Building API: swagger, grpc, graphql What to choose, how to use, and how to survive

Daniel Podolsky, AnchorFree

Who am I

- Golang—evangelist and developer in AnchorFree.

Disclaimer

- This is a survey
- As light as possible
    - Because I hate surveys
- Not everything I’m talking about I’ve tried in prod
    - For the reason
- Captain Obvious says “Hi!”
    - Let’s read the manuals loudly!

Some theory: What is network API

- from the programming language point of view
  - Some glue code
  - Some tools
  - Documentation

Some theory: What is network API

- Pull, push, poll and events
  - From client to server: easy
  - From server to client: not so easy

Some theory: REST and RPC

- CRUD
  - Good
  - But not good enough
- RPC
  - Flexible
  - Sometime too much flexible

Some theory: What is network API, under the hood

- Call convention
- Marshaling
- Transport

Some theory: client side

- Go
- No so Go

Some practice: OpenAPI

- AKA swagger
- Born for REST
- Request-response + JSON + HTTP 1.1 + DSL + codegen/docgen
  - Yes, codegen!
  - There is more than one way to use it
    - But just one is a proper one

Some practice: OpenAPI, Tools, Go

- **`go-swagger`**
  - OpenAPI v2 only
  - Code quality is not so high
- **`oapi-codegen`**
  - OpenAPI v3, partially supported
  - Looks promising

Some practice: OpenAPI, other tools

- Swagger codegen
- Swagger editor
  - Good to have

Some practice: OpenAPI, Pros

- The standard
  - I mean it
- Native support in browsers
- Documentation is excellent

Some practice: OpenAPI, Cons

- JSON is slow in Go
  - Even with easyJSON
- No streams
  - No pushes, no events
- HTTP-bonded
- CRUD is tough
  - No way to keep your team on the CRUD way

Some practice: OpenAPI, How to survive

- As strict as possible
- Build your own RPC standard
  - And document it
- Always start with **`swagger.yml`**
  - I mean it
  - The only way to keep your code and doc in sync

Some practice: gRPC

- Request-response-push-events + protobuf + HTTP/2 + codegen
  - Yes, codegen!
  - Self-documented
    - Or pretend to be

Some practice: gRPC, Tools, Go

- `protoc`

Some practice: gRPC, other Tools

- uber/prototool
  - Must have
- grpc/grpc-web
- grpc-ecosystem/grpc-gateway
  - REST-to-gRPC reverse proxy
  - codegen

Some practice: gRPC, Pros

- The standard
- Client-server, server-client and bidirectional streams
- Reach and powerful ecosystem

Some practice: gRPC, Cons

- No streams native browser support
- No streams in non-native browser support
- Verification apart from type compatibility is on you
- HTTP/2-bonded
  - You can get some freedom with custom dialer

Some practice: gRPC, How to survive

- Use `prototool`
- Use `go generate` wisely
- Use `protoc.cfg`
- Use `option go_package`

Some practice: GraphQL

- I did not use it
  - For the reason

Some practice: GraphQL

- Freeeeedom!!!!!11
  - The main idea is to untight frontend changes delivery from backend changes delivery
    - As in “backend is ready to provide any data available, just ask”

Some practice: GraphQL

- Frontend oriented
- Request-response + JSON + HTTP 1.1 + DSL+ codegen (optional)
  - As a matter of fact GraphQL itself is transport and marshaler agnostic

Some practice: GraphQL, Tools, Go

- **`graphql-go`**
  - Runtime schema parser
- **`gqlgen`**
  - Codegen

Some practice: GraphQL, Tools, Go

- `graph-gophers/dataloader`
  - Cache
  - Dedup
  - Optimisation
  - Nothing like this
    in the other API builders!

Some practice: GraphQL, other Tools

- GraphQL API Explorer
- `graphql-playground`

Some practice: GraphQL, Pros

- Flexible
  - In all means
- Modern

Some practice: GraphQL, Cons

- Not a standard
  - Yet
- No streams
- No blobs (no way to return file to browser)
- Difficult to control the performance and security on backend side
  - Bad-formed query can explode your backend. Or evil-formed...

Some practice: GraphQL, how to survive

- Actually, I do not know
  - But this is a future, so be prepared

Some practice: GraphQL, how to survive

- Try query complexity analysis
  - query depth limit
  - query cost analysis
  - query whitelisting
    - No more freedom,
      no more flexibility

Some practice: GraphQL, how to survive

- Dataloaders

Some practice: GraphQL

- Special thanks to Roman Sharkov
  - tg: @Romshark

Some practice: Twirp

- I did not use it
- Request-response + protobuf/JSON + HTTP 1.1 + codegen
  - Codegen again
  - Self-documented
    - Or pretend to be

Some practice: Twirp, Tools

- Some of gRPC tools could be used

Some practice: Twirp, Pros

- Born for Go
- Simple
- HTTP 1.1
- Mixable with other handlers
- JS client

Some practice: Twirp, Cons

- Not a standard
- No streams
- Verification
  apart from type compatibility
  is on you

Some practice: Twirp, how to survive

- No idea

Some practice: Vulcain

- I did not use it
- Quite exotic
  - hypermedia API
  - Could be used as a gateway on top of traditional API.
- Frontend oriented
- Request-response-events + JSON + HTTP/2

Some practice: Vulcain, Tools

- No idea

Some practice: Vulcain, Pros

- Must be effective
  - Caching
  - Interrupting download
- GraphQL compatibility mode
- OpenAPI compatibility mode

Some practice: Vulcain, Cons

- Not a standard
- HTTP/2 bonded

Some practice: Vulcain, how to survive

- No idea

Some practice: JSON-RPC 2.0

- Absolutely perfect
  - As a spheroidal horse
    in the vacuum environment
- Very simple spec
  - Nothing useless
    - IMHO, nothing useful
- Request-response + JSON

Some practice: JSON-RPC 2.0, Tools, Go

- `net/rpc` plugin(s)

Some practice: JSON-RPC 2.0, Pros

- Simple
- Flexible

Some practice: JSON-RPC 2.0, Cons

- Cons
  - Not a standard
    - Not any more
  - No streams
  - No blobs (no way to return file to browser)
  - All the boilerplate except marshaling-unmarshaling is on you
    - Even doc

Some practice: JSON-RPC 2.0, how to survive

- Try to avoid
- So help you God

Conclusions

- To communicate with clients, right here, right now
  - OpenAPI
    - Best validators spec
    - Best doc generator

Conclusions

- To communicate between servers
  - gRPC
    - Streams

Conclusions

- For the future
  - GraphQL
    - Modern
    - Flexible

Thank you!

Questions
are welcome!

Daniel Podolsky
daniel@djarvur.net