使用Cloud Armor对谷歌云负载均衡限流

谷歌云全球负载均衡Google Cloud Load Balancing (GCLB) 作为一个全球分布式软件定义负载均衡，可以将全球各地的网络请求通过谷歌边缘节点和骨干网高效投递到用户在任一谷歌云数据中心部署的后端服务，是使用频率非常高的一个云产品。其工作模式为通过分布在全球的数百个边缘节点将用户请求接收并通过自建骨干网将这些请求投递到后端服务。GCLB在没有配置WAF的时候，会无条件接收所有外部的HTTP/HTTPS请求，而不做任何限流。但是，在某些场景下，比如流量增长高于后端承载能力，或者遭到CC攻击的时候，用户希望负载均衡能进行限流，在一段时间内减少发送到后端服务请求QPS，从而为后端服务扩容或者代码调整提供时间，通过放弃一部分请求来正常服务主要用户群体，并且防止后端在扩容时不必要的处理积压请求而阻塞。

针对这些场景，可以利用谷歌云应用防火墙Cloud Armor的Rate Limiting功能配合全球负载均衡来实现限流方案。本文通过一个操作实例来介绍这一方案。

创建负载均衡服务

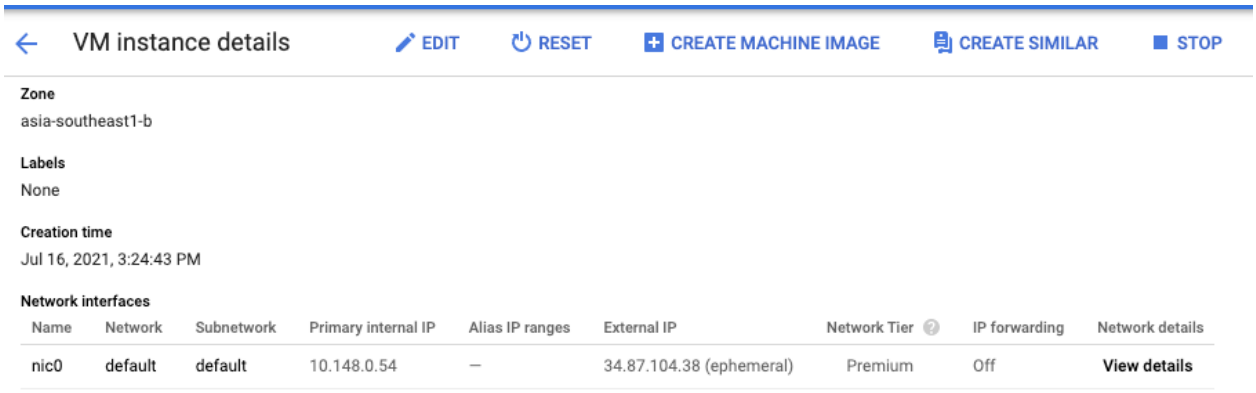
创建web服务实例组

首先在谷歌云新加坡区域创建一台虚机。我们会在这台虚机上搭建模拟HTTP服务器处理HTTP请求。然后以这台虚机为基准来创建一个服务端虚机模板，以供下面步骤来创建实例。如果读者要执行这些命令，请将项目ID、服务账号、虚机配置等参数值改成适用于读者自己的环境的值。

```
gcloud compute instances create mock-server-test --project=youzhi-lab
```

```
--zone=asia-southeast1-b --machine-type=n2-standard-2
--network-interface=network-tier=PREMIUM,subnet=default
--maintenance-policy=MIGRATE
--service-account=247839977271-compute@developer.gserviceaccount.com
--scopes=https://www.googleapis.com/auth/devstorage.read_only,https://www.g
oogleapis.com/auth/logging.write,https://www.googleapis.com/auth/monitoring
.write,https://www.googleapis.com/auth/servicecontrol,https://www.googleapi
s.com/auth/service.management.readonly,https://www.googleapis.com/auth/trac
e.append --tags=http-server,https-server
--image=ubuntu-1804-bionic-v20210702 --image-project=ubuntu-os-cloud
--boot-disk-size=100GB --no-boot-disk-auto-delete
--boot-disk-type=pd-balanced --boot-disk-device-name=mock-server-test-2
--no-shielded-secure-boot --shielded-vtpm --shielded-integrity-monitoring
--reservation-affinity=any
```

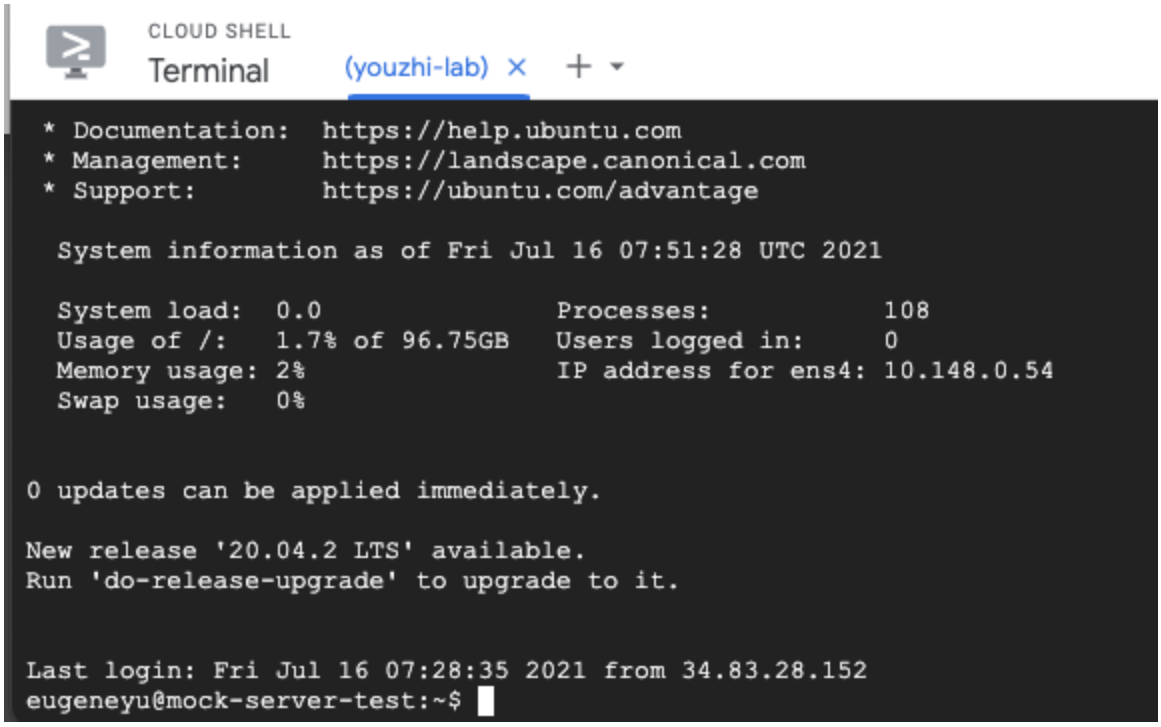
虚机创建成功后可以在控制台看到虚机状态以及分配的公网IP等详情。



可以通过公网IP登录到虚机，或者使用以下命令在Cloud Shell中或者安装了谷歌云SDK的本地环境登录这台虚机。

```
gcloud compute ssh mock-server-test --zone asia-southeast1-b
```

下图是从控制台Cloud Shell上登录虚机的示例。



登录之后，在虚机上执行以下命令，安装和配置Wiremock。Wiremock是一款模拟Web服务器的免费软件，可以用来做Web应用的功能和性能测试。

```
sudo apt update
sudo apt install -y default-jre
wget
https://repo1.maven.org/maven2/com/github/tomakehurst/wiremock-jre8-standal
one/2.29.0/wiremock-jre8-standalone-2.29.0.jar
mkdir mappings
cat >./mappings/post-and-get-test.json <<EOL
{
  "mappings": [
    {
      "request": {
        "method": "GET",
        "url": "/"
      },
      "response": {
        "status": 200,
        "body": "Welcome!\n"
      }
    },
    {
```

```
      "request": {
        "method": "POST",
        "url": "/api/post-test"
      },
      "response": {
        "status": 200,
        "body": "Received POST\n"
      }
    },
    {
      "request": {
        "method": "GET",
        "url": "/api/get-test"
      },
      "response": {
        "status": 200,
        "body": "Received GET\n"
      }
    }
  ]
}
EOL
```

下面的命令将Wiremock注册为开机自动启动的后台服务。

```
cat >./wiremock-start.sh <<EOL
#!/bin/bash
java -jar /usr/local/bin/wiremock-jre8-standalone-2.29.0.jar --port 80
EOL

chmod +x /home/eugeneyu/wiremock-start.sh

cat >./wiremock.service <<EOL
[Unit]
Description=Wiremock

[Service]
WorkingDirectory=/home/eugeneyu
ExecStart=/home/eugeneyu/wiremock-start.sh

[Install]
WantedBy=multi-user.target
```

```
EOL

cat wiremock.service |sudo tee /etc/systemd/system/wiremock.service

sudo chmod +x /etc/systemd/system/wiremock.service

sudo systemctl enable wiremock.service
```

现在在虚机上执行下面的命令，验证Wiremock服务工作正常。

```
curl -v http://127.0.0.1/
```

如果返回状态码不是200，可以用下面命令查一下服务启动是否出错。

```
journalctl -u wiremock.service
```

也可以在本地测试从外网访问虚机的Wiremock服务，确保防火墙没有阻挡。

```
[eugeneyu:~]$ curl -v -X POST http://34.87.104.38/api/post-test
*   Trying 34.87.104.38...
* TCP_NODELAY set
* Connected to 34.87.104.38 (34.87.104.38) port 80 (#0)
> POST /api/post-test HTTP/1.1
> Host: 34.87.104.38
> User-Agent: curl/7.64.1
> Accept: */*
>
< HTTP/1.1 200 OK
< Matched-Stub-Id: 2cc7edd9-6937-4f2e-9bb4-fc21beb47b24
< Vary: Accept-Encoding, User-Agent
< Transfer-Encoding: chunked
<
Received POST
* Connection #0 to host 34.87.104.38 left intact
* Closing connection 0
```

虚机上的服务工作正常后，创建磁盘镜像。

```
gcloud compute images create mock-server-template-image
--project=youzhi-lab --source-disk=mock-server-test
--source-disk-zone=asia-southeast1-b --storage-location=asia-southeast1
```

```
--force
```

基于磁盘镜像创建虚机实例模板。

```
gcloud beta compute --project=youzhi-lab instance-templates create mock-server-template
--machine-type=n2-standard-2 --network=projects/youzhi-lab/global/networks/default
--network-tier=PREMIUM --maintenance-policy=MIGRATE
--service-account=247839977271-compute@developer.gserviceaccount.com
--scopes=https://www.googleapis.com/auth/devstorage.read_only,https://www.googleapis.com/aut
h/logging.write,https://www.googleapis.com/auth/monitoring.write,https://www.googleapis.com/
auth/servicecontrol,https://www.googleapis.com/auth/service.management.readonly,https://www.
googleapis.com/auth/trace.append --tags=http-server,https-server
--image=mock-server-template-image --image-project=youzhi-lab --boot-disk-size=100GB
--boot-disk-type=pd-standard --boot-disk-device-name=mock-server-template
--no-shielded-secure-boot --shielded-vtpm --shielded-integrity-monitoring
--reservation-affinity=any
```

使用下面命令在新加坡区域创建自动伸缩实例组。本例实例组配置为最少2台，最大10台，伸缩指标为请求数量。

```
PROJECT_ID=youzhi-lab
REGION=asia-southeast1
HEALTHCHECK_NAME=mock-server-health-check
INSTANCE_GROUP_NAME=mock-server-mig-sin
INSTANCE_TEMPLATE_NAME=mock-server-template
INSTANCE_GROUP_ZONES=asia-southeast1-a,asia-southeast1-b,asia-southeast1-c
BACKEND_NAME=backend-service-wiremock
URLMAP_NAME=lb-wiremock
TARGET_PROXY_NAME=target-proxy-wiremock
FORWARD_RULE_NAME=forward-rule-wiremock
SECURITY_POLICY_NAME=sec-pol-rate-limit

gcloud compute health-checks create http $HEALTHCHECK_NAME
--project=$PROJECT_ID --port=80 --request-path=/ --proxy-header=NONE
--check-interval=5 --timeout=5 --unhealthy-threshold=2
--healthy-threshold=2

gcloud beta compute --project=$PROJECT_ID instance-groups managed create \
$INSTANCE_GROUP_NAME --base-instance-name=$INSTANCE_GROUP_NAME \
--template=$INSTANCE_TEMPLATE_NAME --size=2 --zones=$INSTANCE_GROUP_ZONES \
--instance-redistribution-type=PROACTIVE --health-check=$HEALTHCHECK_NAME \
--initial-delay=60
```

```
gcloud beta compute --project "$PROJECT_ID" instance-groups managed \
set-autoscaling "$INSTANCE_GROUP_NAME" --region "$REGION" \
--cool-down-period "60" --max-num-replicas "10" --min-num-replicas "2" \
--target-load-balancing-utilization "0.8" --mode "on"
```

创建全球负载均衡GCLB

我们现在创建一个7层全球负载均衡，并制定上面创建的新加坡区域实例组为唯一后端实例组。这样，负载均衡在全球100多个边缘节点接收到的用户请求，会被全量代理转发到新加的实例组进行处理。

```
gcloud compute backend-services create $BACKEND_NAME \
--protocol HTTP \
--health-checks $HEALTHCHECK_NAME \
--global \
--enable-cdn \
--enable-logging \
--timeout=60

gcloud compute backend-services add-backend $BACKEND_NAME \
--balancing-mode=RATE \
--max-rate-per-instance=50 \
--capacity-scaler=1 \
--instance-group=$INSTANCE_GROUP_NAME \
--instance-group-region=$REGION \
--global

gcloud compute url-maps create $URLMAP_NAME \
--default-service $BACKEND_NAME

gcloud compute target-http-proxies create $TARGET_PROXY_NAME \
--url-map $URLMAP_NAME

gcloud compute forwarding-rules create $FORWARD_RULE_NAME \
--global \
--target-http-proxy=$TARGET_PROXY_NAME \
--ports=80
```

创建预览限流防火墙规则

为了防止操作不熟练导致错误并影响生产业务，可以先创建配置一个预览的限流防火墙规则，并绑定到负载均衡。这样当流量超过阈值时，可以从日志里查看限流是否会起作用。注意预览规则需要带--preview参数。

下面的预览规则，是要对”/api/”路径的所有请求做限流。限流的阈值是每分钟1800个请求，即30 QPS。该规则的优先级是1000。

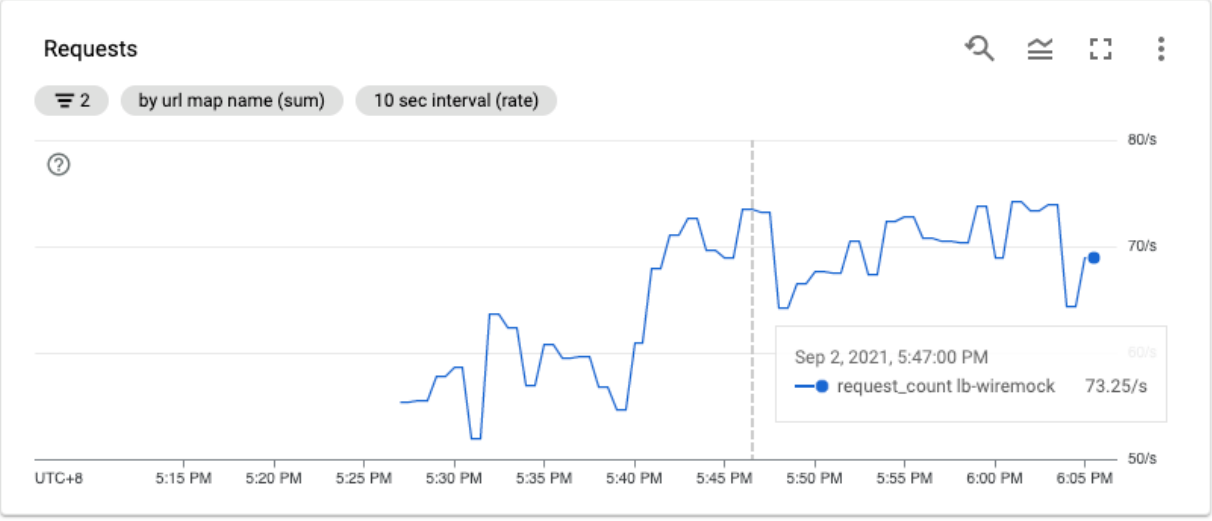
```
gcloud beta compute security-policies create $SECURITY_POLICY_NAME
--type=CLOUD_ARMOR

gcloud beta compute security-policies rules create 1000 \
    --security-policy $SECURITY_POLICY_NAME     \
    --expression "request.path.matches('/api/')" \
    --action "throttle"               \
    --rate-limit-threshold-count 1800         \
    --rate-limit-threshold-interval-sec 60  \
    --conform-action allow           \
    --exceed-action deny-429         \
    --enforce-on-key all             \
    --preview

gcloud beta compute backend-services update $BACKEND_NAME --security-policy
$SECURITY_POLICY_NAME --global
```

模拟请求并观察负载均衡全量转发请求

负载均衡创建好后，可以到控制台查看负载均衡前端IP。然后用以下命令模拟外部用户并发HTTP请求，将这些请求发送到负载均衡。

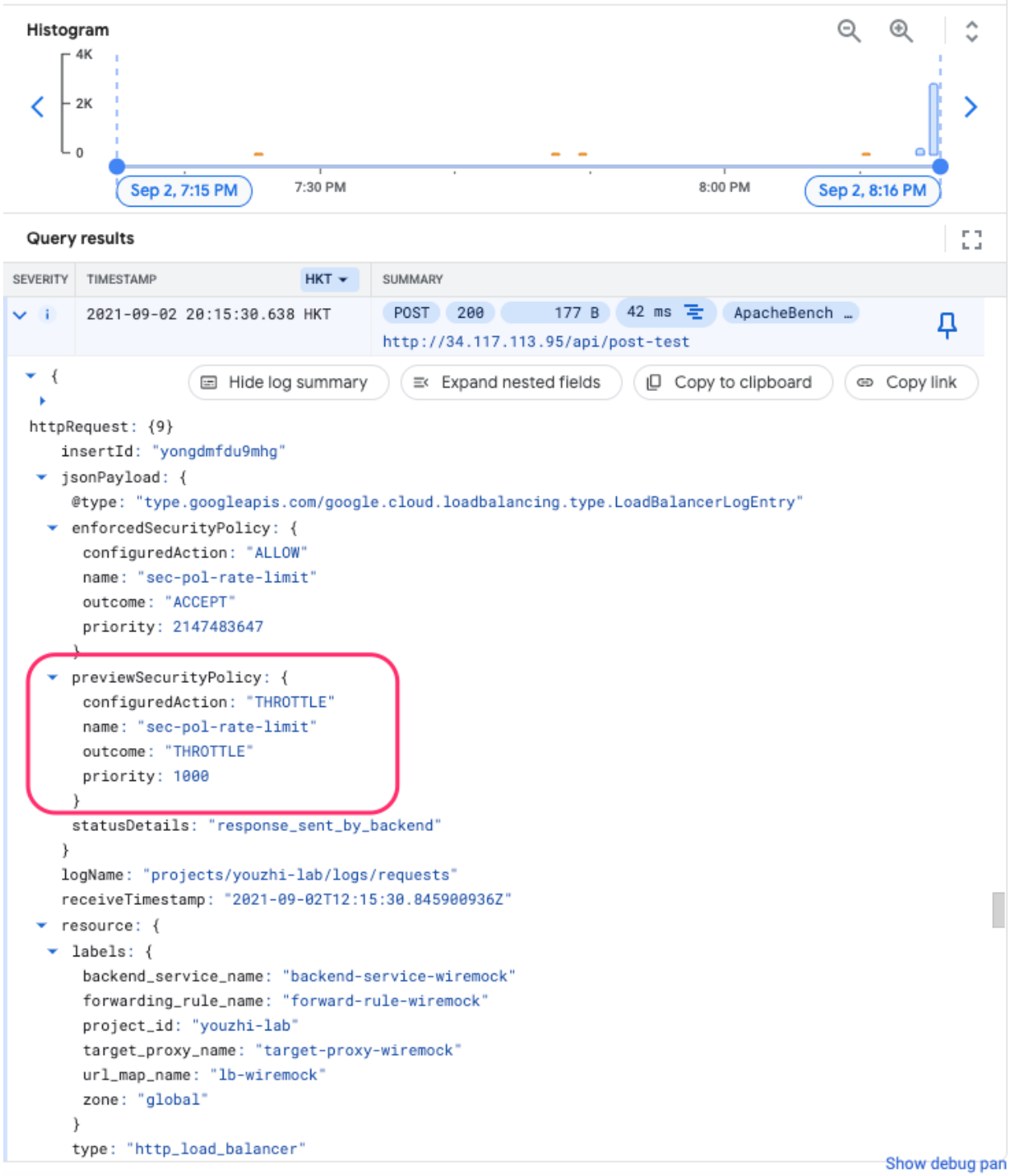
```
ab -n 200000 -c 15 -m POST -H "Content-Length:0" -v 3 \
http://34.117.113.95/api/post-test
```

从控制台监控可以查看负载均衡接收的请求数量。下图显示QPS大概在70左右。

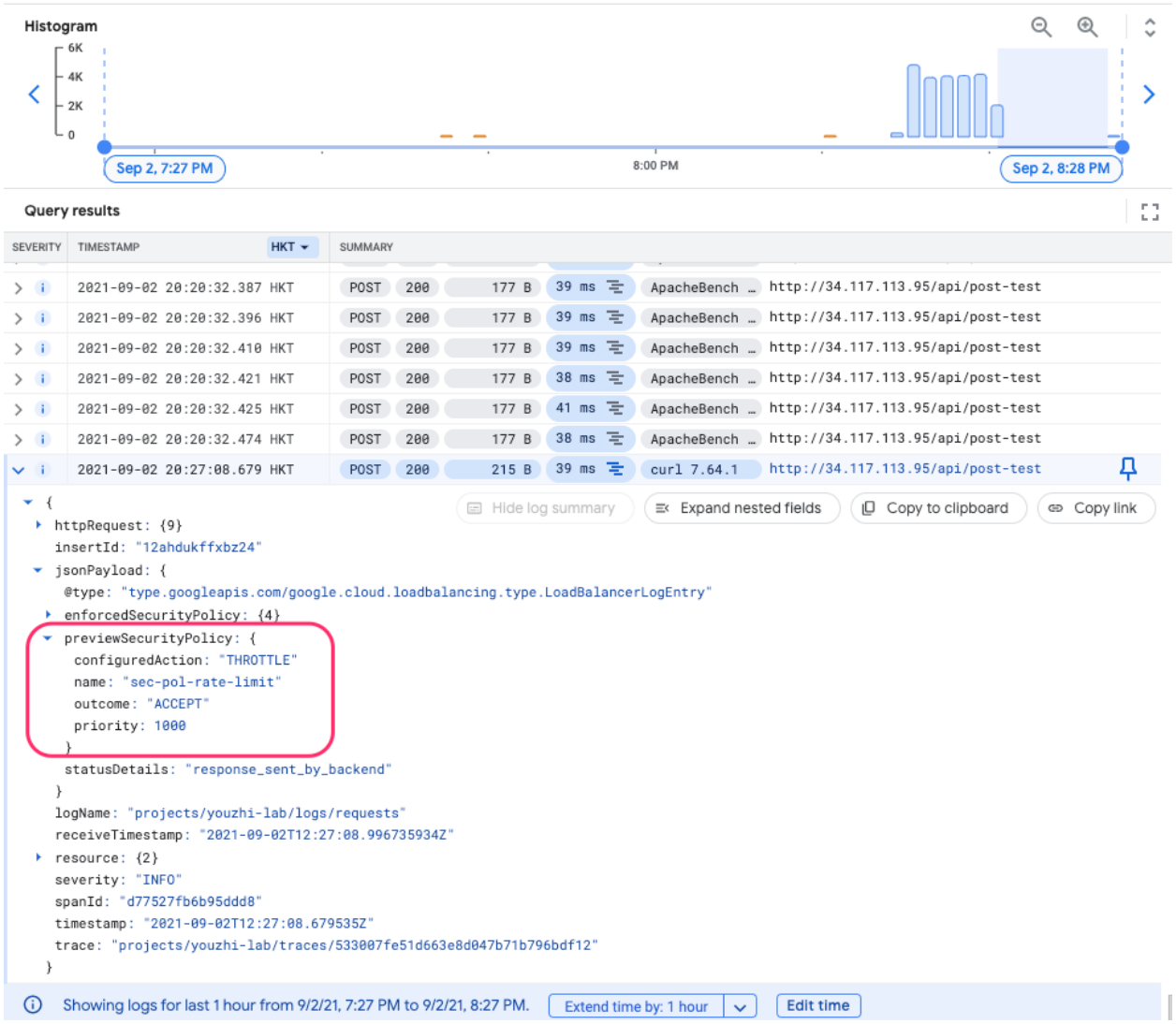


可以看出，目前负载均衡流量并没有被限制。这符合预期，因为负载均衡只配置了一个预览限流规则。

如果打开了负载均衡日志，可以在日志服务界面看到这个负载均衡的请求日志中，多了一个previewSecurityPlicy部分，其中outcome字段为”THROTTLE”的表示该请求命中限流规则，应该要被阻止（但实际上放行了）。



同时也有部分请求，其outcome字段为”ACCEPT”，表示该请求在限流范围以内，应该要被放行（实际上也放行了）。



创建正式限流防火墙规则

在确认预览防火墙规则配置无误后，可以配置正式防火墙规则，来实施流量限制。

注意下面的防火墙规则，跟之前的预览规则相比，去掉了--preview参数，规则优先级也改为了999，比预览规则优先级要高。

```
gcloud beta compute security-policies rules create 999 \
    --security-policy $SECURITY_POLICY_NAME     \
    --expression "request.path.matches('/api/')" \
    --action "throttle"                \
    --rate-limit-threshold-count 1800        \
```

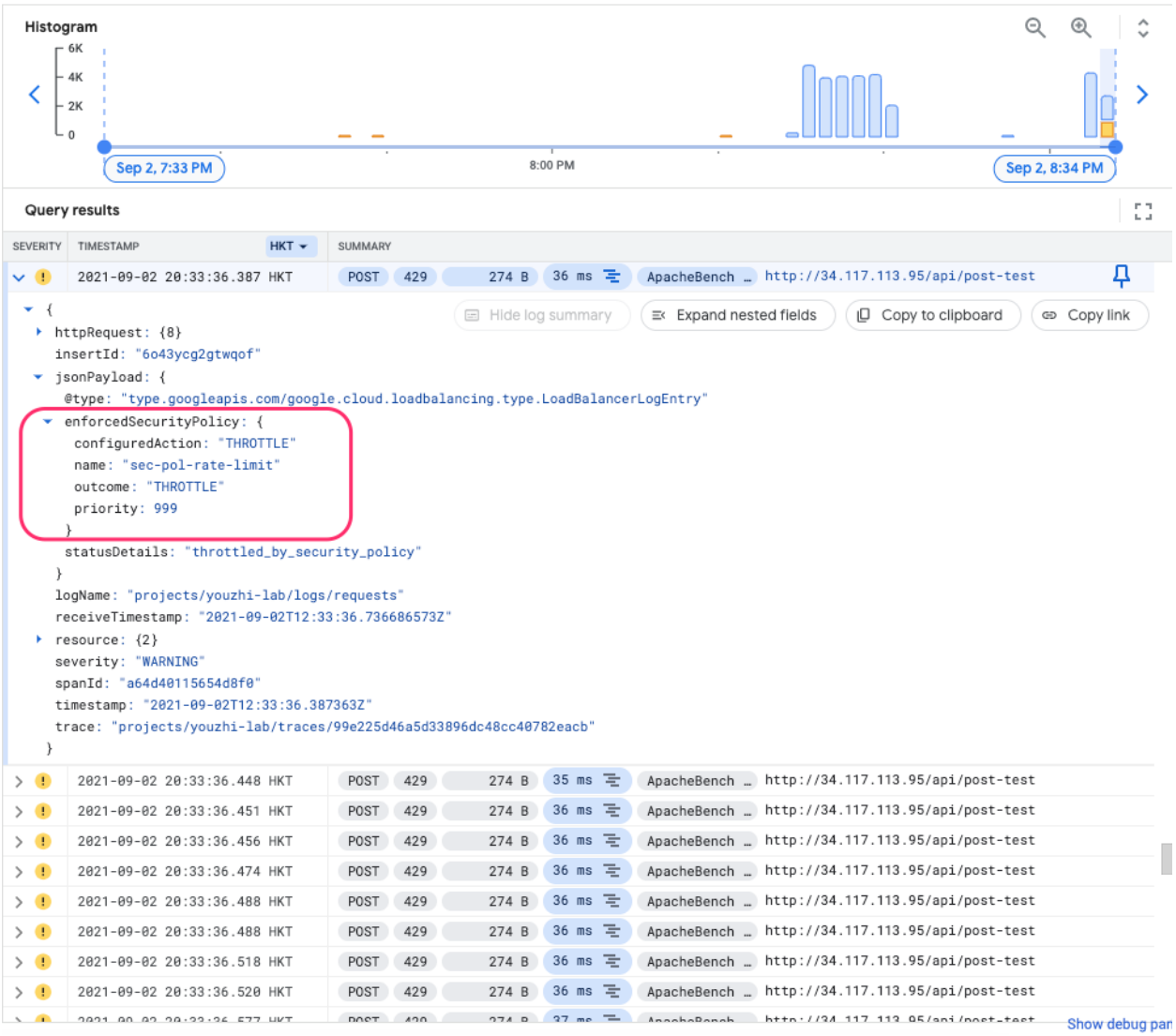
```
    --rate-limit-threshold-interval-sec 60  \
    --conform-action allow           \
    --exceed-action deny-429         \
    --enforce-on-key all
```

创建完规则后，很快会生效。此时可以再做流量测试观察效果。

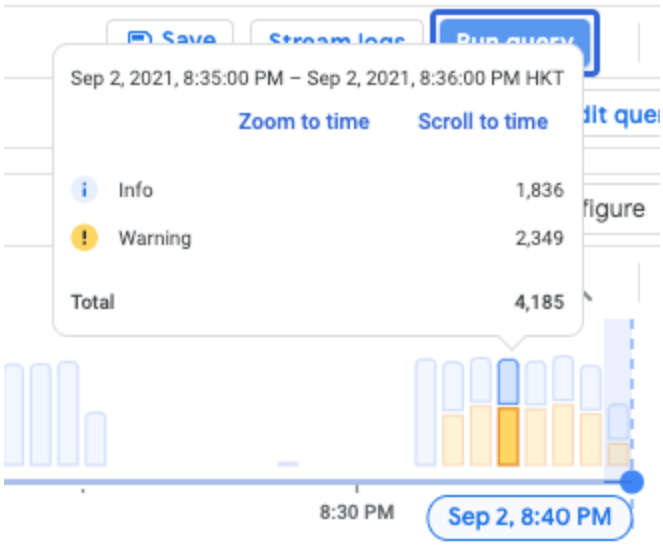
在本地机器执行下面流量测试命令。

```
ab -n 200000 -c 15 -m POST -H "Content-Length:0" -v 3 \
http://34.117.113.95/api/post-test
```

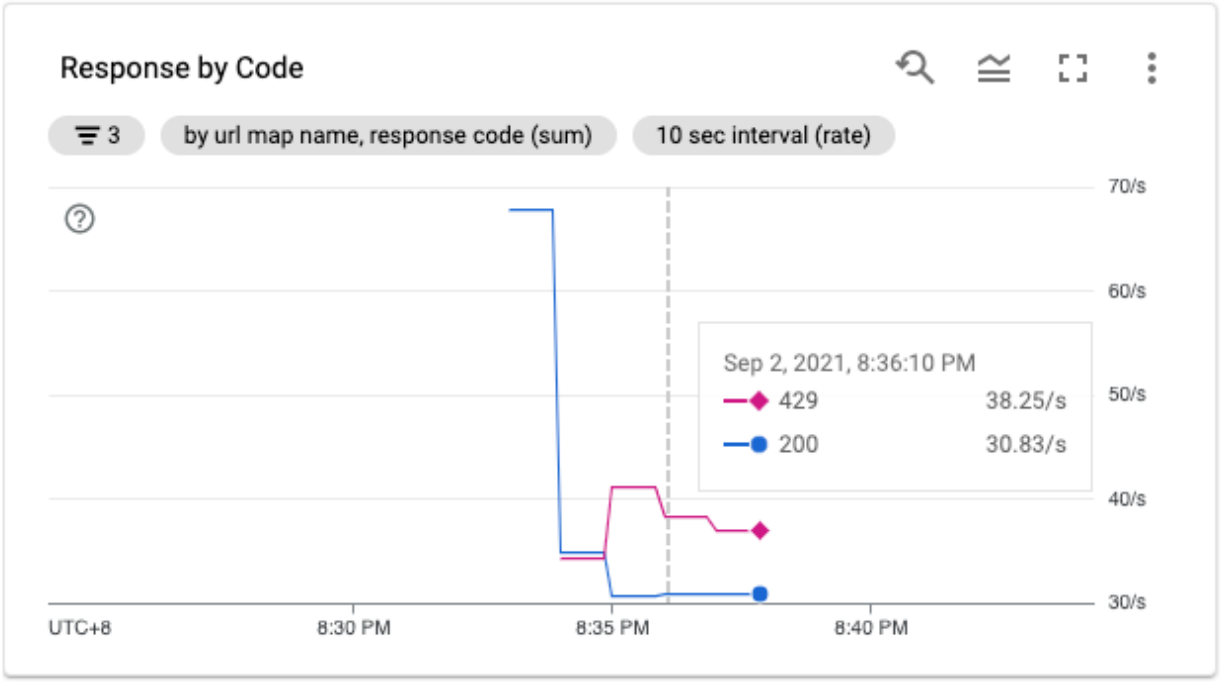
查看负载均衡日志，会看到有很多请求被阻止，返回了429 - Too Many Requests。原因是被优先级为999的防火墙规则限流。

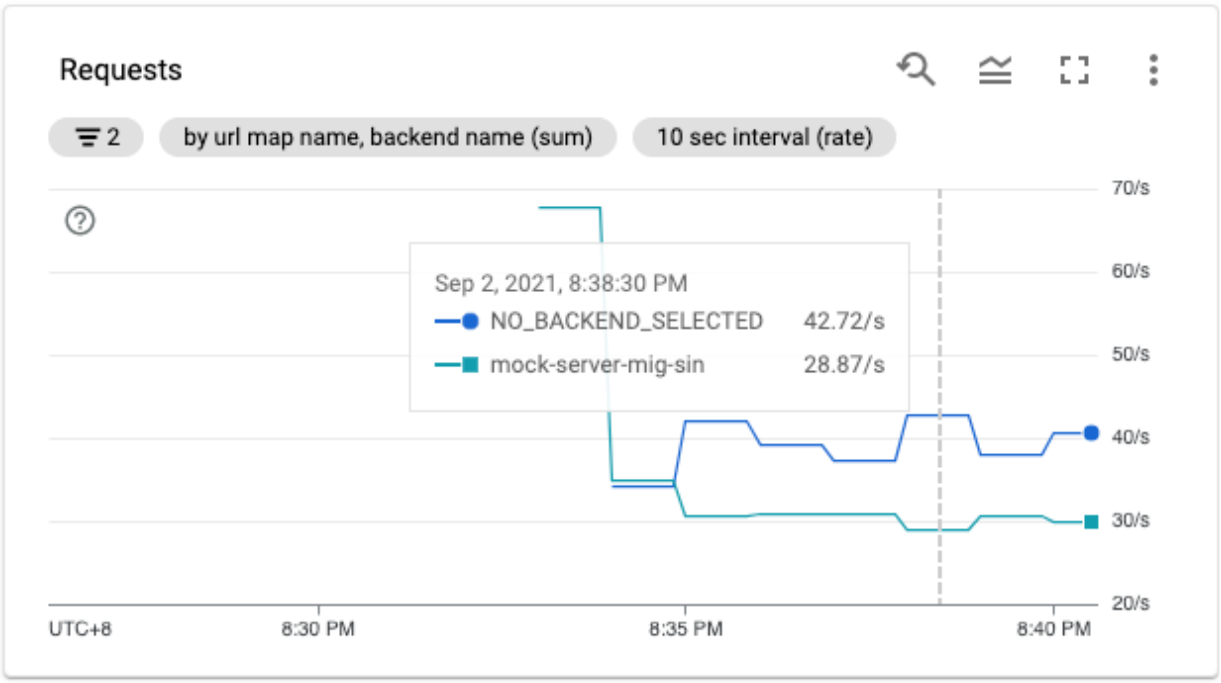


看日志统计柱状图可以发现，每分钟放行的请求为1800个左右，其余请求都被阻止。这与防火墙规则配置一致。



从负载均衡监控面板可以看到，总请求还是在70 QPS左右，但其中有30 QPS投递到了后端，并返回200。其余大概40 QPS被防火墙拦截并返回了429。





以上现象反映防火墙规则配置生效，并按照预期拦截了限流阈值之外的请求。

防火墙限流成本

Cloud Armor防火墙计费有两种方式。一种是按需付费的Standard Tier，一种是包月的Plus Tier。现在拿Standard Tier举例，如果是峰值为总请求1百万QPS，持续5小时，那么使用Cloud Armor规则限流的费用为

- WAF请求计费 - 0.75*1*5*3600 = 13,500
- WAF月租 - 5*1 = 5
- WAF规则月租 - 1*2 = 2 （本例为2条规则）

总费用为13,507美金。

参考文档

[1] Rate limiting overview
[2] Configuring rules for rate limiting

[3] Google Cloud Armor Pricing