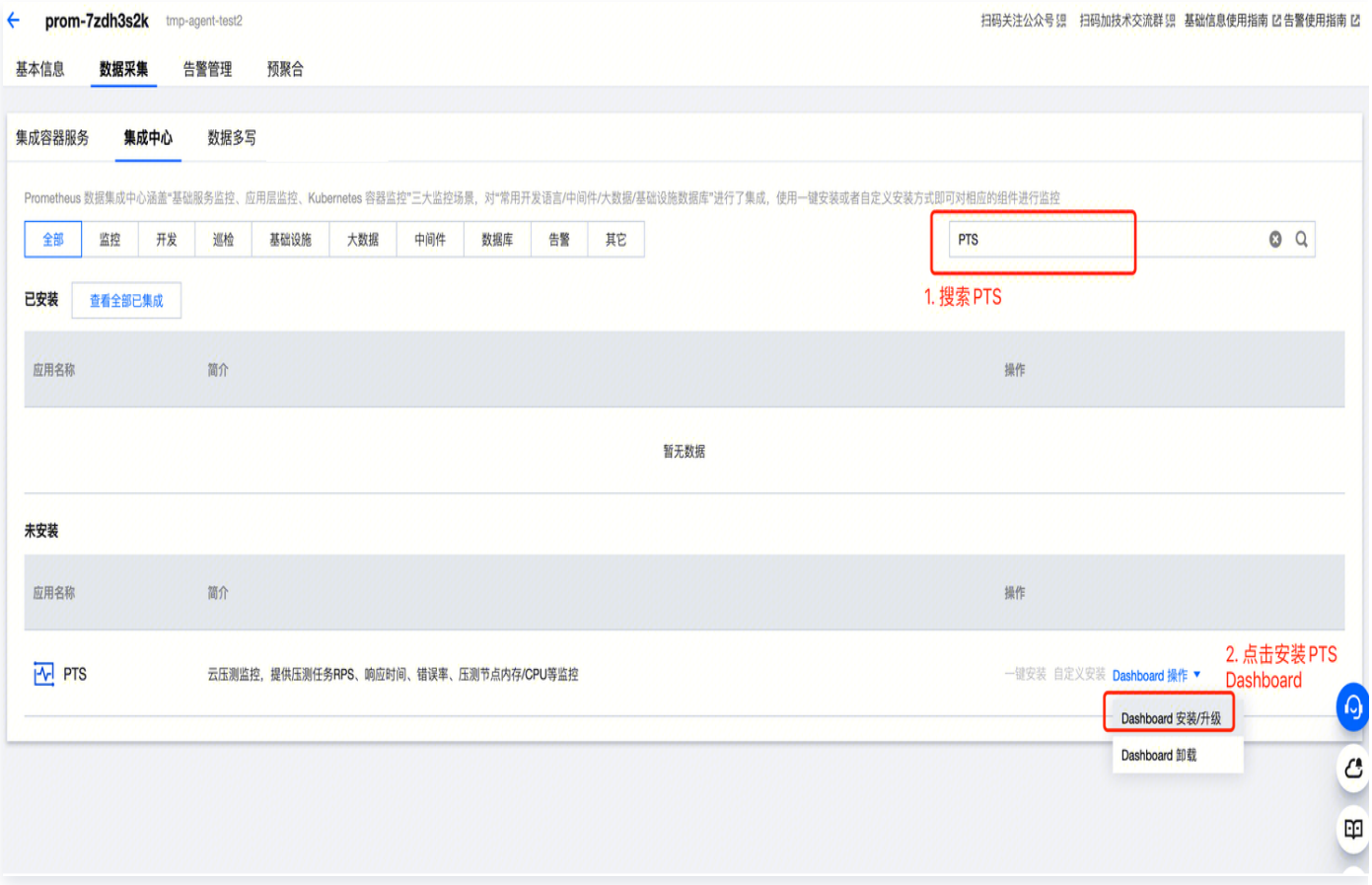
在压测过程中，指标会同步发送到腾讯云 Prometheus 中。您也可以在 Prometheus 中查询指标信息，自定义查询语句。

1. 登录 Prometheus 控制台，在搜索栏根据实例名称找到您的 Prometheus 实例。

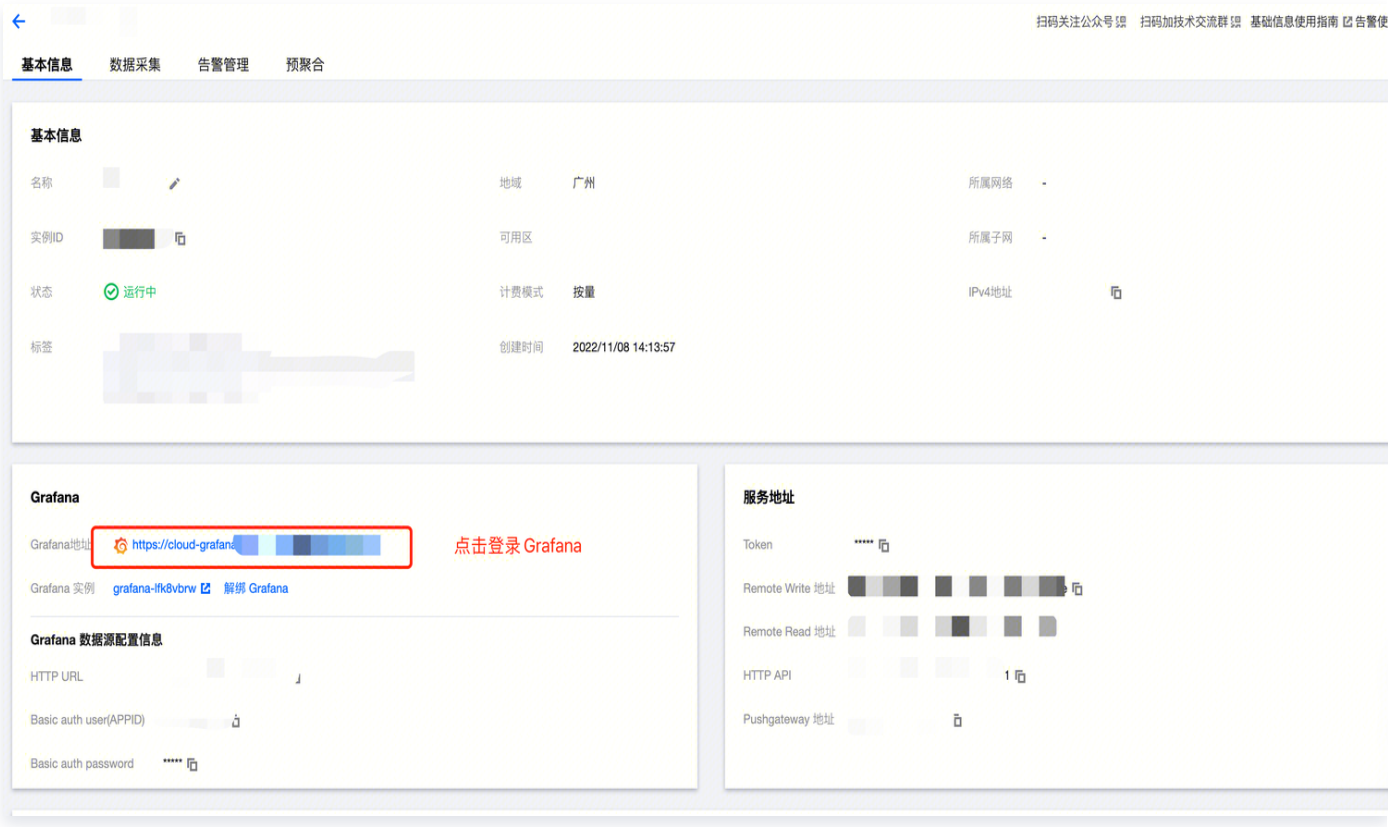
2. 单击实例 ID，进入实例详情。
   如果您的 Prometheus 实例没有对应的 Grafana，可以点击绑定 Grafana。通过 Grafana 来展示 Prometheus 指标。



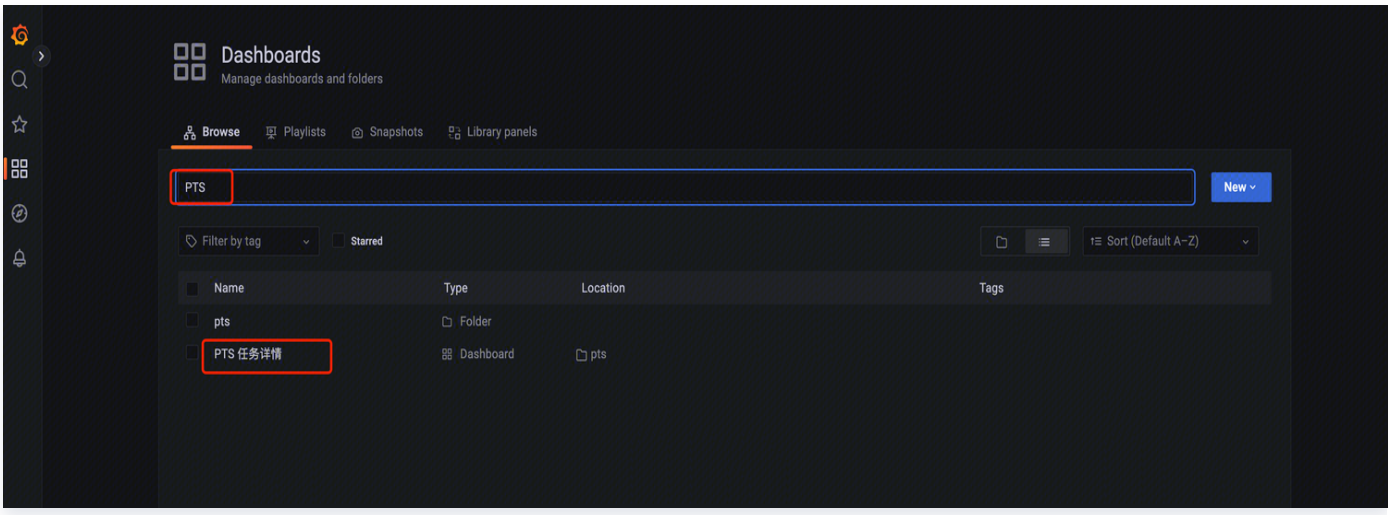
3. 选择**数据采集 >集成中心**，搜索云压测 PTS 应用，在 **Dashboard 操作**中，点击 **Dashboard 安装/升级**，将云压测监控面板安装到 Prometheus 绑定的 Grafana 上。



4. 在 Grafana 中查看压测报告，登录 Prometheus 绑定的 Grafana 页面。

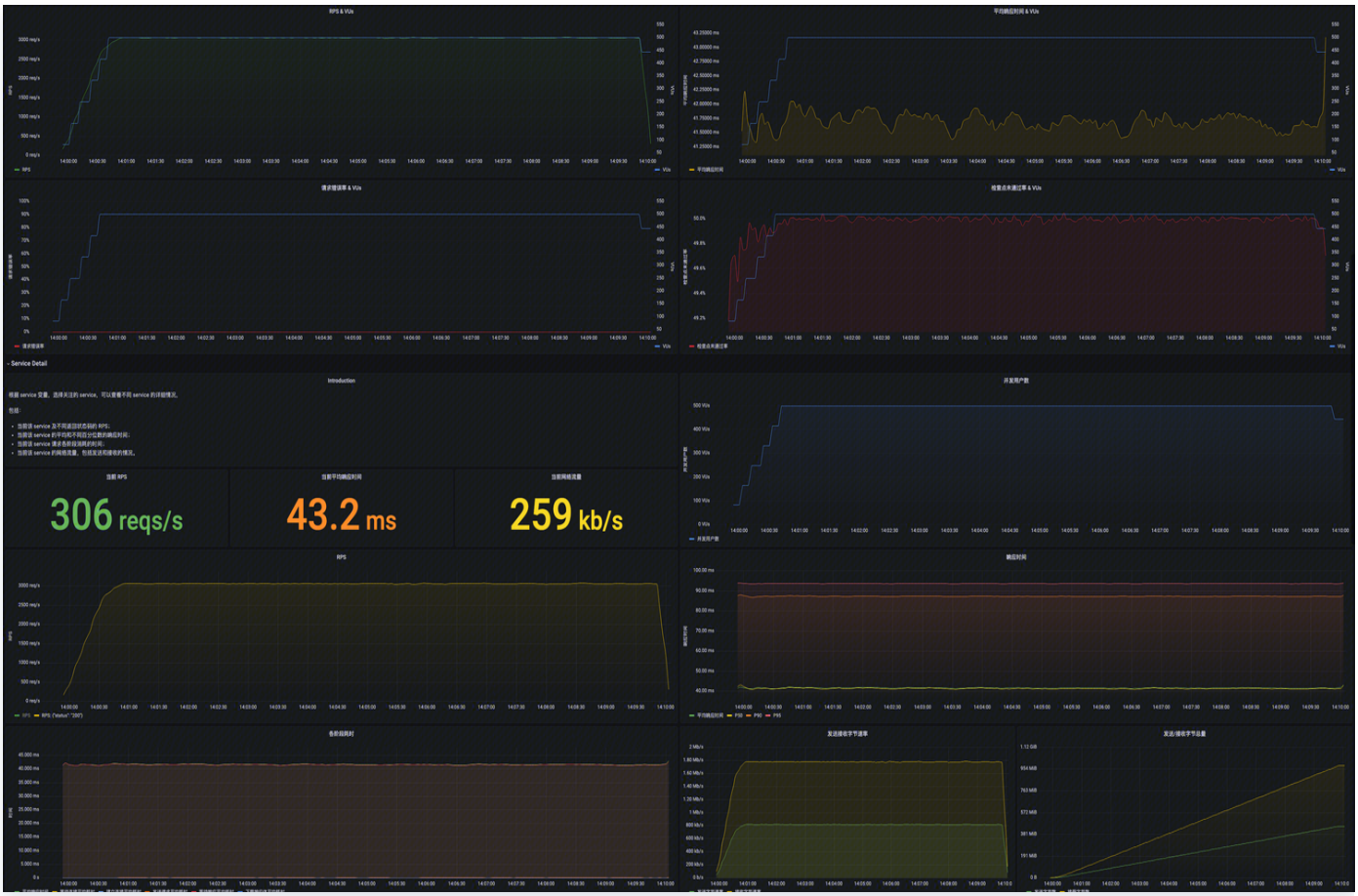


5. 搜索 PTS，查看云压测压测报告。



6. 点击 PTS 任务详情，查看报告详情。


Introduction
Introduction
Overview
当前 VUs 值
445 VUs
并发用户数
RPS
当前 RPS
306 reqs/s
当前平均响应时间
43.2 ms
平均响应时间
当前网络流量
259 kb/s
网络流量
当前请求错误率
0%
请求错误率
当前检查点来通过率
49.7%
检查点来通过率


RPS & VUs
平均响应时间 & VUs
请求错误率 & VUs
检查点来通过率 & VUs
Service Detail
Introduction
当前 RPS
306 reqs/s
当前平均响应时间
43.2 ms
当前网络流量
259 kb/s
并发用户数
RPS
响应时间
各阶段耗时
发送/接收字节速率
发送/接收字节总量

压测时如何评估系统瓶颈

在压测时，主要关注系统吞吐量（RPS，网络带宽）、响应时间、并发用户数等。公式如下：
RPS = VU（并发数）/ 平均响应时间

如何理解压测公式?

以一个线程循环去执行某个请求为例：如果请求平均响应时间是10ms，那么一个并发每秒可执行100个请求，对应的 RPS = 100req/s。
从这个公式可以推导，如果想提升压测的总体 RPS，有以下几种方法：

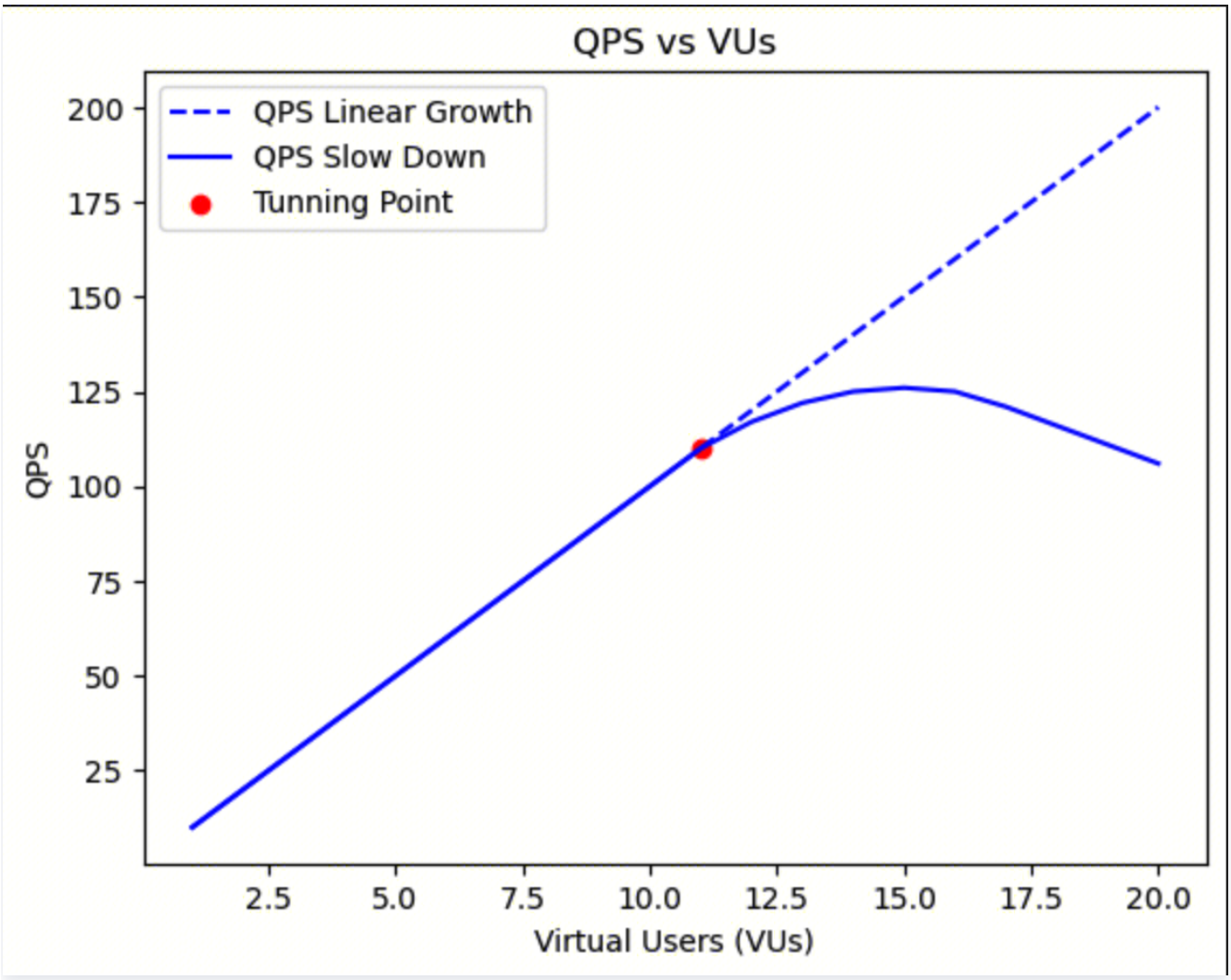
- 增加并发数，且请求平均响应时间保持不变。
  这种情况就是被压服务还没有饱和，压测 RPS 随着并发压力的增加而增加。
- 降低请求平均响应时间， VU 保持不变。
  这种情况比较理想，请求平均响应时间降低，代表被压服务进行了优化，单个 VU 单位时间内能够发送更多的请求。

在现实压测中，很可能随着 VU 增加，被压系统压力增加，响应时间也随之增加。

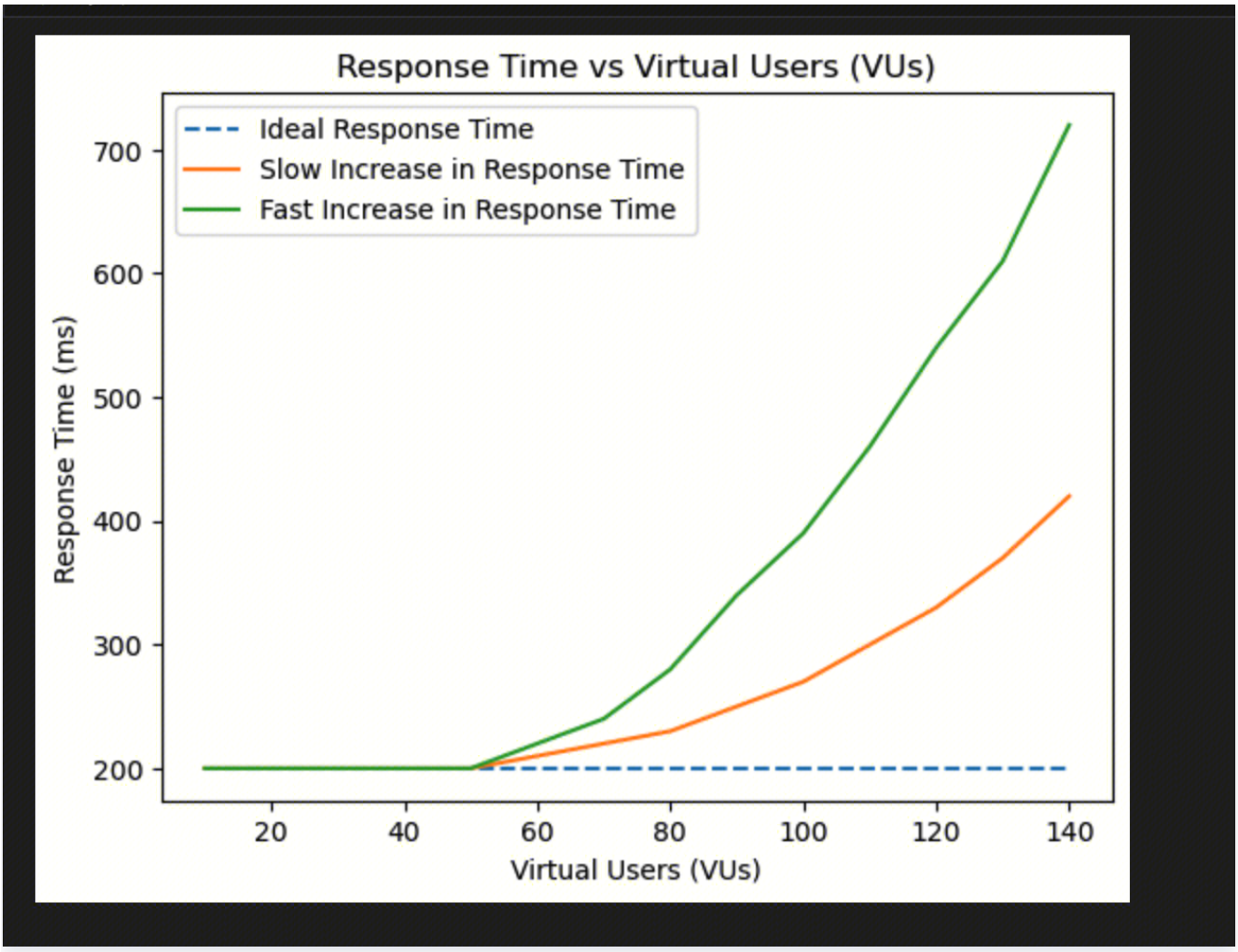
- 如果响应时间增加系数小于 VU 增加系数，总体 RPS 还是在变大，系统还未达到瓶颈。
- 如果响应时间增加系数大于 VU 增加系数，压测的表现就是随着 VU 增大，总体 RPS 反而降低，此时系统已经达到瓶颈。

评估系统性能拐点

压测就是在压力不断增加情况下，找到业务系统扩展性的拐点，即系统瓶颈/最大容量。
在一定的阶段，我们观测到扩展性是线性变化的。到达某一点时，此时对于资源的争夺开始影响性能。这一点也可以认为是系统拐点。作为曲线分界，过了拐点，整体的吞吐量会随着资源争夺加剧偏离线性扩展。最终，资源争夺的开销反而导致完成的请求数变少，吞吐量反而下降。



这种情况可能在系统负载到达 100% 使用率（饱和点）之后发生，也可能在接近100% 使用率的时候，这个时候排队比较明显。
例如：有一个计算密集型系统，在更多请求进来时候，需要更多的线程来执行请求。当CPU使用率接近100%时，由于CPU调度延时增加，性能开始下降。在性能达到峰值后，整体吞吐量反而会随着更多线程加入而下降。线程加入会导致更多上下文切换，消耗CPU资源，实际完成的任务反而变少。
性能的非线性变化，我们也可以通过响应时间的变化来看出来：



性能下降的原因非常多，除了上面提到的频繁上下文切换外，还有如下原因：

- 系统内存不够，开始频繁的换页（swap）来补充内存。
- 随着系统磁盘 IO 增加，磁盘 IO 可能进入缓冲排队。
- 系统内部实现队列算法，来进行削峰操作，导致请求处理等待时间变长。

使用 GoReplay 录制回放请求

最近更新时间：2024-04-02 11:10:41

前言

本文将通过一个实例演示如何使用 GoReplay 录制 Nginx 网关接收到的请求，并将请求各个字段保存成 CSV 文件。在云压测中，通过上传 CSV 参数文件，指定期望的并发数，分布式回放请求到用户指定的地址。

GoRePlay 简介

GoReplay 是一个开源的流量录制回放工具。主要用于捕获实时流量并将其复制到测试环境中。
由于 GoReplay 本身并不提供一个分布式运行方案，只能在单机上运行。在流量录制完成后，受限于单机资源瓶颈，我们很难大规模的重放录制的流量，无法 有效的模拟真实用户流量的压测行为以及极限测试。而腾讯云云压测是一款分布式性能测试服务，可模拟海量用户的真实业务场景。因此我们可以引入云压测，使用云压测来回放 GoReplay 录制的真实流量。

常见 GoReplay 使用场景

- 性能测试：通过复制生产环境的流量到测试环境，可以在不影响真实用户的情况下对应用程序进行压力测试和性能评估。
- 故障排除和调试：当生产环境出现问题时，可以捕获相关的流量并在一个隔离的环境中重放，以便开发人员可以安全地调试问题而不会影响实际服务。
- 回归测试：在发布新版本之前，可以使用 GoReplay 捕获的流量来验证更改是否会引入新的错误或性能问题。
- A/B 测试：可以将流量同时发送到两个服务版本，比较它们的表现，以便做出数据驱动的决策。
- 通过在回放时候，加大回放请求的倍数，模拟高流量情况，可以帮助确定在不同负载下所需的资源量。

GoReplay 流量录制原理

GoReplay 流量录制是监听指定端口流量，录制成 gor 文件（或者发送到其他目的端），方便后续回放。

```
sudo gor  -- input - raw  : 8080   -- output - file requests . gor
```

开始录制回放用户 Nginx 网关

本文以录制回放 Nginx 网关为例，其他所有类型的网关都可以按照相同的方式来录制请求，然后使用云压测来回放用户请求。

环境准备

- Nginx 网关：Nginx 网关上有源源不断的用户请求，需要在 Nginx 网关录制下这些请求。
- GoReplay：请求录制回放工具。

安装 GoReplay 至网关所在机器上，如果网关所在机器是 Linux 或 macOS，可以使用以下命令：

```
# 从官方 GitHub 仓库下载最新的二进制文件
curl -L
https://github.com/buger/goreplay/releases/download/1.3.3/gor_1.3.3_x64.tar.gz |
tar xz

# 将二进制文件移动到你的PATH目录中，例如/usr/local/bin
mv gor /usr/local/bin/
```

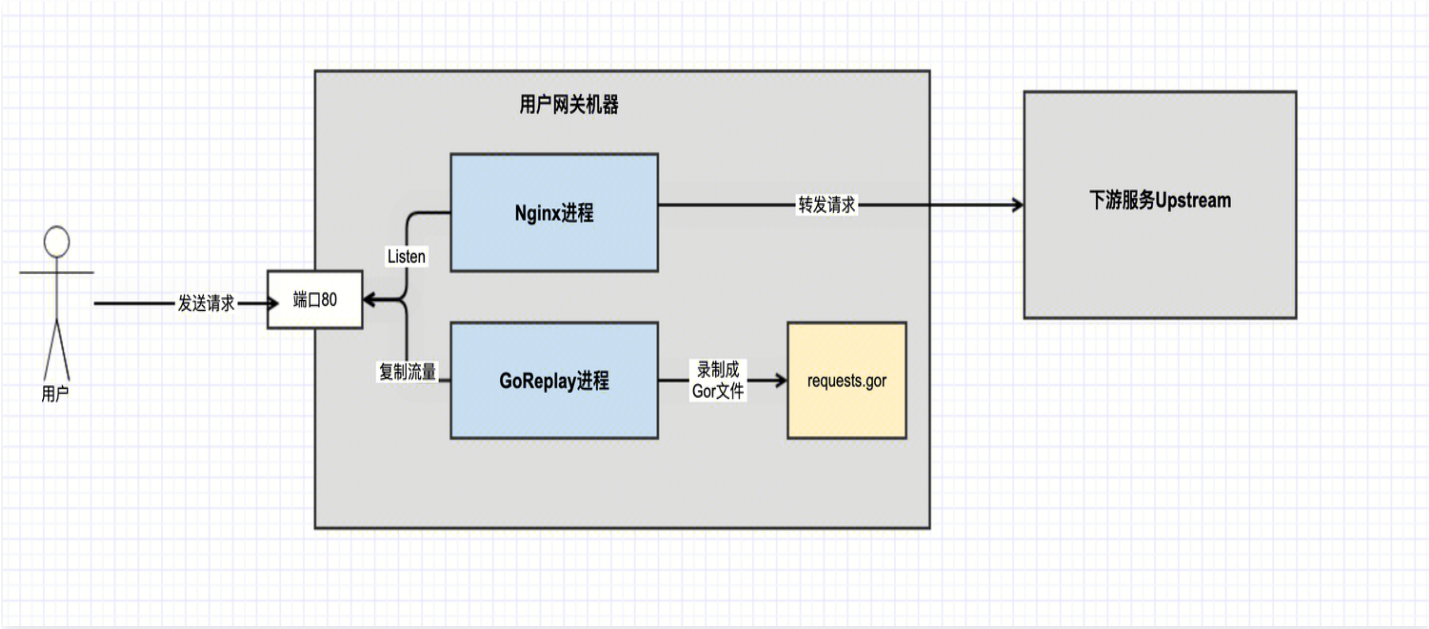
> **说明：**
> 确保替换上面的 URL 中的版本号为最新的版本，仓库地址：
> https://github.com/buger/goreplay 。

- CSV 生成服务：接收 HTTP 请求，将接收到的请求各个字段写入 CSV 文件中。
- 云压测：基于用户上传的 CSV 文件，回放用户录制的所有请求。

实验流程

将 Nginx 上的请求录制成 Gor 文件

本节参与组件（其他组件仅做完整场景展示）：Nginx 网关、GoReplay。
整体架构图如下：



开始录制流量前，需要在网关所在服务器上运行 GoReplay。以下是一个基本的命令示例，它会监听网关上的80端口，并将捕获的流量保存到一个文件中：

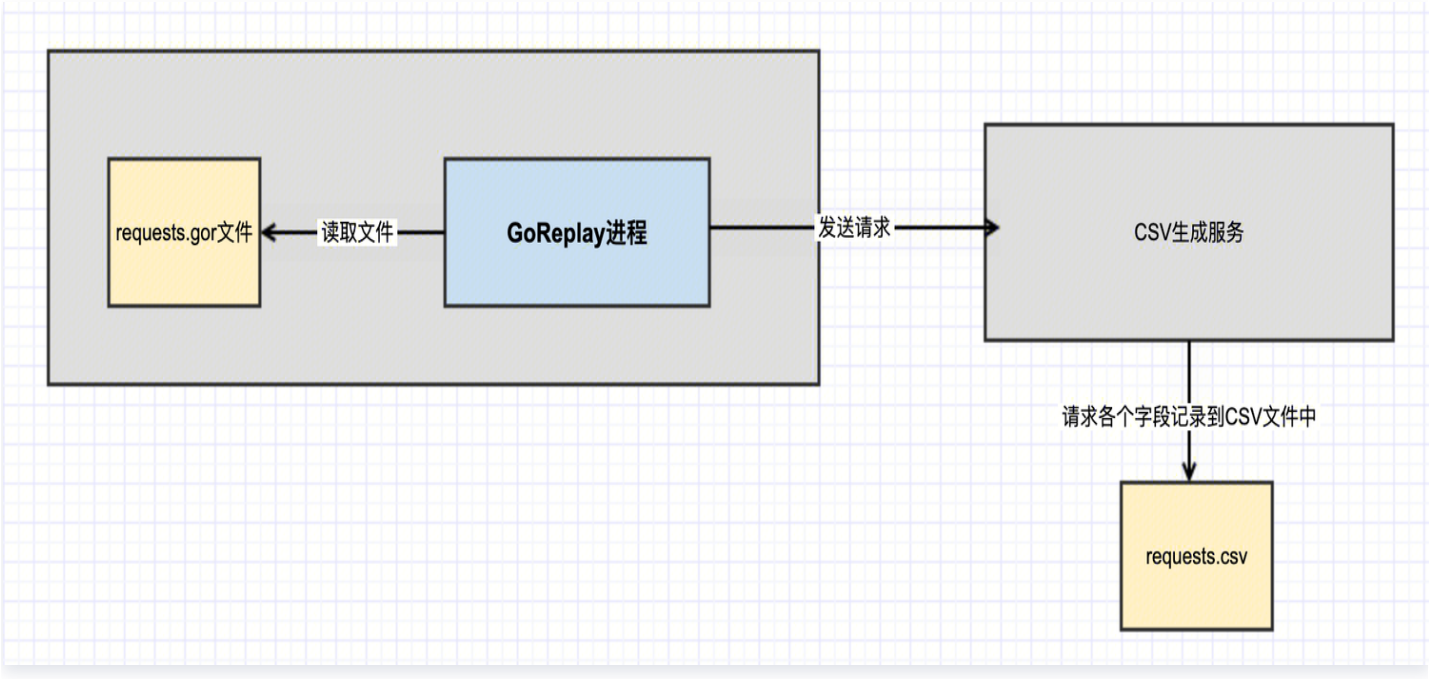
```
sudo gor --input-raw :80 --output-file requests.gor
```

这个命令会捕获所有通过端口80的流量，并将其保存到当前目录下的 requests.gor 文件中。

> ⚠ **注意：**
> 需要 sudo 权限来监听80端口。

将 Gor 文件转换成 CSV 参数文件

本节使用 GoReplay 回放 Gor 文件中记录的请求到 CSV 生成服务。参与组件：GoReplay、CSV 生成服务。整体架构如下：



在 CSV 文件中会记录下请求各个字段，例如 scheme、host、uri、method、base64Body。下面是一个简单 CSV 文件示例：

scheme	host	uri	method	jsonHeaders	base64Body
http	mockhttpbin.pts.svc.cluster.local	/get?page=1	get	{"name":"kk"}	-
http	mockhttpbin.pts.svc.cluster.local	/post	post	{"Hello": 'world',}	dGhpcyBpcyBnb29kCg==

```
scheme,host,uri,method,jsonHeaders,base64Body
```

```
http,mockhttpbin.pts.svc.cluster.local,/get?page=1,get,{"name":"kk"},
http,mockhttpbin.pts.svc.cluster.local,/post,post,
{"hello":"world"},dGhpcyBpcyBnb29kCg==
```

说明：
使用 base64Body，而不是直接记录 body 是因为有些请求的 body 发送的二进制文件，直接写入 CSV 文件会展示成乱码。写成 base64 后，方便后续转换成原来的结构体。

CSV 生成服务代码

编写服务代码，并保存为 main.go 文件。

```go
package main

import (
    "encoding/base64"
    "encoding/csv"
    "encoding/json"
    "fmt"
    "io/ioutil"
    "log"
    "net/http"
    "os"
)

func main() {
    http.HandleFunc("/", requestHandler) // 设置处理函数
    log.Println("Server starting on port 8080...")
    log.Fatal(http.ListenAndServe(":8080", nil))
}

func requestHandler(w http.ResponseWriter, r *http.Request) {
    // 获取请求信息
    scheme := "http" // 默认为http，因为Go的http包不支持直接获取scheme
    if r.TLS != nil {
        scheme = "https"
    }
    host := r.Host
    uri := r.RequestURI
    method := r.Method

    // 将headers转换为JSON格式
    headersJson, err := json.Marshal(r.Header)
    if err != nil {
```

```
        http.Error(w, "Error converting headers to JSON", http.StatusInternalServerError)
        return
    }

    // 读取请求体
    body, err := ioutil.ReadAll(r.Body)
    if err != nil {
        http.Error(w, "Error reading request body", http.StatusInternalServerError)
        return
    }
    defer r.Body.Close()

    // Base64编码请求体
    base64Body := base64.StdEncoding.EncodeToString(body)

    // 写入CSV文件
    record := []string{scheme, host, uri, method, string(headersJson), base64Body}
    err = writeToCSV(record)
    if err != nil {
        http.Error(w, "Error writing to CSV", http.StatusInternalServerError)
        return
    }

    // 发送响应
    fmt.Fprintf(w, "Request logged")
}

func writeToCSV(record []string) error {
    file, err := os.OpenFile("requests.csv", os.O_CREATE|os.O_WRONLY|os.O_APPEND,
0666)
    if err != nil {
        return err
    }
    defer file.Close()

    writer := csv.NewWriter(file)
    defer writer.Flush()

    return writer.Write(record)
}
```

编译并运行 CSV 生成服务

1. 将上述文件保存成 main.go 文件
2. 直接运行代码。

```
go run main.go
```

回放流量到 CSV 生成服务上，用来生成 CSV 文件。

```
gor --input-file requests.gor --output-http "http://csv-server-address:8080" --
http-original-host true
```

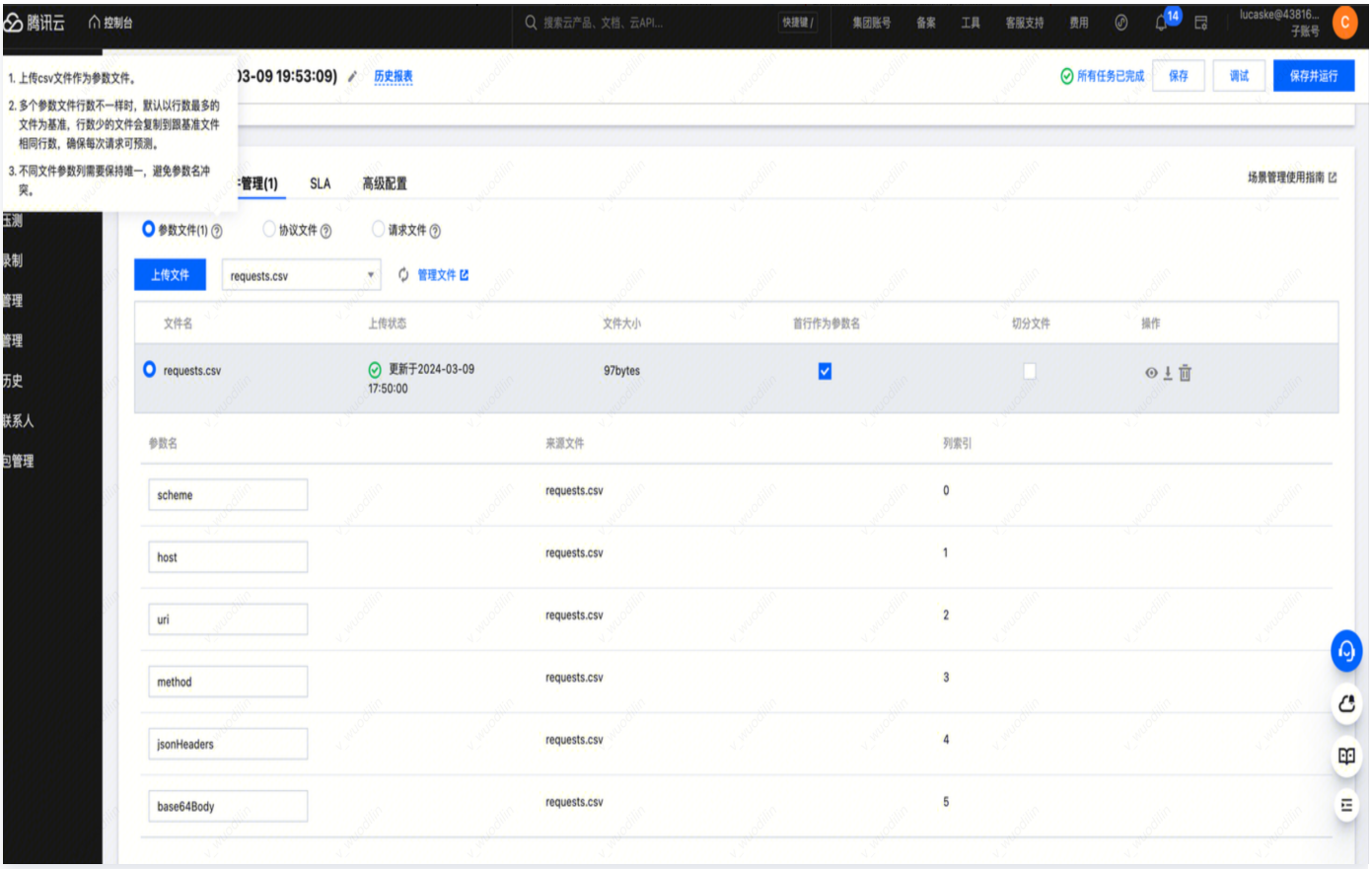
这个命令会读取 requests.gor 文件中的流量，并将其回放到CSV生成服务上，CSV 生成服务默认会将接收到的请求写成 requests.csv 文件里；且生成的流量 host 为请求原本的 host 而非 CSV 服务的地址。

在云压测上使用 CSV 参数文件回放请求

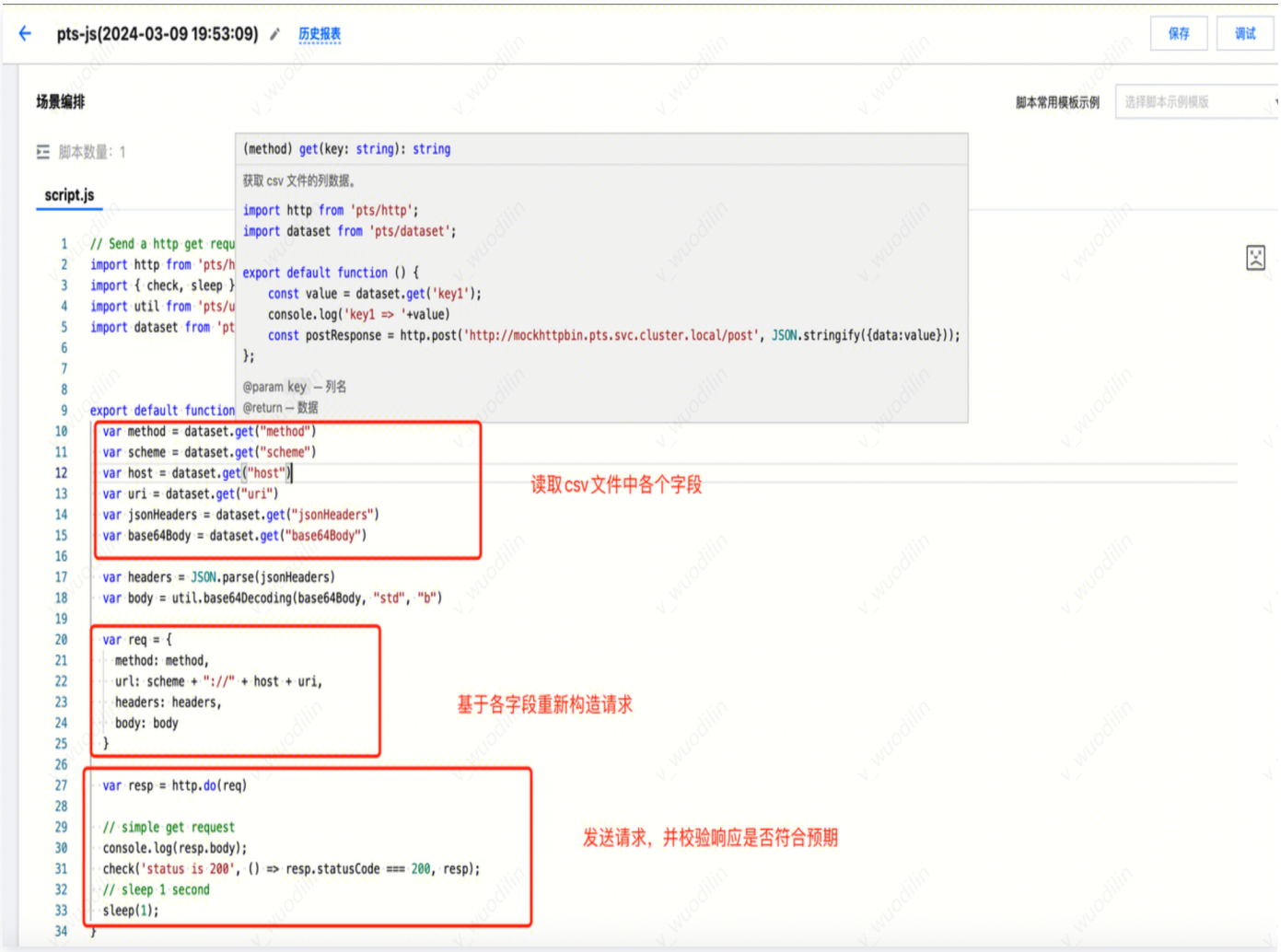
云压测支持用户上传 CSV 文件作为参数文件。您可以动态引用其中的测试数据，供脚本里的变量使用。这样，当施压机并发执行这段代码，每条请求能动态、逐行获取 CSV 里的每行数据，作为请求参数使用。参数文件具体用法可参见 使用参数文件 。

1. 登录 云压测控制台 ，云压测对于首次使用的用户提供一个免费的压测资源包。
2. 在左侧导航栏中选择**测试场景**，单击**新建场景**，选择**脚本模式**。

   云压测脚本模式支持原生 JavaScript ES2015(ES6)+ 语法，并提供额外函数，帮助您在脚本模式下，快速编排压测场景。您可在控制台的在线编辑器里，用 JavaScript 代码描述您的压测场景所需的请求编排、变量定义、结果断言、通用函数等逻辑。（详细的 API 文档请参见：PTS API ）。
3. 上传之前录制的 CSV 文件，作为参数文件。



4. 编写压测脚本，施压机每次执行压测脚本时候，读取 CSV 文件中下一行，利用 CSV 文件中记录的字段重新构造出原始请求。



压测脚本如下：

```
// Send a http get request
import http from 'pts/http';
import { check, sleep } from 'pts';
import util from 'pts/util';
import dataset from 'pts/dataset';


export default function () {
  // 读取csv文件各个字段
  var method = dataset.get("method")
  var scheme = dataset.get("scheme")
  var host = dataset.get("host")
  var uri = dataset.get("uri")
  var jsonHeaders = dataset.get("jsonHeaders")
  var base64Body = dataset.get("base64Body")
```

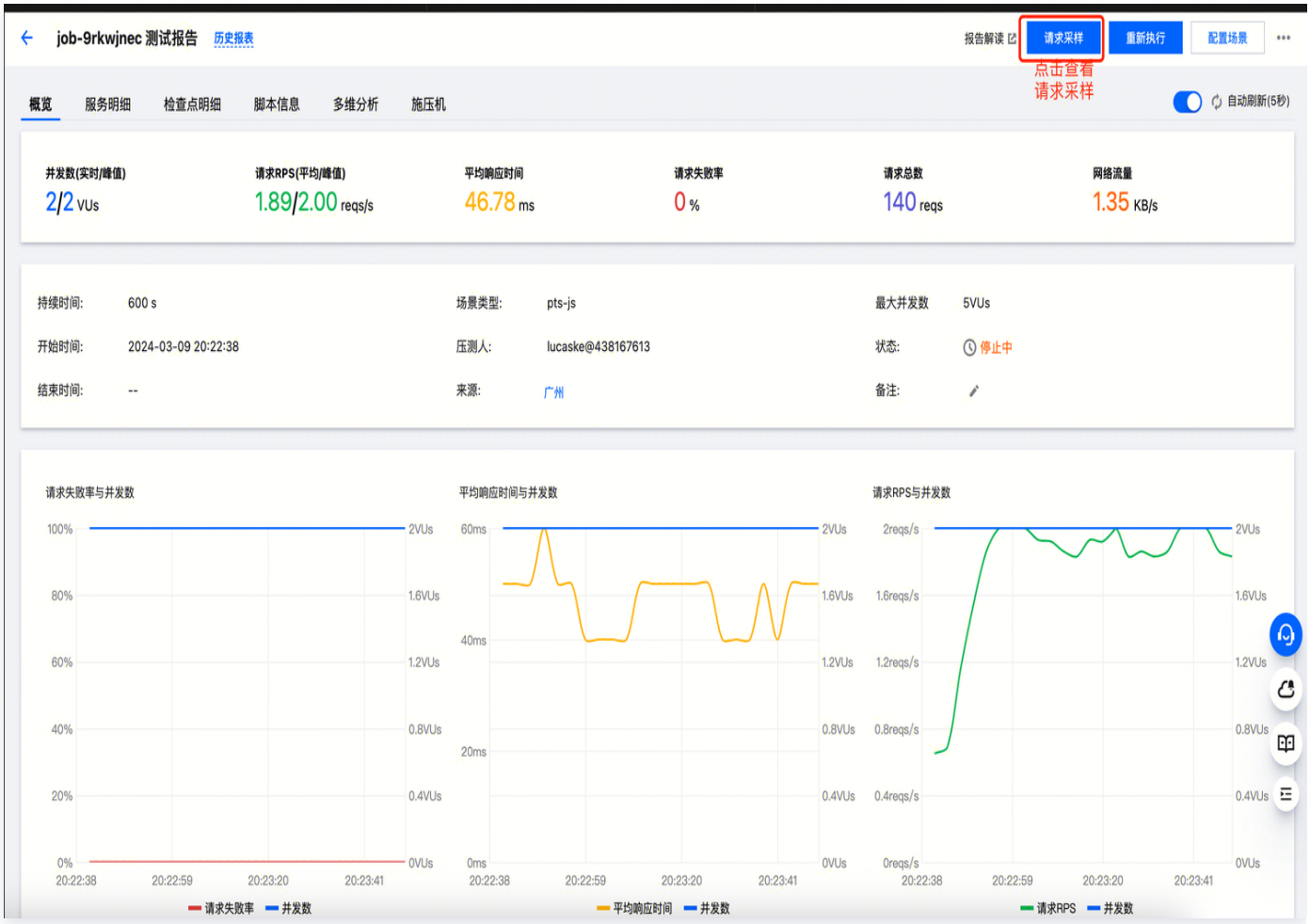
```
  var headers = JSON.parse(jsonHeaders)
  var body = util.base64Decoding(base64Body, "std", "b")

  // 构造请求
  var req = {
    method: method,
    url: scheme + "://" + host + uri,
    headers: headers,
    body: body
  }

  // 发送请求
  var resp = http.do(req)

  // simple get request
  console.log(resp.body);
  check('status is 200', () => resp.statusCode === 200, resp);
  // sleep 1 second
  sleep(1);
}
```

5. 点击**保存并运行**，即可运行压测脚本，回放流量。查看压测报告及**请求采样**，观察请求是否符合预期。


job-9rkwjnec 测试报告
历史报表
报告解读
请求采样
重新执行
配置场景
点击查看
请求采样
概览
服务明细
检查点明细
脚本信息
多维分析
施压机
自动刷新(5秒)
并发数(实时/峰值)
2/2 VUs
请求RPS(平均/峰值)
1.89/2.00 reqs/s
平均响应时间
46.78 ms
请求失败率
0 %
请求总数
140 reqs
网络流量
1.35 KB/s
持续时间:
600 s
场景类型:
pts-js
最大并发数
5VUs
开始时间:
2024-03-09 20:22:38
压测人:
lucaske@438167613
状态:
停止中
结束时间:
--
来源:
广州
备注:
请求失败率与并发数
平均响应时间与并发数
请求RPS与并发数
请求失败率
并发数
平均响应时间
请求RPS

请求采样：


POST http://mockhttpbin.pts.svc.cluster.local/post
200 ok
总耗时 100% 23.49ms
排队等待连接 0% 0.01ms
发送请求 0% 0.03ms
等待响应 100% 23.41ms
下载响应 0% 0.05ms
0ms 5ms 10ms 15ms 20ms 25ms
请求内容
Method POST
URL http://mockhttpbin.pts.svc.cluster.local/post
Headers
X-Pts-Request-Id 48aa8a48-7083-4fe9-bee5-d80e8c316837
Hello world
Connection keep-alive
User-Agent PTSEngine
Accept */*
Body
this is good