Tubi’s Story with gRPC (in Scala)

Scala Meetup @ 2023/01/14

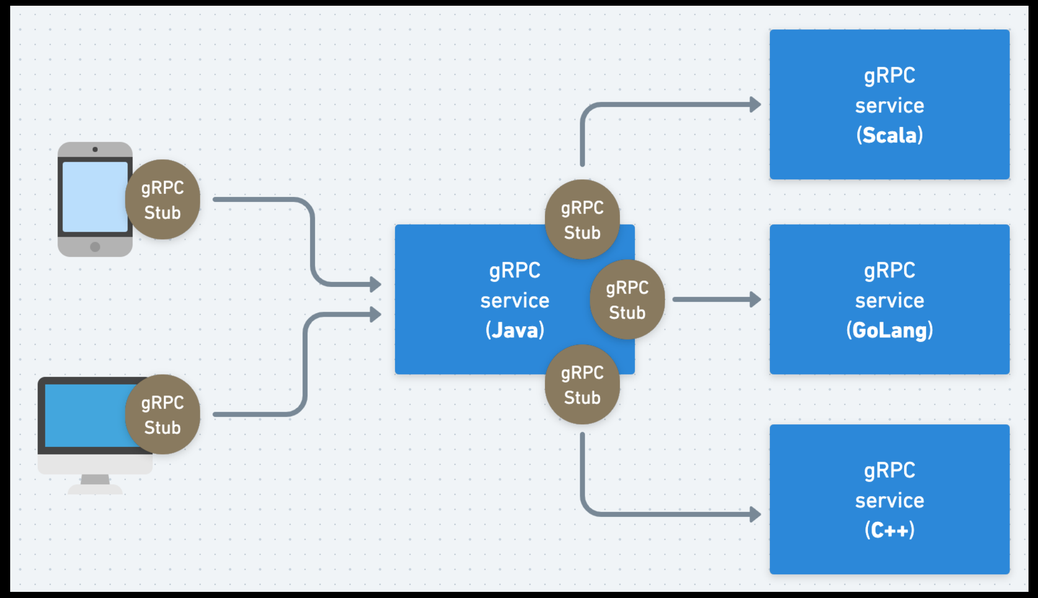
张正来 <zhenglaizhang@tubi.tv>

目录

- gRPC简介
- Tubi Scala team & gRPC
- Akka-gRPC 迁移
- Q&A

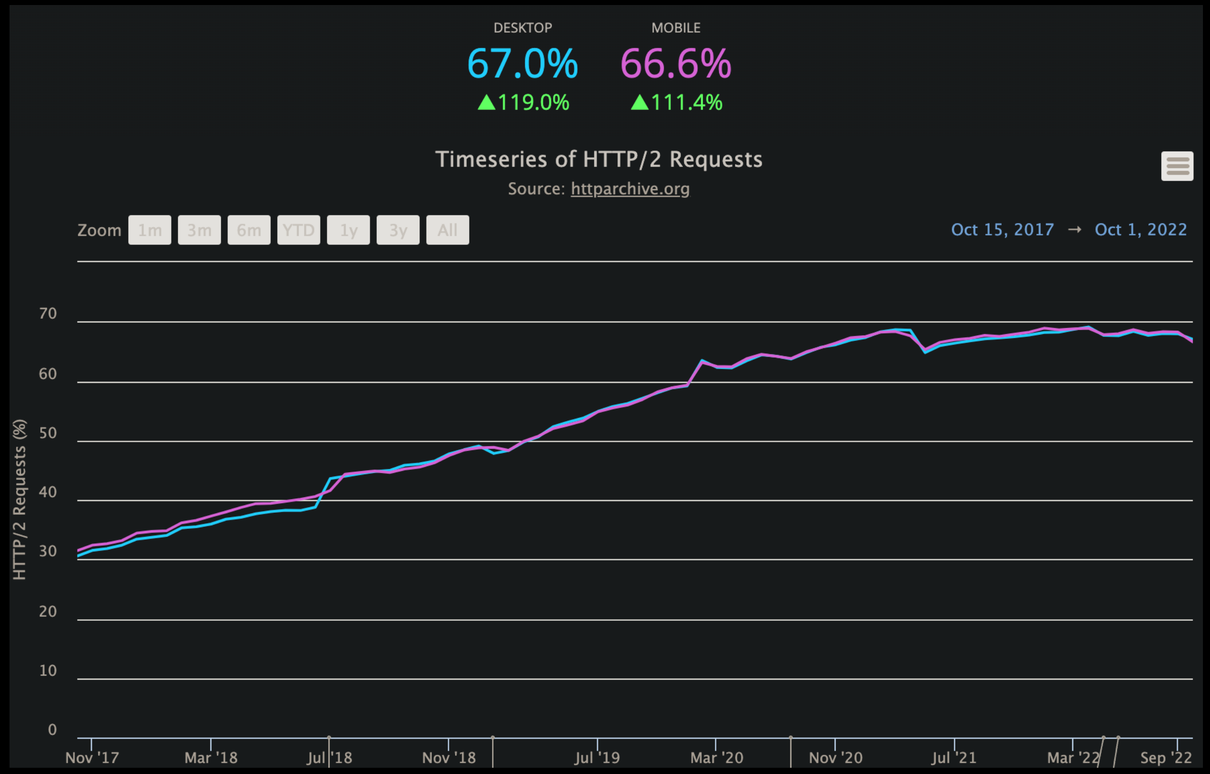
什么是gRPC

- 高性能、开源和通用的 RPC 框架
- Google (2015) => CNCF (2017)
- Cloud Native和Microservices系统通信
- 已被业内广泛应用
- g
  - grpc remote procedure call
  - 'g' stands for something different



gRPC特性（一）

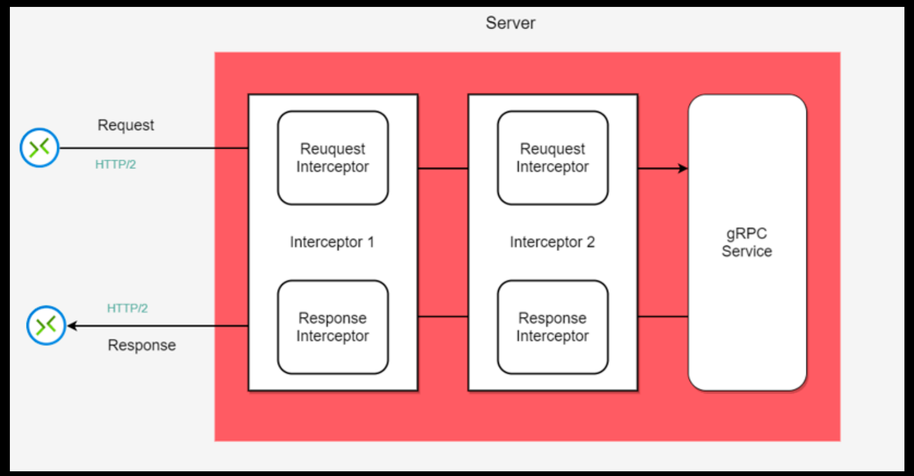
- 编码协议(IDL)默认基于Protocol Buffers
  - 接口优先
  - 高效
  - 可扩展
  - 兼容性好
- 传输协议基于 HTTP/2
  - 二进制帧
  - 流式传输和多路复用（multiplexing）
  - 元数据（metadata）
- 跨语言、跨平台
- 高性能



Ref: https://httparchive.org/reports/state-of-the-web#h2

gRPC特性（二）

- 截止时间（deadline）
- 取消（cancellation）
- 错误处理
- 拦截器（interceptor）扩展机制
- Server reflection
- TLS加密

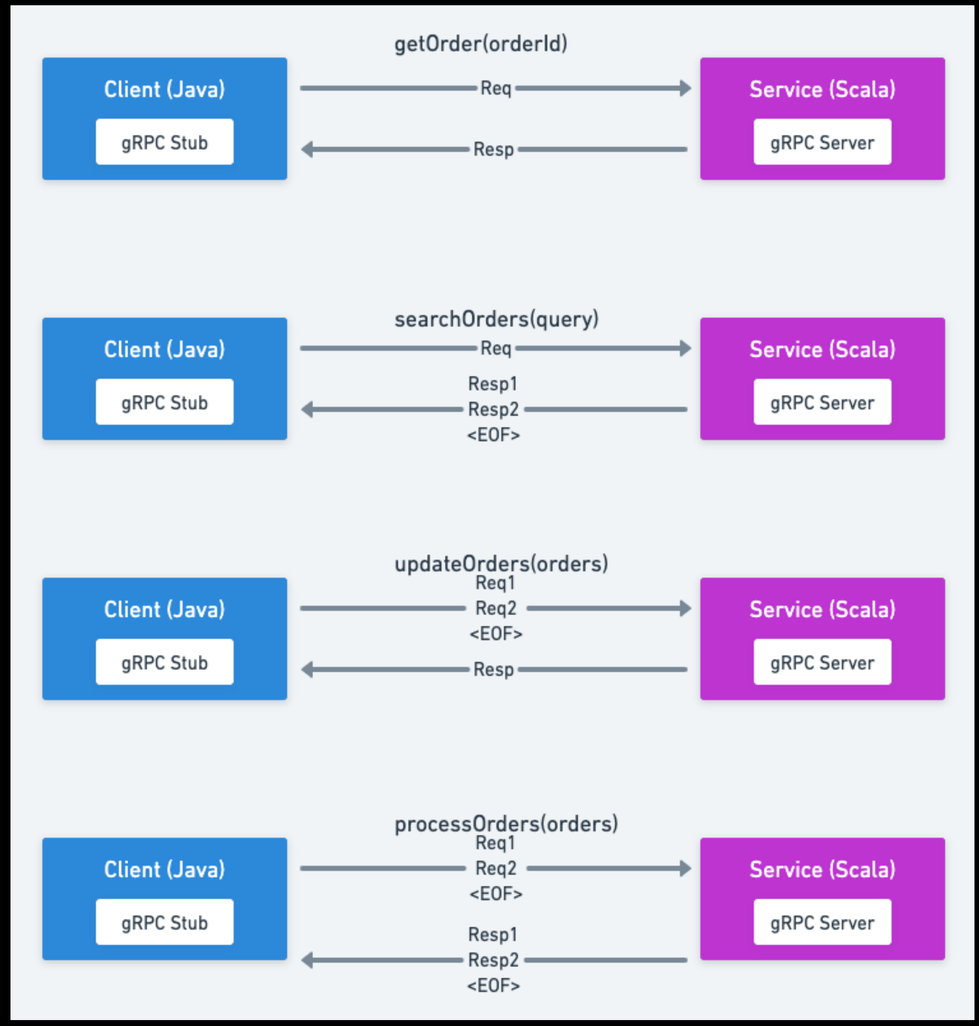


Ref: https://techdozo.dev/wp-content/uploads/2022/04/image-4.png

gRPC状态码	gRPC信息	含义
0	OK	成功
1	CANCELLED	操作已被取消
2	UNKNOWN	未知错误
3	INVALID_ARGUMENT	客户端参数非法
4	DEADLINE_EXCEEDED	操作超过了截止时间
5	NOT_FOUND	请求实体未找到
6	ALREADY_EXISTS	客户端试图创建的实体已存在
7	PERMISSION_DENIED	调用者没有权限执行特定操作
8	RESOURCE_EXHAUSTED	资源已耗尽
9	FAILED_PRECONDITION	操作被拒绝，系统没有处于执行操作所需状态
10	ABORTED	操作被中止
11	OUT_OF_RANGE	操作超出了合法的范围
12	UNIMPLEMENTED	该操作未实现
13	INTERNAL	内部错误
14	UNAVAILABLE	服务当前不可用
15	DATA_LOSS	数据丢失或损坏
16	UNAUTHENTICATED	客户端没有进行操作的合法认证凭证

gRPC通信模式(Demo)

- 单项RPC (unary/simple RPC)
- 服务端流式 RPC (server-side streaming RPC)
- 客户端流式 RPC (client-side streaming RPC)
- 双向流式 RPC (bidirectional streaming RPC)



框架对比

	SOAP	REST	Thrift	gRPC	GraphQL
发布时间	90年代末	2000	2007	2015	2015
格式	XML	JSON, XML, etc	JSON,Binary	默认ProtoBuf, JSON	JSON
传输协议	HTTP/TCP	HTTP	TCP	HTTP/2	HTTP?
Pros	广泛使用、标准明确	Resource，数据格式灵活，易上手、架构风格	用途广泛、高性能	现代化、易上手、灵活、高性能	data structuring灵活、查询灵活和强大、现代化，schema一等公民
Cons	Heavy payload，复杂，难于上手，难于调试	过于理论化，标准不统一，服务接口没有强类型定义，低效的文本格式	流式数据处理能力欠缺，没有很好的发展起来	不支持HTTP/1.1，浏览器支持有待完善	上手困难，容易出现性能问题

Tubi Scala team & gRPC

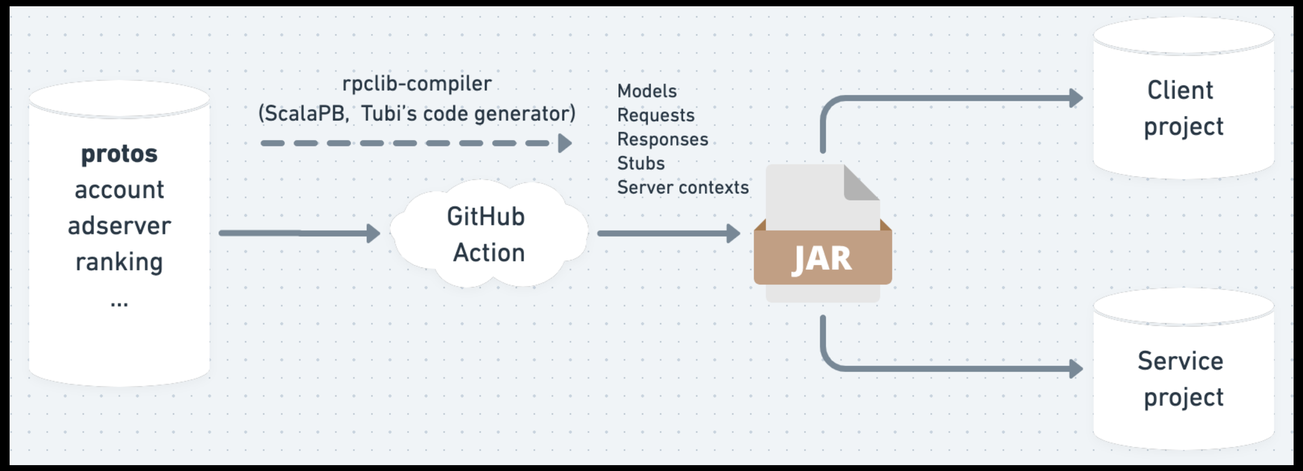
- **Rainmaker** (广告)
- **Delphi** (推荐系统在线服务)
- **Data**
- **ML** (广告，推荐)

```
commit 5a23800c926a7bd47d2f2fc5811078626e36449f
Author: Marios                 @tubi.tv>
Date:   Wed Feb 1 00:02:15 2017 -0800

    initial commit, single proto definition

commit 5c607576603b7004f1e221075bfe4c83959dba39
Author: Marios                 @tubi.tv>
Date:   Thu Mar 2 01:59:40 2017 -0800

    Some basic stubs to test with and bazel build
```

Why gRPC

- 面向服务定义而非资源定义，适合各种业务需求
- 强类型约束，降低开发和维护成本，提高工程师生产力
- 多语言支持，不同团队可以根据业务需要选择更合适的编程语言
- 不同团队可以通过集中的ProtoBuf代码库轻松的共享服务定义
- ProtoBuf高效的序列化性能和较小的序列化体积
- 原生支持 stream 操作, 大数据量传输更加高效
- 性能好，可支持高吞吐低延迟服务
- 社区活跃、生态快速成熟

开发流程



- protos
  - mono repo
  - 集中存储tubi内部各个gRPC服务定义
  - ScalaPB auxiliary options
- rpclib-compiler
  - tubi自研的sbt插件
  - 调用ScalaPB编译protobufs生成models类型定义(messages和enums)
  - 调用自研的code generator生成client stub和server skeleton
- rpclib-runtime
  - 服务器端/客户端辅助方法
  - MDC和Envoy配置(x-request-id/x-tubi-header)
  - 重试策略、超时配置、断路器策略

测试

- 单元测试和集成测试
- grpcurl和grpc_cli
- 本地和staging环境验证
- grpc-ui

gRPC Server Target

ranking:50051

Use TLS
Restart Connection
Use local proto
Use request metadata

	Key	Value
×	x-request-id	test

+

Services personal.PersonalizeMatrix

Methods PersonalizeAutoplay

```
{
  "request_context":
    {
      "device_id": "abc",
      "user_id": 0,
      "platform": 1,
      "is_authenticated": false,
      "ip_address": "127.0.0.1",
      "utc_timestamp": "2017-01-15T01:30:15.01Z",
      "country": 1,
      "timezone": "San Francisco"
    },
  "current_content": {
```

Schema Input

```
message PersonalizeAutoplayRequest {
  .personal.RequestContext request_contex
  .personal.Content source_content = 2 [(
  .personal.Content current_content = 3 [
  oneof candidates {
    .personal.Episodes episodes = 4 [(.sc
    .personal.Container container = 5 [(.
    .personal.Sequels sequels = 10 [(.sca
  }
```

SUBMIT

Response:

Time : 112ns

```
{
  "content": [
    {
      "id": 99999999999,
      "contentType": "SERIES"
    },
```

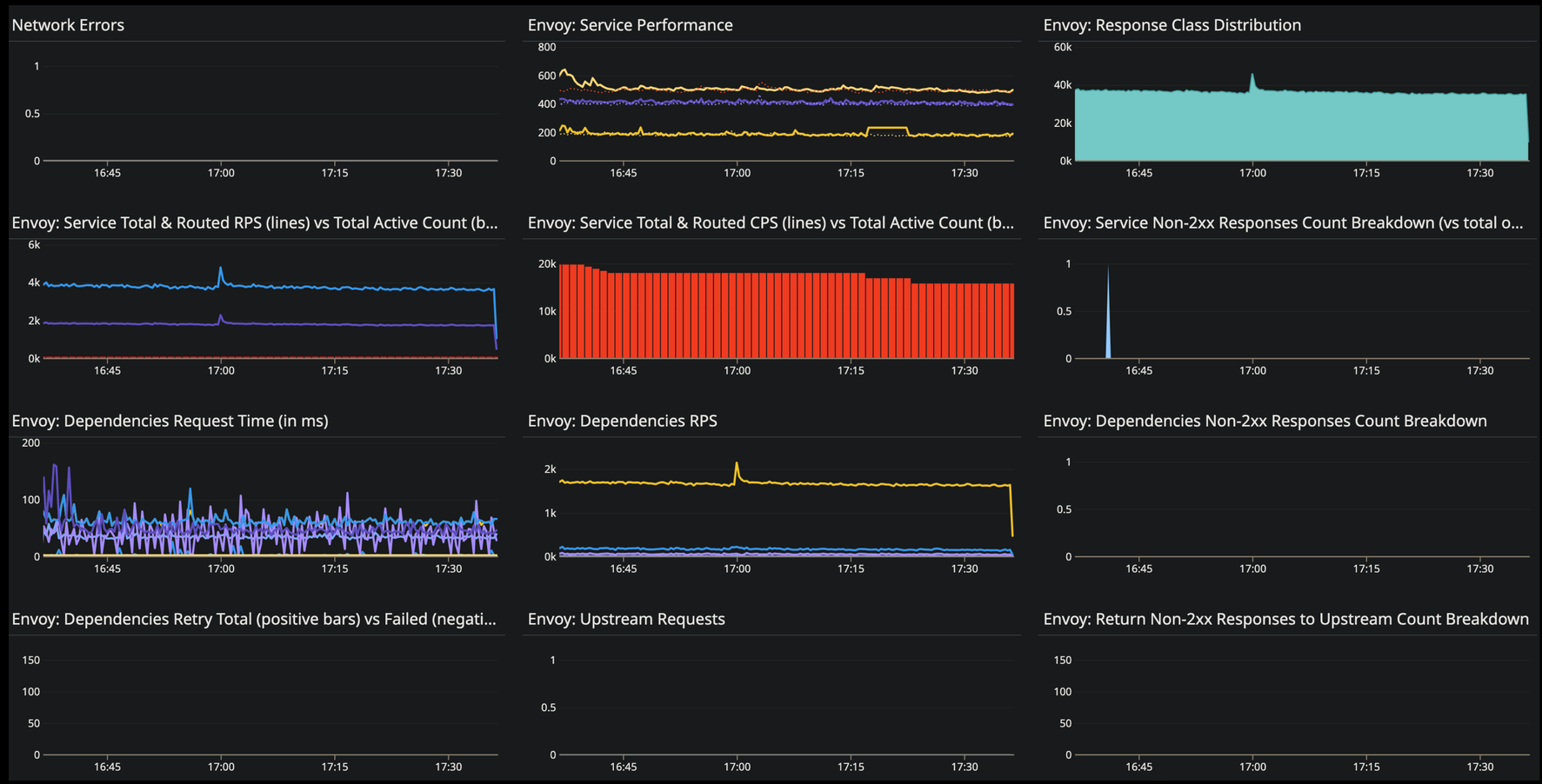
部署

- K8s紧密集成
  - gRPC Health Check
  - 服务发现 -> k8s service
  - k8s readiness和liveness
  - Envoy

```yaml
readinessProbe:
  initialDelaySeconds: 20
  timeoutSeconds: 5
  exec:
    command: [ "/bin/grpc_health_probe", "-addr=:50051", "-service=ranking" ]
livenessProbe:
  initialDelaySeconds: 20
  timeoutSeconds: 5
  exec:
    command: [ "/bin/grpc_health_probe", "-addr=:50051", "-service=ranking" ]
```

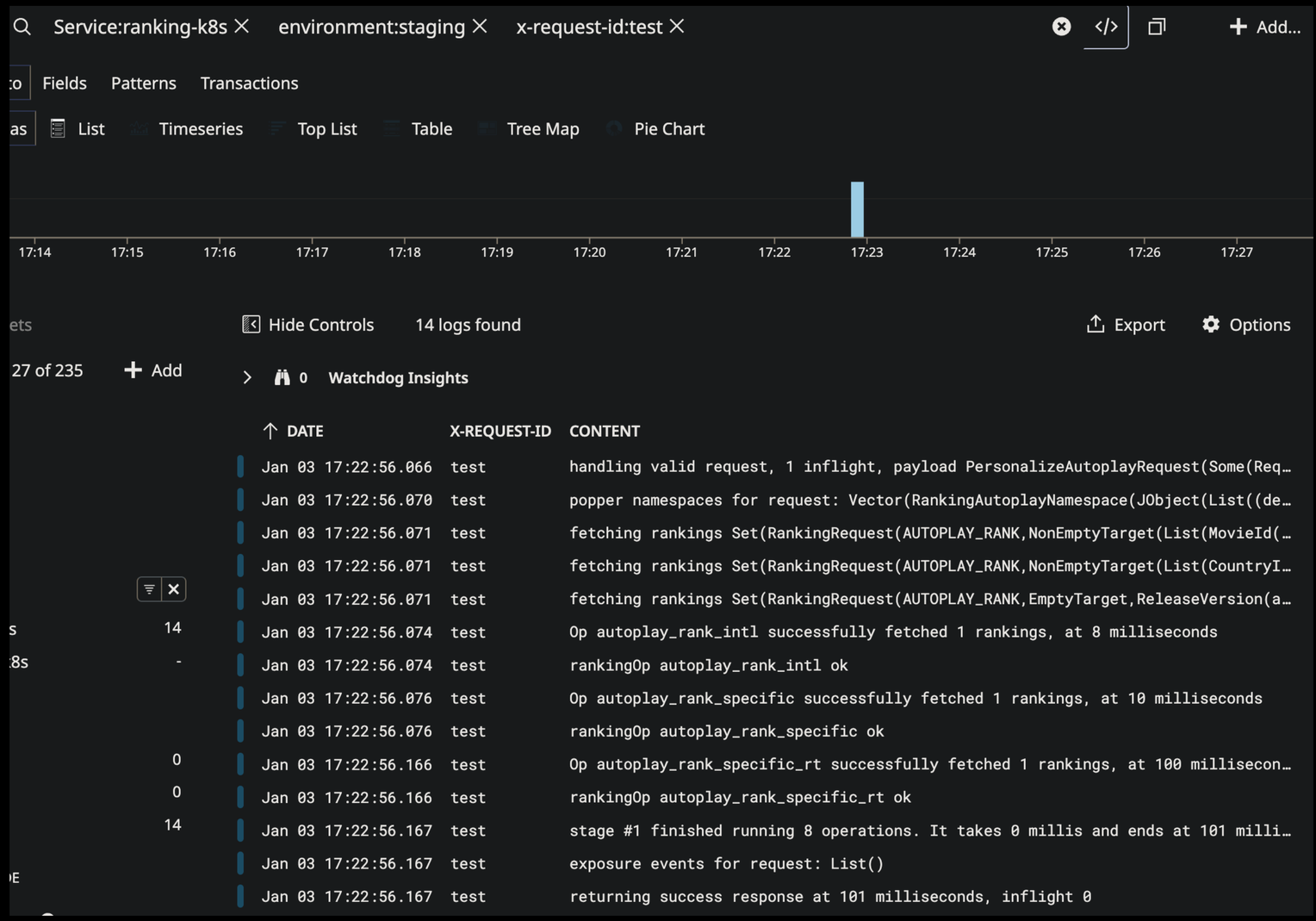
Metrics

- 系统 Metrics
- App 通用 Metrics
- App metrics



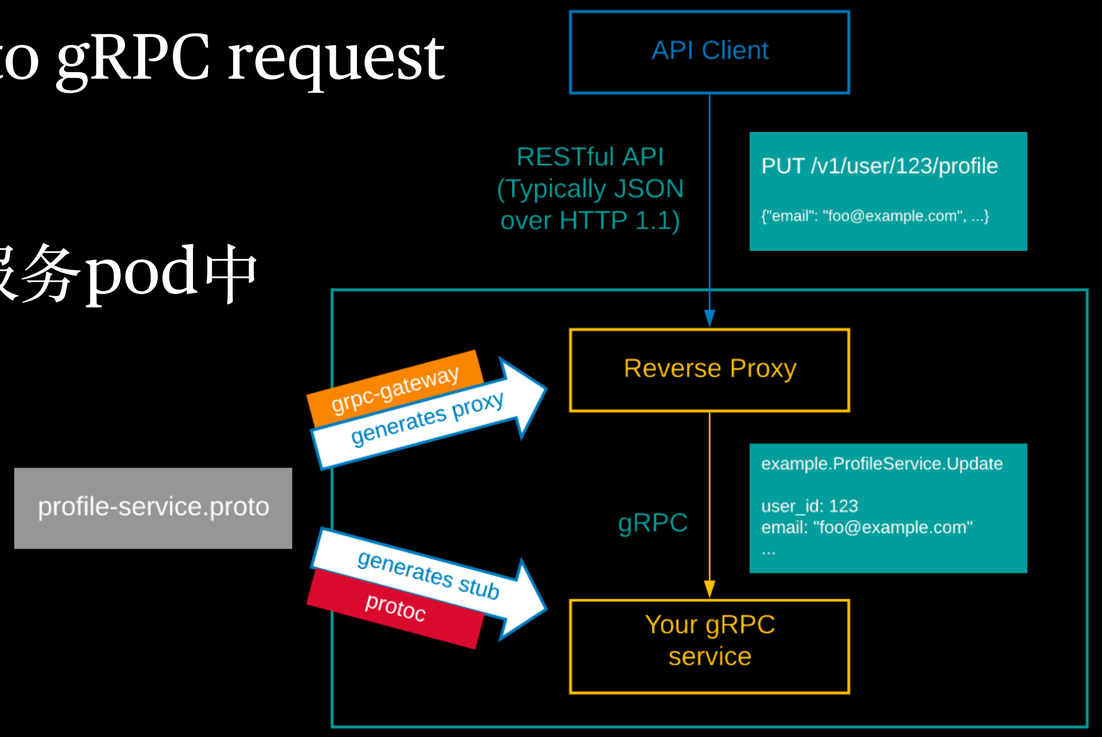
Logging with MDC

- Mapped Diagnostic Context
- x-request-id
- 客户端id、客户端ip、客户端参数
- 在海量日志数据中快速捞到可用的日志信息



REST on gRPC

- grpc-bridge
  - Internal service proxy HTTP request to gRPC request
  - 基于grpc-gateway
  - 以sidecar container方式注入到gRPC服务pod中



ref: architecture_introduction_diagram

Akka-gRPC迁移

感谢Weiwen同学的迁移工作

Akka-gRPC 概述

- 基于Akka Streams和Akka HTTP (http2)
- 流式gRPC服务
- 代码生成器
  - Model类定义
  - 基于Akka Stream Sources 的服务端API skeleton
  - Akka HTTP route用来辅助在Akka http上serving gRPC
  - Client stub
- gRPC runtime实现
- gRPC-web支持

Why Akka-gRPC

- Tubi后端重度依赖Akka生态
- 降低维护现有rpclib-runtime的成本
  - 底层实现绑定低版本ScalaPB，间接导致无法升级grpc-api/grpc-core
- Ready for production
- 高性能
  - 官方benchmark
- 更好的特性支持，如server reflection、gRPC-web、metrics

Akka-gRPC性能

- *Intel(R) Xeon(R) Platinum 8259CL CPU @ 2.50GHz*
- *OS: Ubuntu 20.04.4 LTS*
- *Naive random UUID generation*

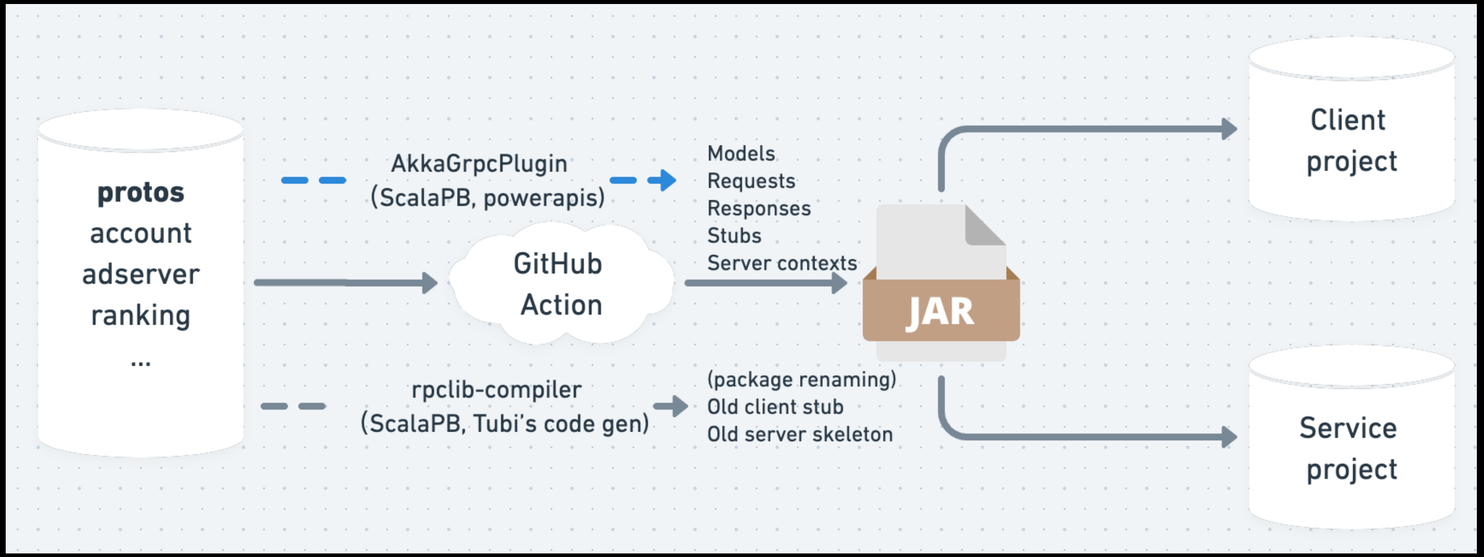
		Akka-grpc(1.1.0)	Rpclib(2.0.16)	Akka-grpc(2.1.4)	Rpclib(3.0.0)
Unary	Summary	Count: 200000 Total: 51.01 s Slowest: 25.18 ms Fastest: 0.42 ms Average: 4.68 ms Requests/sec: 3921.14	Count: 200000 Total: 58.23 s Slowest: 824.90 ms Fastest: 0.29 ms Average: 5.68 ms Requests/sec: 3434.54	Count: 200000 Total: 39.00 s Slowest: 26.68 ms Fastest: 0.29 ms Average: 3.37 ms Requests/sec: 5128.60	Count: 200000 Total: 42.21 s Slowest: 39.53 ms Fastest: 0.26 ms Average: 3.61 ms Requests/sec: 4737.99
	Latency distribution	10 % in 2.05 ms 25 % in 3.21 ms 50 % in 4.42 ms 75 % in 5.76 ms 90 % in 7.69 ms 95 % in 8.90 ms 99 % in 11.10 ms	10 % in 1.88 ms 25 % in 2.89 ms 50 % in 4.14 ms 75 % in 6.50 ms 90 % in 10.27 ms 95 % in 13.85 ms 99 % in 26.75 ms	10 % in 1.40 ms 25 % in 2.12 ms 50 % in 2.97 ms 75 % in 4.17 ms 90 % in 6.06 ms 95 % in 7.15 ms 99 % in 8.99 ms	10 % in 1.41 ms 25 % in 2.24 ms 50 % in 3.23 ms 75 % in 4.46 ms 90 % in 6.39 ms 95 % in 7.64 ms 99 % in 10.04 ms
Client Stream	Summary	Count: 100000 Total: 62.33 s Slowest: 58.52 ms Fastest: 0.82 ms Average: 13.72 ms Requests/sec: 1604.46	Count: 100000 Total: 200.84 s Slowest: 182.31 ms Fastest: 3.02 ms Average: 59.12 ms Requests/sec: 497.90	Count: 100000 Total: 53.92 s Slowest: 63.82 ms Fastest: 0.65 ms Average: 11.55 ms Requests/sec: 1854.61	Count: 100000 Total: 185.94 s Slowest: 113.62 ms Fastest: 4.41 ms Average: 54.42 ms Requests/sec: 537.81
	Latency distribution	10 % in 7.53 ms 25 % in 10.63 ms 50 % in 13.52 ms 75 % in 16.75 ms 90 % in 20.13 ms 95 % in 22.09 ms 99 % in 26.41 ms	10 % in 48.74 ms 25 % in 54.06 ms 50 % in 58.68 ms 75 % in 63.71 ms 90 % in 69.67 ms 95 % in 74.76 ms 99 % in 90.67 ms	10 % in 6.56 ms 25 % in 8.65 ms 50 % in 10.98 ms 75 % in 14.08 ms 90 % in 17.47 ms 95 % in 19.39 ms 99 % in 23.78 ms	10 % in 44.57 ms 25 % in 50.02 ms 50 % in 54.66 ms 75 % in 59.49 ms 90 % in 64.40 ms 95 % in 67.81 ms 99 % in 75.93 ms

Akka-gRPC性能(cont.)

		Akka-grpc(1.1.0)	Rpclib(2.0.16)	Akka-grpc(2.1.4)	Rpclib(3.0.0)
Server Stream	Summary	Count: 100000 Total: 64.84 s Slowest: 131.21 ms Fastest: 1.06 ms Average: 14.84 ms Requests/sec: 1542.33	Count: 100000 Total: 97.66 s Slowest: 121.97 ms Fastest: 1.45 ms Average: 24.62 ms Requests/sec: 1024.00	Count: 100000 Total: 64.78 s Slowest: 75.62 ms Fastest: 1.16 ms Average: 14.99 ms Requests/sec: 1543.76	Count: 100000 Total: 81.64 s Slowest: 82.76 ms Fastest: 1.11 ms Average: 20.48 ms Requests/sec: 1224.83
	Latency distribution	10 % in 8.69 ms 25 % in 11.61 ms 50 % in 14.69 ms 75 % in 17.82 ms 90 % in 20.88 ms 95 % in 22.89 ms 99 % in 27.74 ms	10 % in 13.67 ms 25 % in 18.12 ms 50 % in 23.50 ms 75 % in 29.59 ms 90 % in 36.38 ms 95 % in 41.78 ms 99 % in 55.01 ms	10 % in 9.26 ms 25 % in 11.88 ms 50 % in 14.69 ms 75 % in 17.88 ms 90 % in 21.04 ms 95 % in 23.13 ms 99 % in 28.01 ms	10 % in 11.74 ms 25 % in 15.42 ms 50 % in 19.90 ms 75 % in 24.92 ms 90 % in 29.75 ms 95 % in 32.97 ms 99 % in 39.91 ms
Bidi Stream	Summary	Count: 100000 Total: 109.99 s Slowest: 107.71 ms Fastest: 1.73 ms Average: 27.06 ms Requests/sec: 909.18	Count: 100000 Total: 309.82 s Slowest: 618.19 ms Fastest: 6.45 ms Average: 87.64 ms Requests/sec: 322.76	Count: 100000 Total: 97.59 s Slowest: 108.56 ms Fastest: 1.94 ms Average: 23.89 ms Requests/sec: 1024.71	Count: 100000 Total: 275.75 s Slowest: 254.31 ms Fastest: 5.00 ms Average: 77.73 ms Requests/sec: 362.65
	Latency distribution	10 % in 15.04 ms 25 % in 21.36 ms 50 % in 27.53 ms 75 % in 32.85 ms 90 % in 37.71 ms 95 % in 40.89 ms 99 % in 48.30 ms	10 % in 66.18 ms 25 % in 81.27 ms 50 % in 90.16 ms 75 % in 97.72 ms 90 % in 104.81 ms 95 % in 109.87 ms 99 % in 121.93 ms	10 % in 14.09 ms 25 % in 18.94 ms 50 % in 23.79 ms 75 % in 28.73 ms 90 % in 33.55 ms 95 % in 36.65 ms 99 % in 43.09 ms	10 % in 57.39 ms 25 % in 71.09 ms 50 % in 79.63 ms 75 % in 86.98 ms 90 % in 94.15 ms 95 % in 99.78 ms 99 % in 117.92 ms

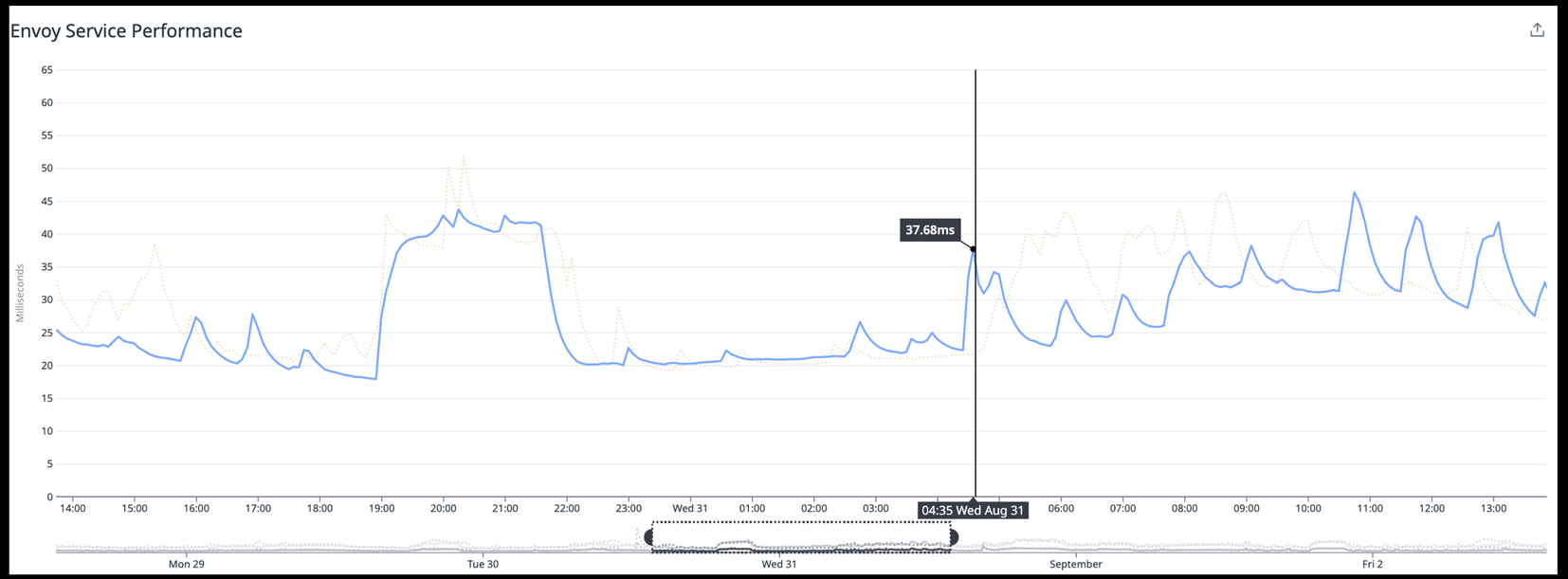
迁移步骤

- 更新rpclib-runtime使用Akka-gRPC
- 更新protos使用Akka-gRPC sbt plugin
- 利用Akka-gRPC的power apis重新实现MDC、interceptor机制
- 难点在于兼容现有老的代码
  - ScalaPB / rpclib-compiler / Akka-gRPC
  - 通过class renaming共存
  - 提供一个大版本的过渡期



结果

- 性能指标维持不变
  - latency基本维持不变
  - 内存和cpu基本维持不变
- 可维护性提升
  - 升级grpc依赖到新版
  - 代码可读性提高
  - Akka社区力量



Q&A and Thanks!