APISIX 自建开发环境-测试汇总

测试什么

之前，已经对 APISIX 的功能逻辑，进行了相关的测试，主要测试的是 APISIX 本身的能力，并在测试过程中，思考其如何适配我们的场景。

然而，这些测试还不完善，实际上，针对微服务场景的测试，还有一些。

测试项

测试项	测试人员	测试目标	测试说明	测试结果	补充
性能	@束锋华	1.1 QPS	需要分别有几个并发数量级，比如： QPS 100、500、1000、3000 等		压测工具，建议使用 k6。参考：https://k6.io
	@任浩 @董红帅	1.2 Latency	测试延迟，测试网关转发时，自身的转发性能，Envoy 转发自身只消耗 1ms	结论： 1. APISIX 转发性能最强，大概 0.5ms	
Upstream 变化	@董红帅	2.1 Upstream 的动态变化	Upstream 的动态变化，可以从 dashboard 上看。	Istio-apisix-translator 已动态化调整，已测试。	
	@任浩	2.2 Upstream 动态生效速度	重要！！！	已测试，参考 2.3 结论	
	@束锋华	2.3 Upstream HTTP 无损热更新	测试当 Upstream 上游减少时，是否会引发 HTTP 5XX 问题。	结论： Upstream 变化不论实例增减，都是无损的。	
	@董红帅	2.4 Upstream gRPC 无损热更新	测试当 Upstream 上游减少时，是否会引发 gRPC 错误。 gRPC 是长连接，所以测试的长连接场景。测试时，必须保证至少有 1 个 Upstream 服务。	结论： 1. Upstream 变化生效几乎是接近实时的。 2. Upstream 变化时，压测	

				过程是无损的。	
Route 变化		3.1 规则动态变化		Istio-apisix-translator 已动态化 apisix 中微服务的流量。	
	@董红帅	3.2 规则生效速度	重要！！！！	结论： 1. 压测过程看，访问规则的变化是实时的。	
	@董红帅	3.3.1 规则变化 HTTP 无损—新规则权重高于旧规则场景	测试规则变化，不会引发 HTTP 5XX 4XX 问题	3.2 场景的测试，就是新规则权重高于旧规则。 结论：不会引发 4XX、5XX。	
	@董红帅	3.3.2 双规则，新规则高于旧规则，但删除新规则	测试规则变化，不会引发 HTTP 5XX 4XX 问题	结论 1. 不会引发 4XX、5XX 问题。	
	@董红帅	3.4 规则变化 gRPC 无损	测试规则变化，不会引发长连接场景出错		

测试结果

1.1 QPS 测试

> 使用 k6 压测
>
> 上游服务：beehivefe.mservice.svc.ab（10.72.16.97）

1. 固定QPS压测

压测条件：

a. 压测 30s

b. 不经过网关：直接压 Pod 容器，Pod 为 Nginx，4核4G（work_connections = 65535, worker_processes = 4）

c. 经过 APISIX：机器为 8核 8G。
d. Avg 耗时取 k6 的 http_req_duration 字段
e. 并发用户数 10 - 30000
f. Pod 和 APISIX 不同机器

QPS	1w	2w	3w	4w	5w	6w	7w	8w	9w	10w	11w	12w
吞吐量（不经过网关） 平均耗时	9986 194.85μs	19904 215.22μs	29538 210.32μs	38472 216.46μs	47376 204.49μs	55821 455.98μs	62571 571.74μs	69425 769.49μs	74051 1.37ms	77213 3.63ms	78512 7.21ms	76389 10.53ms
吞吐量（经过apisix网关） 平均耗时	9983 660.13μs	19885 623.08μs	29296 627.57μs	38545 636.39μs	47120 718.4μs	54680 2.93ms	60132 7.49ms	65268 12.33ms	65556 25.81ms	62498 46.35ms	63620 50.88ms	61719 51.17ms

从数据上看，QPS到11w的时候，原始服务基本达到吞吐量的峰值，但是耗时已经明显升高，9w左右的时候耗时和吞吐是一个比较好的状态

经过网关的情况下，8w-9w的时候，服务吞吐量已经到达峰值，耗时比原始服务要增加很多，比较优的吞吐和耗时是在5w-6w左右的时候

有一点值得注意的是，在12w QPS请求的情况下，apisix网关的负载也没有到达最大（80%左右），原始服务的负载也只有40%左右

2. 固定并发压测 1

附：只模拟用户数进行压测

压测条件：

a. 压测 30s
b. 不经过网关：直接压 Pod 容器，Pod 为 Nginx，4核4G
c. 经过 APISIX：机器为 8核 8G。
d. Avg 耗时取 k6 的 iteration_duration 字段。
e. 每项测试 5 次。
f. Pod 和 apisix 不同机器

压测结果：

并发用户数	100	200	500	1000
不经过网关	QPS：92552	QPS：89246	QPS：86296	QPS：81811

	avg：1.06ms	avg：2.2ms	avg：5.39ms	avg：11.97ms
经过 APISIX	QPS：54931 avg：3.62ms	QPS：60583 avg：3.28ms	QPS：65994 avg：avg=7.82ms	QPS：74992 avg：13.5ms

3. 固定并发压测 2

附：只模拟用户数进行压测

压测条件：

1. 压测 30s
2. 不经过网关：直接压 Pod 容器，Pod 为 Nginx，1核1G
3. 经过 APISIX：机器为 8核 8G。
4. Avg 耗时取 k6 的 iteration_duration 字段。
5. 每项测试 5 次。
6. Pod 和 APISIX 同机器，抹除 APISIX 到 Pod 之间的网络影响。

压测结果1（APISIX 为 8核）：

并发用户数	100	200	500	1000
不经过网关	QPS：92552 avg：1.06ms	QPS：89246 avg：2.2ms	QPS：86296 avg：5.39ms	
经过 APISIX	QPS：54931 avg：3.62ms	QPS：72765 avg：2.73ms	QPS：65994 avg：avg=7.82ms	

> 当机器为 8 核时，Nginx 的各个 worker 进程 CPU 使用率在 80-90 之间波动。

压测结果2（APISIX 为 24核）：

并发用户数	100	200	500	1000
不经过网关	QPS：92552 avg：1.06ms	QPS：89246 avg：2.2ms	QPS：86296 avg：5.39ms	QPS：81811 avg：11.97ms
经过 APISIX	QPS：92163 avg：1.07ms	QPS：90121 avg：2.19ms	QPS：83507 avg：5.68ms 说明：当为 64 核时，QPS 依然不再增幅。	

1.2 Latency 测试

> 使用 k6 压测

①：安装 K6

在节点 10.72.8.68 上，安装 k6s（非 docker 方式），参考：https://github.com/grafana/k6#running-k6

②：在节点上，创建测试 js

JavaScript

```javascript
import http from 'k6/http';
export let options = {
  vus: 100, // 指定要同时运行的虚拟用户数量
  duration: '30s', // 指定测试运行的总持续时间
};
// default 默认函数
export default function () {
  // 标头
  let params = { headers: { 'Content-Type': 'application/json' } };

  var res=http.get("http://beehivebuild.mtech.svc.ab",params)
}
```

③：压测现有的 Envoy 网关

将 beehivebuild.mtech.svc.ab 指向到 开发环境微服务网关 `172.16.130.87`

Shell

```shell
vim /etc/hosts
# 添加如下内容
172.16.130.87 beehivebuild.mtech.svc.ab
```

验证 hosts 添加成功

Shell

```
[root@knode10-72-8-68 benchmark]# ping beehivebuild.mtech.svc.ab
PING beehivebuild.mtech.svc.ab (172.16.130.87) 56(84) bytes of data.
64 bytes from beehivebuild.mtech.svc.ab (172.16.130.87): icmp_seq=1 ttl=59 time=5.34 ms
64 bytes from beehivebuild.mtech.svc.ab (172.16.130.87): icmp_seq=2 ttl=59 time=6.82 ms
```

开始压测微服务网关

Shell

```
k6 run get_beehivebuild.js
```

结果如下：

Shell

```
[root@knode10-72-8-68 benchmark]# k6 run get_beehivebuild.js

          /\      |‾‾| /‾‾/   /‾‾/
     /\  /  \     |  |/  /   /  /
    /  \/    \    |     (   /   ‾‾\
   /          \   |  |\  \ |  (‾)  |
  / __________ \  |__| \__\ \_____/ .io

  execution: local
     script: get_beehivebuild.js
     output: -

  scenarios: (100.00%) 1 scenario, 100 max VUs, 1m0s max duration (incl. graceful stop):
           * default: 100 looping VUs for 30s (gracefulStop: 30s)


running (0m30.0s), 000/100 VUs, 182937 complete and 0 interrupted iterations
default ✓ [======================================] 100 VUs  30s

     data_received..................: 32 MB  1.1 MB/s
     data_sent......................: 23 MB  750 kB/s
     http_req_blocked...............: avg=6.24µs  min=1.03µs  med=2.51µs  max=7.09ms   p(90)=4.13µs  p(95)=4.6µs
     http_req_connecting............: avg=3.13µs  min=0s      med=0s      max=6.02ms   p(90)=0s      p(95)=0s
     http_req_duration..............: avg=16.29ms min=10.02ms med=12.05ms max=672.75ms p(90)=16.97ms p(95)=19.92ms
```

```
25        { expected_response:true }...: avg=16.29ms min=10.02ms med=12.05ms
    max=672.75ms p(90)=16.97ms p(95)=19.92ms
26      http_req_failed................: 0.00%  ✓ 0           ✗ 182937
27      http_req_receiving.............: avg=39.58µs min=8.97µs  med=32.45µs
    max=11.77ms  p(90)=56.91µs p(95)=68.55µs
28      http_req_sending...............: avg=14.63µs min=5.02µs  med=11.27µs
    max=5.71ms   p(90)=19.49µs p(95)=28.28µs
29      http_req_tls_handshaking.......: avg=0s      min=0s      med=0s      max=0s
    p(90)=0s      p(95)=0s
30      http_req_waiting...............: avg=16.24ms min=9.97ms  med=11.99ms
    max=672.67ms p(90)=16.92ms p(95)=19.86ms
31      http_reqs......................: 182937 6095.029262/s
32      iteration_duration.............: avg=16.39ms min=10.07ms med=12.14ms
    max=672.89ms p(90)=17.07ms p(95)=20.02ms
33      iterations.....................: 182937 6095.029262/s
34      vus............................: 100    min=100       max=100
35      vus_max........................: 100    min=100       max=100
```

> 压测总结：
>
> 1. QPS：5500（3次取平均）

④：压测 APISIX

将 `beehivebuild.mtech.svc.ab` 指向到 APISIX 后，压测 APISIX

Shell

```
1  vim /etc/hosts
2  # 改为如下内容
3  10.72.8.62 beehivebuild.mtech.svc.ab
```

验证访问正确性：

Shell

```
[root@knode10-72-8-68 benchmark]# curl -v -H "Host: beehivebuild.mtech.svc.ab" "http://10.72.8.62"
* About to connect() to 10.72.8.62 port 80 (#0)
*   Trying 10.72.8.62...
* Connected to 10.72.8.62 (10.72.8.62) port 80 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.29.0
> Accept: */*
> Host: beehivebuild.mtech.svc.ab
>
< HTTP/1.1 200 OK
< Content-Type: text/plain; charset=utf-8
< Content-Length: 2
< Connection: keep-alive
< Date: Mon, 28 Feb 2022 09:46:34 GMT
< Server: APISIX/2.11.0
<
* Connection #0 to host 10.72.8.62 left intact
ok[root@knode10-72-8-68 benchmark]#
```

开始压测：

Shell

```
[root@knode10-72-8-68 benchmark]# k6 run get_beehivebuild.js

          /\      |‾‾| /‾‾/   /‾‾/
     /\  /  \     |  |/  /   /  /
    /  \/    \    |     (   /   ‾‾\
   /          \   |  |\  \ |  (‾)  |
  / __________ \  |__| \__\ \_____/ .io

  execution: local
     script: get_beehivebuild.js
     output: -

  scenarios: (100.00%) 1 scenario, 100 max VUs, 1m0s max duration (incl. graceful stop):
           * default: 100 looping VUs for 30s (gracefulStop: 30s)


running (0m30.0s), 000/100 VUs, 251396 complete and 0 interrupted iterations
default ✓ [======================================] 100 VUs  30s

     data_received..................: 42 MB  1.4 MB/s
     data_sent                      : 31 MB  1.0 MB/s
```

```
21      data_sent.....................: 31 MB  1.0 MB/s
22      http_req_blocked...............: avg=8.17µs  min=929ns    med=2.63µs
    max=14.68ms  p(90)=4.2µs   p(95)=5.63µs
23      http_req_connecting............: avg=3.71µs  min=0s       med=0s
    max=14.22ms  p(90)=0s      p(95)=0s
24      http_req_duration..............: avg=11.8ms  min=476.2µs  med=2.15ms
    max=115.96ms p(90)=62.22ms p(95)=71.68ms
25        { expected_response:true }...: avg=11.8ms  min=476.2µs  med=2.15ms
    max=115.96ms p(90)=62.22ms p(95)=71.68ms
26      http_req_failed................: 0.00%  ✓ 0           ✗ 251396
27      http_req_receiving.............: avg=47.36µs min=9.19µs   med=30.21µs
    max=18.42ms  p(90)=55.64µs p(95)=76.38µs
28      http_req_sending...............: avg=19.61µs min=3.8µs    med=11.93µs
    max=16.78ms  p(90)=21.64µs p(95)=36.95µs
29      http_req_tls_handshaking.......: avg=0s      min=0s       med=0s
    max=0s       p(90)=0s      p(95)=0s
30      http_req_waiting...............: avg=11.74ms min=441.17µs med=2.1ms
    max=115.86ms p(90)=62.15ms p(95)=71.62ms
31      http_reqs......................: 251396 8374.500947/s
32      iteration_duration.............: avg=11.92ms min=525.53µs med=2.26ms
    max=116.51ms p(90)=62.37ms p(95)=71.81ms
33      iterations.....................: 251396 8374.500947/s
34      vus............................: 100    min=100       max=100
35      vus_max........................: 100    min=100       max=100
```

> 压测结果：
>
> 1. QPS：8400（3次取平均）

⑤：直接压 Pod IP

Shell

```
1  import http from 'k6/http';
2  export let options = {
3    vus: 100, // 指定要同时运行的虚拟用户数量
4    duration: '30s', // 指定测试运行的总持续时间
5  };
6  // default 默认函数
7  export default function () {
8    // 标头
9    let params = { headers: { 'Content-Type': 'application/json' } };
10
11   var res=http.get("http://10.72.16.96",params)
12 }
```

Shell

```
1  [root@knode10-72-8-68 benchmark]# k6 run get_beehivebuild_pod.js
```

```

          /\      |‾‾| /‾‾/   /‾‾/
     /\  /  \     |  |/  /   /  /
    /  \/    \    |     (   /   ‾‾\
   /          \   |  |\  \ |  (‾)  |
  / __________ \  |__| \__\ \_____/ .io

  execution: local
     script: get_beehivebuild_pod.js
     output: -

  scenarios: (100.00%) 1 scenario, 100 max VUs, 1m0s max duration (incl. graceful stop):
           * default: 100 looping VUs for 30s (gracefulStop: 30s)


running (0m30.1s), 000/100 VUs, 261875 complete and 0 interrupted iterations
default ✓ [======================================] 100 VUs  30s

     data_received..................: 31 MB  1.0 MB/s
     data_sent......................: 29 MB  949 kB/s
     http_req_blocked...............: avg=4.79µs  min=993ns    med=2.32µs  max=19.39ms  p(90)=3.78µs  p(95)=5.05µs
     http_req_connecting............: avg=551ns   min=0s       med=0s      max=4.03ms   p(90)=0s      p(95)=0s
     http_req_duration..............: avg=11.33ms min=193.76µs med=1.3ms   max=207.27ms p(90)=76.37ms p(95)=87.61ms
       { expected_response:true }...: avg=11.33ms min=193.76µs med=1.3ms   max=207.27ms p(90)=76.37ms p(95)=87.61ms
     http_req_failed................: 0.00%  ✓ 0           ✗ 261875
     http_req_receiving.............: avg=51.58µs min=8.33µs   med=24.31µs max=24.17ms  p(90)=45.12µs p(95)=62.6µs
     http_req_sending...............: avg=21.13µs min=4.13µs   med=10.69µs max=21.49ms  p(90)=19.5µs  p(95)=29.64µs
     http_req_tls_handshaking.......: avg=0s      min=0s       med=0s      max=0s       p(90)=0s      p(95)=0s
     http_req_waiting...............: avg=11.26ms min=177.13µs med=1.25ms  max=207.22ms p(90)=76.28ms p(95)=87.55ms
     http_reqs......................: 261875 8706.389142/s
     iteration_duration.............: avg=11.46ms min=238.29µs med=1.4ms   max=207.38ms p(90)=76.66ms p(95)=87.73ms
     iterations.....................: 261875 8706.389142/s
     vus............................: 100    min=100       max=100
     vus_max........................: 100    min=100       max=100
```

> 压测结果：
>
> 1. QPS：8700（3次取平均）

1.2 总结

1、并发数 100，请求 30s

	Envoy【4C 8G】	APISIX【4C 8G】	Pod IP【1C 512M】
QPS	5500	8400	8700
性能损耗	36%	3%	-
平均耗时	16.39ms	11.92ms	11.46ms

2、并发 100，请求 20w 次

	Envoy【4C 8G】	APISIX【4C 8G】	Pod IP【1C 512M】
QPS	5631	8576	8917
平均耗时	avg=17.69ms	avg=11.44ms	avg=10.97ms
平均转发延迟	7ms	0.5ms	-

2.3 Upstream HTTP 无损热更新

①：测试条件

直接找一个 HTTP 服务，以 beehivebuild-mtech 为例。

测试当 Pod 由 2 个变为 1 个的时候，压测过程处是否有报错。

k6 脚本如下：

Shell

```
import http from 'k6/http';
export let options = {
  vus: 100, // 指定要同时运行的虚拟用户数量
  // duration: '30s', // 指定测试运行的总持续时间
  iterations: 200000, // 次数
};
// default 默认函数
export default function () {
  // 标头
  let params = { headers: { 'Content-Type': 'application/json' } };

  var res=http.get("http://beehivebuild.mtech.svc.ab",params)
}
```

k6 宿主机 hosts 如下：

Shell

```
10.72.8.63  beehivebuild.mtech.svc.ab
```

②：执行压测

Shell

```
k6 run get_beehivebuild.js
```

在压测过程中，去 aos 微服务中，缩容实例到 1 个。

③：分析结果：

Shell

```
[root@knode10-72-8-68 benchmark]# k6 run get_beehivebuild.js

          /\      |‾‾| /‾‾/   /‾‾/
     /\  /  \     |  |/  /   /  /
    /  \/    \    |     (   /   ‾‾\
   /          \   |  |\  \ |  (‾)  |
  / __________ \  |__| \__\ \_____/ .io

  execution: local
     script: get_beehivebuild.js
     output: -

  scenarios: (100.00%) 1 scenario, 100 max VUs, 10m30s max duration (incl.
```

```
    graceful stop):
14           * default: 200000 iterations shared among 100 VUs (maxDuration:
    10m0s, gracefulStop: 30s)
15
16
17  running (00m25.8s), 000/100 VUs, 200000 complete and 0 interrupted iterations
18  default ✓ [======================================] 100 VUs  00m25.8s/10m0s
    200000/200000 shared iters
19
20       data_received..................: 33 MB  1.3 MB/s
21       data_sent......................: 25 MB  953 kB/s
22       http_req_blocked...............: avg=9.88µs  min=834ns    med=2.5µs
    max=20.03ms  p(90)=3.99µs  p(95)=5.31µs
23       http_req_connecting............: avg=4.93µs  min=0s       med=0s
    max=19.3ms   p(90)=0s      p(95)=0s
24       http_req_duration..............: avg=12.76ms min=461.03µs med=1.28ms
    max=119.9ms  p(90)=77.34ms p(95)=84.73ms
25         { expected_response:true }...: avg=12.76ms min=461.03µs med=1.28ms
    max=119.9ms  p(90)=77.34ms p(95)=84.73ms
26       http_req_failed................: 0.00%  ✓ 0           ✗ 200000
27       http_req_receiving.............: avg=53.07µs min=9.24µs   med=28.33µs
    max=23ms     p(90)=54.84µs p(95)=80.8µs
28       http_req_sending...............: avg=20.63µs min=4.57µs   med=11.71µs
    max=21.83ms  p(90)=21.2µs  p(95)=39.31µs
29       http_req_tls_handshaking.......: avg=0s      min=0s       med=0s
    max=0s       p(90)=0s      p(95)=0s
30       http_req_waiting...............: avg=12.68ms min=433.37µs med=1.23ms
    max=119.4ms  p(90)=77.25ms p(95)=84.66ms
31       http_reqs......................: 200000 7745.007917/s
32       iteration_duration.............: avg=12.89ms min=513.37µs med=1.39ms
    max=120.08ms p(90)=77.56ms p(95)=84.85ms
33       iterations.....................: 200000 7745.007917/s
34       vus............................: 100    min=100       max=100
35       vus_max........................: 100    min=100       max=100
```

从结果来看，http_req_failed 为 0 ，说明，实例缩容，不会出现 4XX 5XX。

④：测试 Upstream 变化的生效速度

这个测试有 2 个场景：

1. Pod 新增场景
2. Pod 减少场景

> 分析：需要看 apisix 的日志，看 admin api 日志。
>
> 以 Pod 新增为例，当 apisix 开始出现新 Pod 的日志时，看 admin api 的 Upstream 规则变化是否也是在同一刻出现。

Pod 新增（IP 为：10.72.16.37）场景：

Apisix 日志：

```shell
10.72.8.68 - - [01/Mar/2022:17:31:10 +0800] beehivebuild.mtech.svc.ab "GET / HTTP/1.1" 200 2 0.082 "-" "k6/0.36.0 (https://k6.io/)" 10.72.16.96:80 200 0.083 "http://beehivebuild.mtech.svc.ab"
10.72.8.68 - - [01/Mar/2022:17:31:10 +0800] beehivebuild.mtech.svc.ab "GET / HTTP/1.1" 200 2 0.002 "-" "k6/0.36.0 (https://k6.io/)" 10.72.16.96:80 200 0.001 "http://beehivebuild.mtech.svc.ab"
10.72.8.68 - - [01/Mar/2022:17:31:10 +0800] beehivebuild.mtech.svc.ab "GET / HTTP/1.1" 200 2 0.002 "-" "k6/0.36.0 (https://k6.io/)" 10.72.16.96:80 200 0.001 "http://beehivebuild.mtech.svc.ab"
10.72.8.68 - - [01/Mar/2022:17:31:10 +0800] beehivebuild.mtech.svc.ab "GET / HTTP/1.1" 200 2 0.002 "-" "k6/0.36.0 (https://k6.io/)" 10.72.16.96:80 200 0.002 "http://beehivebuild.mtech.svc.ab"
10.72.8.68 - - [01/Mar/2022:17:31:10 +0800] beehivebuild.mtech.svc.ab "GET / HTTP/1.1" 200 2 0.000 "-" "k6/0.36.0 (https://k6.io/)" 10.72.16.96:80 200 0.001 "http://beehivebuild.mtech.svc.ab"
10.72.8.68 - - [01/Mar/2022:17:31:10 +0800] beehivebuild.mtech.svc.ab "GET / HTTP/1.1" 200 2 0.000 "-" "k6/0.36.0 (https://k6.io/)" 10.72.16.96:80 200 0.001 "http://beehivebuild.mtech.svc.ab"
10.72.8.68 - - [01/Mar/2022:17:31:10 +0800] beehivebuild.mtech.svc.ab "GET / HTTP/1.1" 200 2 0.004 "-" "k6/0.36.0 (https://k6.io/)" 10.72.16.96:80 200 0.004 "http://beehivebuild.mtech.svc.ab"
10.72.8.68 - - [01/Mar/2022:17:31:10 +0800] beehivebuild.mtech.svc.ab "GET / HTTP/1.1" 200 2 0.007 "-" "k6/0.36.0 (https://k6.io/)" 10.72.16.37:80 200 0.006 "http://beehivebuild.mtech.svc.ab"
10.72.8.68 - - [01/Mar/2022:17:31:10 +0800] beehivebuild.mtech.svc.ab "GET / HTTP/1.1" 200 2 0.005 "-" "k6/0.36.0 (https://k6.io/)" 10.72.16.37:80 200 0.005 "http://beehivebuild.mtech.svc.ab"
10.72.8.68 - - [01/Mar/2022:17:31:10 +0800] beehivebuild.mtech.svc.ab "GET / HTTP/1.1" 200 2 0.007 "-" "k6/0.36.0 (https://k6.io/)" 10.72.16.37:80 200 0.006 "http://beehivebuild.mtech.svc.ab"
10.72.8.68 - - [01/Mar/2022:17:31:10 +0800] beehivebuild.mtech.svc.ab "GET / HTTP/1.1" 200 2 0.005 "-" "k6/0.36.0 (https://k6.io/)" 10.72.16.37:80 200 0.005 "http://beehivebuild.mtech.svc.ab"
10.72.8.68 - - [01/Mar/2022:17:31:10 +0800] beehivebuild.mtech.svc.ab "GET / HTTP/1.1" 200 2 0.001 "-" "k6/0.36.0 (https://k6.io/)" 10.72.16.96:80 200 0.000 "http://beehivebuild.mtech.svc.ab"
```

Admin api 日志：

Shell

```
1  10.72.8.68 - - [01/Mar/2022:17:31:10 +0800] 10.72.8.62:9180 "PUT
   /apisix/admin/upstreams/microservice-beehive-beehivebuild-mtech-412398-50051
   HTTP/1.1" 200 589 0.022 "-" "go-resty/2.7.0 (https://github.com/go-resty/resty)"
   - - - "http://10.72.8.62:9180"
2  10.72.8.68 - - [01/Mar/2022:17:31:10 +0800] 10.72.8.62:9180 "PUT
   /apisix/admin/upstreams/microservice-beehive-beehivebuild-mtech-412398-80
   HTTP/1.1" 200 574 0.021 "-" "go-resty/2.7.0 (https://github.com/go-resty/resty)"
   - - - "http://10.72.8.62:9180"
```

可看出来，时间都是 `2022:17:31:10` ，可以认为，Upstream 的变化，是实时的。

> 同理：Pod 的减少。
>
> 但是，Pod 减少时，可能和 Pod 新增有出入。当 Pod 减少时，Upstream 规则已变化，但是此时，apisix 日志还会出现几秒已经删除的 Pod 的日志，这是因为，Upstream 变化的一刻，仍然有部分 apisix 的请求还没完成请求。
>
> 因此：验证这个最好的方式是，压测的 QPS 不要太高。尽可能的少，这样就排除了 Upstream 变化那一刻 apisix 仍然有大量请求发出的情况。

2.4 Upstream gRPC 无损热更新

①：前置条件

要测试 gRPC 无损更新，就得先有 gRPC Upstream。正好，原来在微服务中，创建过一个应用 grpctest-mtest。

创建一个 grpctest 的 Route，绑定到已有的 Upstream `microservice-beehive-grpctestgo-mtest-10003395-50051` 上即可。

因为已经在 apisix 50051 的端口上，开启了 http2 的支持，因此，50051 也就支持了 gRPC。

②：验证路由和服务的有效性

在本机 Mac 上，创建 gRPC test 的 Client。

Go

```go
package main

import (
    example "gitlab.mfwdev.com/mtest/grpctestgo/grpc"
    "log"
    "strconv"
    "time"

    "golang.org/x/net/context"
    "google.golang.org/grpc"
)

// 定义请求地址
const (
    ADDRESS string = "grpctestgo.mtest.svc.ab:50051"
)

// main 方法实现对 gRPC 接口的请求
func main() {
    conn, err := grpc.Dial(ADDRESS, grpc.WithInsecure())
    if err != nil {
        log.Fatalln("Can't connect: " + ADDRESS)
    }
    defer conn.Close()
    client := example.NewGreeterClient(conn)
    var total int
    for {
        total++
        resp, err := client.SayHello(context.Background(), &example.HelloRequest{Name: "request num: " + strconv.Itoa(total)})
        if err != nil {
            log.Fatalln("Do Format error:" + err.Error())
        }
        log.Println(resp.Message)
        time.Sleep(1 * time.Second)
    }
}
```

然后，将域名 `grpctestgo.mtest.svc.ab` 指向到 APISIX

Shell

```
10.72.8.62 grpctestgo.mtest.svc.ab
```

然后，测试

Shell

```
go build
./client
```

验证是否正确，首先是验证 client 输出正常

Groovy

```
$ ./client
2022/03/01 11:02:43 ok
2022/03/01 11:02:44 ok
2022/03/01 11:02:45 ok
2022/03/01 11:02:46 ok
2022/03/01 11:02:47 ok
2022/03/01 11:02:48 ok
2022/03/01 11:02:49 ok
2022/03/01 11:02:50 ok
2022/03/01 11:02:51 ok
2022/03/01 11:02:52 ok
2022/03/01 11:02:53 ok
2022/03/01 11:02:54 ok
2022/03/01 11:02:55 ok
2022/03/01 11:02:56 ok
2022/03/01 11:02:57 ok
2022/03/01 11:02:58 ok
2022/03/01 11:02:59 ok
2022/03/01 11:03:00 ok
2022/03/01 11:03:01 ok
2022/03/01 11:03:02 ok
```

然后，在 apisix 网关的日志中，可以看到相关日志

CSS

```
172.18.29.225 - - [01/Mar/2022:11:02:44 +0800] grpctestgo.mtest.svc.ab:50051 "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.028 "-" "grpc-go/1.42.0" 172.16.195.126:50051 200 0.028 "grpc://grpctestgo.mtest.svc.ab:50051"
172.18.29.225 - - [01/Mar/2022:11:02:45 +0800] grpctestgo.mtest.svc.ab:50051 "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0" 172.16.195.126:50051 200 0.007 "grpc://grpctestgo.mtest.svc.ab:50051"
172.18.29.225 - - [01/Mar/2022:11:02:46 +0800] grpctestgo.mtest.svc.ab:50051 "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0" 172.16.195.126:50051 200 0.007 "grpc://grpctestgo.mtest.svc.ab:50051"
172.18.29.225 - - [01/Mar/2022:11:02:47 +0800] grpctestgo.mtest.svc.ab:50051 "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0" 172.16.195.126:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
```

```
5  172.18.29.225 - - [01/Mar/2022:11:02:48 +0800] grpctestgo.mtest.svc.ab:50051
   "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0"
   172.16.195.126:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
6  172.18.29.225 - - [01/Mar/2022:11:02:49 +0800] grpctestgo.mtest.svc.ab:50051
   "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0"
   172.16.195.126:50051 200 0.007 "grpc://grpctestgo.mtest.svc.ab:50051"
7  172.18.29.225 - - [01/Mar/2022:11:02:50 +0800] grpctestgo.mtest.svc.ab:50051
   "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.006 "-" "grpc-go/1.42.0"
   172.16.195.126:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
8  172.18.29.225 - - [01/Mar/2022:11:02:51 +0800] grpctestgo.mtest.svc.ab:50051
   "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0"
   172.16.195.126:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
9  172.18.29.225 - - [01/Mar/2022:11:02:52 +0800] grpctestgo.mtest.svc.ab:50051
   "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.006 "-" "grpc-go/1.42.0"
   172.16.195.126:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
10 172.18.29.225 - - [01/Mar/2022:11:02:53 +0800] grpctestgo.mtest.svc.ab:50051
   "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0"
   172.16.195.126:50051 200 0.007 "grpc://grpctestgo.mtest.svc.ab:50051"
11 172.18.29.225 - - [01/Mar/2022:11:02:54 +0800] grpctestgo.mtest.svc.ab:50051
   "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.008 "-" "grpc-go/1.42.0"
   172.16.195.126:50051 200 0.008 "grpc://grpctestgo.mtest.svc.ab:50051"
12 172.18.29.225 - - [01/Mar/2022:11:02:55 +0800] grpctestgo.mtest.svc.ab:50051
   "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0"
   172.16.195.126:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
13 172.18.29.225 - - [01/Mar/2022:11:02:56 +0800] grpctestgo.mtest.svc.ab:50051
   "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0"
   172.16.195.126:50051 200 0.007 "grpc://grpctestgo.mtest.svc.ab:50051"
14 172.18.29.225 - - [01/Mar/2022:11:02:57 +0800] grpctestgo.mtest.svc.ab:50051
   "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0"
   172.16.195.126:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
15 172.18.29.225 - - [01/Mar/2022:11:02:58 +0800] grpctestgo.mtest.svc.ab:50051
   "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0"
   172.16.195.126:50051 200 0.007 "grpc://grpctestgo.mtest.svc.ab:50051"
16 172.18.29.225 - - [01/Mar/2022:11:03:00 +0800] grpctestgo.mtest.svc.ab:50051
   "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.006 "-" "grpc-go/1.42.0"
   172.16.195.126:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
17 172.18.29.225 - - [01/Mar/2022:11:03:01 +0800] grpctestgo.mtest.svc.ab:50051
   "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.006 "-" "grpc-go/1.42.0"
   172.16.195.126:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
18 172.18.29.225 - - [01/Mar/2022:11:03:02 +0800] grpctestgo.mtest.svc.ab:50051
   "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.006 "-" "grpc-go/1.42.0"
   172.16.195.126:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
19 172.18.29.225 - - [01/Mar/2022:11:03:03 +0800] grpctestgo.mtest.svc.ab:50051
   "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0"
   172.16.195.126:50051 200 0.007 "grpc://grpctestgo.mtest.svc.ab:50051"
20 172.18.29.225 - - [01/Mar/2022:11:03:04 +0800] grpctestgo.mtest.svc.ab:50051
   "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0"
   172.16.195.126:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
```

③：开始 gRPC 压测过程

```shell
./client
```

④：在 aos 微服务 `grpctestgo-mtest` 扩容到 2 个实例，从网关侧，观察生效速度

扩容场景1：

Shell

```
1  172.18.29.225 - - [01/Mar/2022:11:23:48 +0800] grpctestgo.mtest.svc.ab:50051
   "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.012 "-" "grpc-go/1.42.0"
   172.16.195.126:50051 200 0.012 "grpc://grpctestgo.mtest.svc.ab:50051"
2  172.18.29.225 - - [01/Mar/2022:11:23:49 +0800] grpctestgo.mtest.svc.ab:50051
   "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.012 "-" "grpc-go/1.42.0"
   172.16.195.28:50051 200 0.011 "grpc://grpctestgo.mtest.svc.ab:50051"
3  172.18.29.225 - - [01/Mar/2022:11:23:50 +0800] grpctestgo.mtest.svc.ab:50051
   "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0"
   172.16.195.126:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
4  172.18.29.225 - - [01/Mar/2022:11:23:51 +0800] grpctestgo.mtest.svc.ab:50051
   "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0"
   172.16.195.28:50051 200 0.007 "grpc://grpctestgo.mtest.svc.ab:50051"
5  172.18.29.225 - - [01/Mar/2022:11:23:52 +0800] grpctestgo.mtest.svc.ab:50051
   "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.006 "-" "grpc-go/1.42.0"
   172.16.195.126:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
6  172.18.29.225 - - [01/Mar/2022:11:23:53 +0800] grpctestgo.mtest.svc.ab:50051
   "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0"
   172.16.195.126:50051 200 0.007 "grpc://grpctestgo.mtest.svc.ab:50051"
7  10.72.8.68 - - [01/Mar/2022:11:23:51 +0800] 10.72.8.62:9180 "PUT
   /apisix/admin/upstreams/microservice-beehive-grpctestgo-mtest-10003395-50051
   HTTP/1.1" 200 567 0.020 "-" "go-resty/2.7.0 (https://github.com/go-resty/resty)"
   - - - "http://10.72.8.62:9180"
8  10.72.8.68 - - [01/Mar/2022:11:23:51 +0800] 10.72.8.62:9180 "PUT
   /apisix/admin/upstreams/microservice-beehive-grpctestgo-mtest-10003395-80
   HTTP/1.1" 200 555 0.017 "-" "go-resty/2.7.0 (https://github.com/go-resty/resty)"
   - - - "http://10.72.8.62:9180"
9  172.18.29.225 - - [01/Mar/2022:11:23:54 +0800] grpctestgo.mtest.svc.ab:50051
   "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0"
   172.16.195.126:50051 200 0.007 "grpc://grpctestgo.mtest.svc.ab:50051"
10 172.18.29.225 - - [01/Mar/2022:11:23:55 +0800] grpctestgo.mtest.svc.ab:50051
   "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.010 "-" "grpc-go/1.42.0"
   172.16.195.126:50051 200 0.010 "grpc://grpctestgo.mtest.svc.ab:50051"
11 172.18.29.225 - - [01/Mar/2022:11:23:56 +0800] grpctestgo.mtest.svc.ab:50051
   "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.010 "-" "grpc-go/1.42.0"
   172.16.195.126:50051 200 0.010 "grpc://grpctestgo.mtest.svc.ab:50051"
12 172.18.29.225 - - [01/Mar/2022:11:23:57 +0800] grpctestgo.mtest.svc.ab:50051
   "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0"
   172.16.195.126:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
13 172.18.29.225 - - [01/Mar/2022:11:23:58 +0800] grpctestgo.mtest.svc.ab:50051
   "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.006 "-" "grpc-go/1.42.0"
   172.16.195.126:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
```

> 结论：秒级

扩容场景2：

Shell

```
172.18.29.225 - - [01/Mar/2022:15:27:43 +0800] grpctestgo.mtest.svc.ab:50051 "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.010 "-" "grpc-go/1.42.0" 172.16.195.126:50051 200 0.010 "grpc://grpctestgo.mtest.svc.ab:50051"
172.18.29.225 - - [01/Mar/2022:15:27:43 +0800] grpctestgo.mtest.svc.ab:50051 "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.010 "-" "grpc-go/1.42.0" 172.16.195.126:50051 200 0.010 "grpc://grpctestgo.mtest.svc.ab:50051"
172.18.29.225 - - [01/Mar/2022:15:27:43 +0800] grpctestgo.mtest.svc.ab:50051 "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.010 "-" "grpc-go/1.42.0" 172.16.195.126:50051 200 0.010 "grpc://grpctestgo.mtest.svc.ab:50051"
172.18.29.225 - - [01/Mar/2022:15:27:43 +0800] grpctestgo.mtest.svc.ab:50051 "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.010 "-" "grpc-go/1.42.0" 172.16.195.126:50051 200 0.010 "grpc://grpctestgo.mtest.svc.ab:50051"
172.18.29.225 - - [01/Mar/2022:15:27:43 +0800] grpctestgo.mtest.svc.ab:50051 "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.010 "-" "grpc-go/1.42.0" 172.16.195.126:50051 200 0.010 "grpc://grpctestgo.mtest.svc.ab:50051"
172.18.29.225 - - [01/Mar/2022:15:27:43 +0800] grpctestgo.mtest.svc.ab:50051 "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.012 "-" "grpc-go/1.42.0" 172.16.195.28:50051 200 0.011 "grpc://grpctestgo.mtest.svc.ab:50051"
172.18.29.225 - - [01/Mar/2022:15:27:43 +0800] grpctestgo.mtest.svc.ab:50051 "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.012 "-" "grpc-go/1.42.0" 172.16.195.28:50051 200 0.011 "grpc://grpctestgo.mtest.svc.ab:50051"
172.18.29.225 - - [01/Mar/2022:15:27:43 +0800] grpctestgo.mtest.svc.ab:50051 "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0" 172.16.195.28:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
172.18.29.225 - - [01/Mar/2022:15:27:43 +0800] grpctestgo.mtest.svc.ab:50051 "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0" 172.16.195.126:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
10.72.8.68 - - [01/Mar/2022:15:27:44 +0800] 10.72.8.62:9180 "PUT /apisix/admin/upstreams/microservice-beehive-grpctestgo-mtest-10003395-50051 HTTP/1.1" 200 592 0.039 "-" "go-resty/2.7.0 (https://github.com/go-resty/resty)" - - - "http://10.72.8.62:9180"
```

> 结论：秒级

缩容场景：

Shell

```
172.18.29.225 - - [01/Mar/2022:15:31:25 +0800] grpctestgo.mtest.svc.ab:50051 "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.006 "-" "grpc-go/1.42.0" 172.16.195.28:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
172.18.29.225 - - [01/Mar/2022:15:31:25 +0800] grpctestgo.mtest.svc.ab:50051 "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.006 "-" "grpc-go/1.42.0" 172.16.195.126:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
172.18.29.225 - - [01/Mar/2022:15:31:25 +0800] grpctestgo.mtest.svc.ab:50051 "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.006 "-" "grpc-go/1.42.0"
```

```
172.16.195.28:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
4  172.18.29.225 - - [01/Mar/2022:15:31:25 +0800] grpctestgo.mtest.svc.ab:50051
"POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.006 "-" "grpc-go/1.42.0"
172.16.195.126:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
5  172.18.29.225 - - [01/Mar/2022:15:31:25 +0800] grpctestgo.mtest.svc.ab:50051
"POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.006 "-" "grpc-go/1.42.0"
172.16.195.126:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
6  172.18.29.225 - - [01/Mar/2022:15:31:25 +0800] grpctestgo.mtest.svc.ab:50051
"POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.006 "-" "grpc-go/1.42.0"
172.16.195.28:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
7  172.18.29.225 - - [01/Mar/2022:15:31:25 +0800] grpctestgo.mtest.svc.ab:50051
"POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.005 "-" "grpc-go/1.42.0"
172.16.195.126:50051 200 0.005 "grpc://grpctestgo.mtest.svc.ab:50051"
8  172.18.29.225 - - [01/Mar/2022:15:31:25 +0800] grpctestgo.mtest.svc.ab:50051
"POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.006 "-" "grpc-go/1.42.0"
172.16.195.28:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
9  172.18.29.225 - - [01/Mar/2022:15:31:25 +0800] grpctestgo.mtest.svc.ab:50051
"POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.006 "-" "grpc-go/1.42.0"
172.16.195.28:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
10 172.18.29.225 - - [01/Mar/2022:15:31:25 +0800] grpctestgo.mtest.svc.ab:50051
"POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.224 "-" "grpc-go/1.42.0"
172.16.195.28:50051 200 0.224 "grpc://grpctestgo.mtest.svc.ab:50051"
11 10.72.8.68 - - [01/Mar/2022:15:31:27 +0800] 10.72.8.62:9180 "PUT
/apisix/admin/upstreams/microservice-beehive-grpctestgo-mtest-10003395-50051
HTTP/1.1" 200 567 0.025 "-" "go-resty/2.7.0 (https://github.com/go-resty/resty)"
- - - "http://10.72.8.62:9180"
12 172.18.29.225 - - [01/Mar/2022:15:31:28 +0800] grpctestgo.mtest.svc.ab:50051
"POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0"
172.16.195.126:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
13 172.18.29.225 - - [01/Mar/2022:15:31:28 +0800] grpctestgo.mtest.svc.ab:50051
"POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0"
172.16.195.126:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
14 172.18.29.225 - - [01/Mar/2022:15:31:28 +0800] grpctestgo.mtest.svc.ab:50051
"POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0"
172.16.195.126:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
15 172.18.29.225 - - [01/Mar/2022:15:31:28 +0800] grpctestgo.mtest.svc.ab:50051
"POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0"
172.16.195.126:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
16 172.18.29.225 - - [01/Mar/2022:15:31:28 +0800] grpctestgo.mtest.svc.ab:50051
"POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0"
172.16.195.126:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
17 172.18.29.225 - - [01/Mar/2022:15:31:28 +0800] grpctestgo.mtest.svc.ab:50051
"POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0"
172.16.195.126:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
18 172.18.29.225 - - [01/Mar/2022:15:31:28 +0800] grpctestgo.mtest.svc.ab:50051
"POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0"
172.16.195.126:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
19 172.18.29.225 - - [01/Mar/2022:15:31:28 +0800] grpctestgo.mtest.svc.ab:50051
```

```
    "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0"
    172.16.195.126:50051 200 0.006 "grpc://grpctestgo.mtest.svc.ab:50051"
20  172.18.29.225 - - [01/Mar/2022:15:31:28 +0800] grpctestgo.mtest.svc.ab:50051
    "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0"
    172.16.195.126:50051 200 0.007 "grpc://grpctestgo.mtest.svc.ab:50051"
21  172.18.29.225 - - [01/Mar/2022:15:31:28 +0800] grpctestgo.mtest.svc.ab:50051
    "POST /example.Greeter/SayHello HTTP/2.0" 200 9 0.007 "-" "grpc-go/1.42.0"
    172.16.195.126:50051 200 0.007 "grpc://grpctestgo.mtest.svc.ab:50051"
```

> 结论：秒级

说明：

1. `PUT /apisix/admin/upstreams` 这行说明是 Upstream 变化后，APISIX Upstream 更新。
2. 可以看到，更新前，APISIX 指向 2 个不同 Upstream，更新后，指向 1 个 Upstream 。
3. 从日志看，可以认为是实时生效的。

2.4 结论

1. Upstream 的变化，是秒级生效的。无法从日志上，精确到毫秒，绝对的精确，得从 APISIX 本身能打印出来才行。
2. 为了排除日志时间上的干扰，最好的测试方式是，看 63 网关的压测日志，看 62 网关的 admin api 更新日志，然后看 Upstream 的生效时间。
3. Upstream 的扩缩容，是无损的（说明：如果有损，则压测工具会报错退出）。

3.2 规则生效速度

①：测试条件

首先，istio-apisix-translator 已经可以做到，当 Aos 微服务的 `默认` 访问规则变化时，APISIX 同步变化。因此，我们只需要测试当 aos 切流量时，apisix 何时生效即可。

oudder-mservice 应用下有 2 个版本

1. 一个版本指向 172 IP
2. 一个版本指向 10.72 IP

切流量后，如果规则发生变化时，apisix 日志中的上游，从 10.72 IP 变为 172 IP 即为切换生效。

②：压测过程中，切 oudder-mservice 的流量

首先，k6s 压测脚本为：

Shell

```
import http from 'k6/http';
export let options = {
  vus: 50, // 指定要同时运行的虚拟用户数量
  duration: '20s', // 指定测试运行的总持续时间
};
// default 默认函数
export default function () {
  // 标头
  let params = { headers: { 'Content-Type': 'application/json' } };

  var res=http.get("http://oudder.mservice.svc.ab",params)
}
```

主机的host，已经将 oudder.mservice.svc.ab 指向到了 63 网关

Shell

```
10.72.8.63    oudder.mservice.svc.ab
```

开始压测：

Shell

```
[root@knode10-72-8-68 benchmark]# k6 run get_oudder_route.js

          /\      |‾‾| /‾‾/   /‾‾/
     /\  /  \     |  |/  /   /  /
    /  \/    \    |     (   /   ‾‾\
   /          \   |  |\  \ |  (‾)  |
  / __________ \  |__| \__\ \_____/ .io

  execution: local
     script: get_oudder_route.js
     output: -

  scenarios: (100.00%) 1 scenario, 50 max VUs, 50s max duration (incl. graceful
stop):
           * default: 50 looping VUs for 20s (gracefulStop: 30s)

^C
running (11.1s), 00/50 VUs, 12591 complete and 50 interrupted iterations
default ✗ [====================>----------------] 50 VUs  11.1s/20s

     data_received..................: 3.4 MB 310 kB/s
     data_sent......................: 1.5 MB 137 kB/s
```

```
     http_req_blocked...............: avg=13µs    min=1.24µs  med=3.37µs
max=4.3ms    p(90)=4.75µs  p(95)=6.29µs
     http_req_connecting............: avg=6.94µs  min=0s      med=0s
max=4.09ms   p(90)=0s      p(95)=0s
     http_req_duration..............: avg=43.91ms min=1.92ms  med=17.43ms
max=328.45ms p(90)=90.79ms p(95)=95.66ms
       { expected_response:true }...: avg=43.91ms min=1.92ms  med=17.43ms
max=328.45ms p(90)=90.79ms p(95)=95.66ms
     http_req_failed................: 0.00%  ✓ 0           ✗ 12591
     http_req_receiving.............: avg=57µs    min=13.75µs med=50.08µs
max=7.5ms    p(90)=78.73µs p(95)=92.15µs
     http_req_sending...............: avg=21.74µs min=5.83µs  med=15.16µs
max=3.13ms   p(90)=29.37µs p(95)=41.39µs
     http_req_tls_handshaking.......: avg=0s      min=0s      med=0s      max=0s
p(90)=0s      p(95)=0s
     http_req_waiting...............: avg=43.83ms min=1.7ms   med=17.32ms
max=328.37ms p(90)=90.7ms  p(95)=95.56ms
     http_reqs......................: 12591  1133.845858/s
     iteration_duration.............: avg=44.03ms min=2.05ms  med=17.55ms
max=328.57ms p(90)=90.92ms p(95)=95.79ms
     iterations.....................: 12591  1133.845858/s
     vus............................: 50     min=50        max=50
     vus_max........................: 50     min=50        max=50
```

> 由于压测只有 20s，所以要及时切流量

③：分析日志

Shell

```
10.72.8.68 - - [01/Mar/2022:16:13:49 +0800] oudder.mservice.svc.ab "GET /
HTTP/1.1" 200 70 0.088 "-" "k6/0.36.0 (https://k6.io/)" 10.72.16.163:80 200
0.088 "http://oudder.mservice.svc.ab"
10.72.8.68 - - [01/Mar/2022:16:13:49 +0800] oudder.mservice.svc.ab "GET /
HTTP/1.1" 200 70 0.090 "-" "k6/0.36.0 (https://k6.io/)" 10.72.16.163:80 200
0.089 "http://oudder.mservice.svc.ab"
10.72.8.68 - - [01/Mar/2022:16:13:49 +0800] oudder.mservice.svc.ab "GET /
HTTP/1.1" 200 70 0.089 "-" "k6/0.36.0 (https://k6.io/)" 10.72.16.163:80 200
0.089 "http://oudder.mservice.svc.ab"
10.72.8.68 - - [01/Mar/2022:16:13:49 +0800] oudder.mservice.svc.ab "GET /
HTTP/1.1" 200 70 0.099 "-" "k6/0.36.0 (https://k6.io/)" 10.72.16.163:80 200
0.099 "http://oudder.mservice.svc.ab"
10.72.8.68 - - [01/Mar/2022:16:13:49 +0800] oudder.mservice.svc.ab "GET /
HTTP/1.1" 200 70 0.098 "-" "k6/0.36.0 (https://k6.io/)" 10.72.16.163:80 200
0.097 "http://oudder.mservice.svc.ab"
10.72.8.68 - - [01/Mar/2022:16:13:49 +0800] oudder.mservice.svc.ab "GET /
HTTP/1.1" 200 70 0.097 "-" "k6/0.36.0 (https://k6.io/)" 10.72.16.163:80 200
0.096 "http://oudder.mservice.svc.ab"
```

```
7  10.72.8.68 - - [01/Mar/2022:16:13:49 +0800] oudder.mservice.svc.ab "GET /
   HTTP/1.1" 200 70 0.096 "-" "k6/0.36.0 (https://k6.io/)" 10.72.16.163:80 200
   0.095 "http://oudder.mservice.svc.ab"
8  10.72.8.68 - - [01/Mar/2022:16:13:49 +0800] oudder.mservice.svc.ab "GET /
   HTTP/1.1" 200 70 0.013 "-" "k6/0.36.0 (https://k6.io/)" 172.16.197.96:80 200
   0.014 "http://oudder.mservice.svc.ab"
9  10.72.8.68 - - [01/Mar/2022:16:13:49 +0800] oudder.mservice.svc.ab "GET /
   HTTP/1.1" 200 70 0.026 "-" "k6/0.36.0 (https://k6.io/)" 172.16.197.96:80 200
   0.026 "http://oudder.mservice.svc.ab"
10 10.72.8.68 - - [01/Mar/2022:16:13:49 +0800] oudder.mservice.svc.ab "GET /
   HTTP/1.1" 200 70 0.013 "-" "k6/0.36.0 (https://k6.io/)" 172.16.197.96:80 200
   0.014 "http://oudder.mservice.svc.ab"
11 10.72.8.68 - - [01/Mar/2022:16:13:49 +0800] oudder.mservice.svc.ab "GET /
   HTTP/1.1" 200 70 0.013 "-" "k6/0.36.0 (https://k6.io/)" 172.16.197.96:80 200
   0.014 "http://oudder.mservice.svc.ab"
12 10.72.8.68 - - [01/Mar/2022:16:13:49 +0800] oudder.mservice.svc.ab "GET /
   HTTP/1.1" 200 70 0.012 "-" "k6/0.36.0 (https://k6.io/)" 172.16.197.96:80 200
   0.013 "http://oudder.mservice.svc.ab"
13 10.72.8.68 - - [01/Mar/2022:16:13:49 +0800] oudder.mservice.svc.ab "GET /
   HTTP/1.1" 200 70 0.012 "-" "k6/0.36.0 (https://k6.io/)" 172.16.197.96:80 200
   0.013 "http://oudder.mservice.svc.ab"
14 10.72.8.68 - - [01/Mar/2022:16:13:49 +0800] oudder.mservice.svc.ab "GET /
   HTTP/1.1" 200 70 0.012 "-" "k6/0.36.0 (https://k6.io/)" 172.16.197.96:80 200
   0.012 "http://oudder.mservice.svc.ab"
15 10.72.8.68 - - [01/Mar/2022:16:13:49 +0800] oudder.mservice.svc.ab "GET /
   HTTP/1.1" 200 70 0.017 "-" "k6/0.36.0 (https://k6.io/)" 172.16.197.96:80 200
   0.019 "http://oudder.mservice.svc.ab"
16 10.72.8.68 - - [01/Mar/2022:16:13:49 +0800] oudder.mservice.svc.ab "GET /
   HTTP/1.1" 200 70 0.008 "-" "k6/0.36.0 (https://k6.io/)" 172.16.197.96:80 200
   0.009 "http://oudder.mservice.svc.ab"
```

可以看到，从 `2022:16:13:49` 开始，上游 IP 开始变为了 10.72 IP 变为了 172 IP。

而 Route 的变更日志如下：

Shell

```
1  10.72.8.68 - - [01/Mar/2022:16:13:49 +0800] 10.72.8.62:9180 "PUT
   /apisix/admin/routes/microservice-beehive-oudder-mservice-413031-80 HTTP/1.1"
   200 662 0.013 "-" "go-resty/2.7.0 (https://github.com/go-resty/resty)" - - -
   "http://10.72.8.62:9180"
```

时间也是 `2022:16:13:49`

结论：Route 的变化，是实时的。

3.1 结论

1. Route 的变化是实时的。

2. 多次试验，结果是一致的。

3.3.2 多个规则同时生效情况下，新规则权重高，旧规则权重低，删除新规则

①：测试条件

创建 2 个访问规则，新的规则权重高，旧的规则权重低。但是，我们不需要手动创建，只需要在 oudder-mservice 应用下，发布 2 个版本，然后，都点一下上线即可。

> Istio-apisix-translator 会同步这 2 个规则，最新点上线的，权重高。

②：压测

压测脚本，复用 3.2 场景的。

首先，k6s 压测脚本为：

```Shell
import http from 'k6/http';
export let options = {
  vus: 50, // 指定要同时运行的虚拟用户数量
  duration: '20s', // 指定测试运行的总持续时间
};
// default 默认函数
export default function () {
  // 标头
  let params = { headers: { 'Content-Type': 'application/json' } };

  var res=http.get("http://oudder.mservice.svc.ab",params)
}
```

主机的host，已经将 oudder.mservice.svc.ab 指向到了 63 网关

```Shell
10.72.8.63    oudder.mservice.svc.ab
```

③：执行压测

```Shell
k6 run get_oudder_route.js
```

④：压测过程中，从 apisix admin dashboard 中，删除新的访问规则，

⑤：最后看 k6 的压测报告即可。

Shell

```
[root@knode10-72-8-68 benchmark]# k6 run get_oudder_route.js

          /\      |‾‾| /‾‾/   /‾‾/
     /\  /  \     |  |/  /   /  /
    /  \/    \    |     (   /   ‾‾\
   /          \   |  |\  \ |  (‾)  |
  / __________ \  |__| \__\ \_____/ .io

  execution: local
     script: get_oudder_route.js
     output: -

  scenarios: (100.00%) 1 scenario, 50 max VUs, 50s max duration (incl. graceful stop):
           * default: 50 looping VUs for 20s (gracefulStop: 30s)


running (20.0s), 00/50 VUs, 21964 complete and 0 interrupted iterations
default ✓ [======================================] 50 VUs  20s

     data_received..................: 6.0 MB 299 kB/s
     data_sent......................: 2.6 MB 132 kB/s
     http_req_blocked...............: avg=10.47µs min=1.21µs  med=3.34µs max=2.9ms    p(90)=4.74µs  p(95)=6.25µs
     http_req_connecting............: avg=5.34µs  min=0s      med=0s     max=1.64ms   p(90)=0s      p(95)=0s
     http_req_duration..............: avg=45.43ms min=2.53ms  med=17.83ms max=296.4ms  p(90)=92.72ms p(95)=96.64ms
       { expected_response:true }...: avg=45.43ms min=2.53ms  med=17.83ms max=296.4ms  p(90)=92.72ms p(95)=96.64ms
     http_req_failed................: 0.00%  ✓ 0          ✗ 21964
     http_req_receiving.............: avg=56.59µs min=14.64µs med=48.59µs max=6.08ms   p(90)=77.93µs p(95)=91.3µs
     http_req_sending...............: avg=21.15µs min=5.8µs   med=14.98µs max=9.83ms   p(90)=28.04µs p(95)=39.78µs
     http_req_tls_handshaking.......: avg=0s      min=0s      med=0s      max=0s       p(90)=0s      p(95)=0s
     http_req_waiting...............: avg=45.36ms min=2.46ms  med=17.75ms max=296.3ms  p(90)=92.65ms p(95)=96.55ms
     http_reqs......................: 21964  1095.545175/s
     iteration_duration.............: avg=45.56ms min=2.63ms  med=17.96ms max=296.52ms p(90)=92.84ms p(95)=96.82ms
     iterations.....................: 21964  1095.545175/s
     vus............................: 50     min=50       max=50
     vus_max........................: 50     min=50       max=50
```

```
     vus_max.....................: 50     min=50        max=50

^C[root@knode10-72-8-68 benchmark]# k6 run get_oudder_route.js

          /\      |‾‾| /‾‾/   /‾‾/
     /\  /  \     |  |/  /   /  /
    /  \/    \    |     (   /   ‾‾\
   /          \   |  |\  \ |  (‾)  |
  / __________ \  |__| \__\ \_____/ .io

  execution: local
     script: get_oudder_route.js
     output: -

  scenarios: (100.00%) 1 scenario, 50 max VUs, 50s max duration (incl. graceful stop):
           * default: 50 looping VUs for 20s (gracefulStop: 30s)


running (20.0s), 00/50 VUs, 22228 complete and 0 interrupted iterations
default ✓ [======================================] 50 VUs  20s

     data_received..................: 6.1 MB 303 kB/s
     data_sent......................: 2.7 MB 133 kB/s
     http_req_blocked...............: avg=9.85µs  min=1.22µs  med=3.37µs  max=1.92ms   p(90)=4.7µs   p(95)=6.2µs
     http_req_connecting............: avg=4.89µs  min=0s      med=0s      max=1.23ms   p(90)=0s      p(95)=0s
     http_req_duration..............: avg=44.9ms  min=2.92ms  med=17.59ms max=486.29ms p(90)=92.59ms p(95)=96.84ms
       { expected_response:true }...: avg=44.9ms  min=2.92ms  med=17.59ms max=486.29ms p(90)=92.59ms p(95)=96.84ms
     http_req_failed................: 0.00%  ✓ 0           ✗ 22228
     http_req_receiving.............: avg=56.77µs min=14.74µs med=49.38µs max=6.85ms   p(90)=77.8µs  p(95)=89.24µs
     http_req_sending...............: avg=19.47µs min=5.56µs  med=14.88µs max=956.82µs p(90)=28.41µs p(95)=39.71µs
     http_req_tls_handshaking.......: avg=0s      min=0s      med=0s      max=0s       p(90)=0s      p(95)=0s
     http_req_waiting...............: avg=44.83ms min=2.83ms  med=17.53ms max=486.23ms p(90)=92.51ms p(95)=96.77ms
     http_reqs......................: 22228  1108.970298/s
     iteration_duration.............: avg=45.02ms min=3ms     med=17.73ms max=486.42ms p(90)=92.72ms p(95)=96.97ms
     iterations.....................: 22228  1108.970298/s
     vus............................: 50     min=50        max=50
     vus_max........................: 50     min=50        max=50
```

可以看到，http_req_failed 数量为 0 ，即：规则变化，不会产生 4XX 5XX 问题。

3.4 规则变化 gRPC 无损

①：测试条件

由于 2 个原因，测试 gRPC 要麻烦

1. Istio-apisix-translator 没有自动同步 gRPC Route 到 apisix，因为 istio 中，没有标记 gRPC 的地方，translator 无法识别规则是 gRPC 还是非 gRPC 。
2. k6 无法压测 gRPC

因此，得自己写脚本测试 gRPC。

我们之前，已在 apisix 中，手动创建了 `grpctestgo` 的路由规则

Mac 机器绑定的 hosts 如下：

```
Apache
1  10.72.8.63 grpctestgo.mtest.svc.ab
```

由于规则变化场景较多，我们主要以访问规则变化为例，比如增加插件。

②：开始压测过程，并同时调整 Route 开启此访问规则的流控插件

gRPC 测试脚本如下：

Go

```go
package main

import (
    example "gitlab.mfwdev.com/mtest/grpctestgo/grpc"
    "log"
    "strconv"
    "time"

    "golang.org/x/net/context"
    "google.golang.org/grpc"
)

// 定义请求地址
const (
    ADDRESS string = "grpctestgo.mtest.svc.ab:50051"
)

// main 方法实现对 gRPC 接口的请求
func main() {
    conn, err := grpc.Dial(ADDRESS, grpc.WithInsecure())
    if err != nil {
        log.Fatalln("Can't connect: " + ADDRESS)
    }
    defer conn.Close()
    client := example.NewGreeterClient(conn)
    var total int
    for i := 0; i < 20; i++ {
        go func() {
            for {
                total++
                resp, err := client.SayHello(context.Background(), &example.HelloRequest{Name: "request num: " + strconv.Itoa(total)})
                if err != nil {
                    log.Fatalln("Do Format error:" + err.Error())
                }
                log.Println(resp.Message)
                time.Sleep(50 * time.Millisecond)
            }
        }()
    }
    select {}
}
```

脚本说明：

1. 并发数为 20，每个请求结束后，休眠 50ms。

3.4 结论

1. 插件的变化，不会引发 4XX 5XX 问题。
2. 为了验证插件的生效，使用 `limit-count` 插件，当设置的 QPS 较低时，脚本直接中断，说明插件的配置，对 gRPC 来说，是有效的。