Lessons from building an API Management Platform

Feb 14, 2020

@Marketplace

- Madan Thangavelu
- Ankit Srivastava

Uber

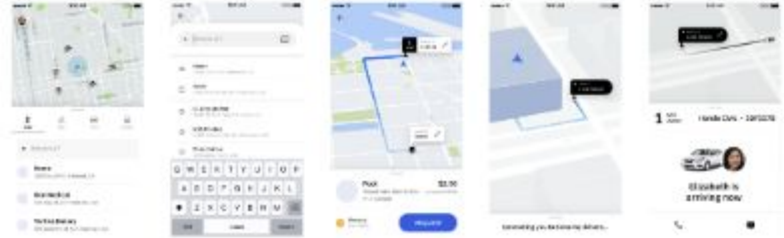
Agenda

01 API Gateway Features
02 Self Service API Gateway
03 Principles
04 Making Choices
05 Learnings

20+ Apps

1600+ APIs

3000+ services



API Gateway Features

Security

CORS
Data validation
Rate limit policies
Client access policies
Data encryption

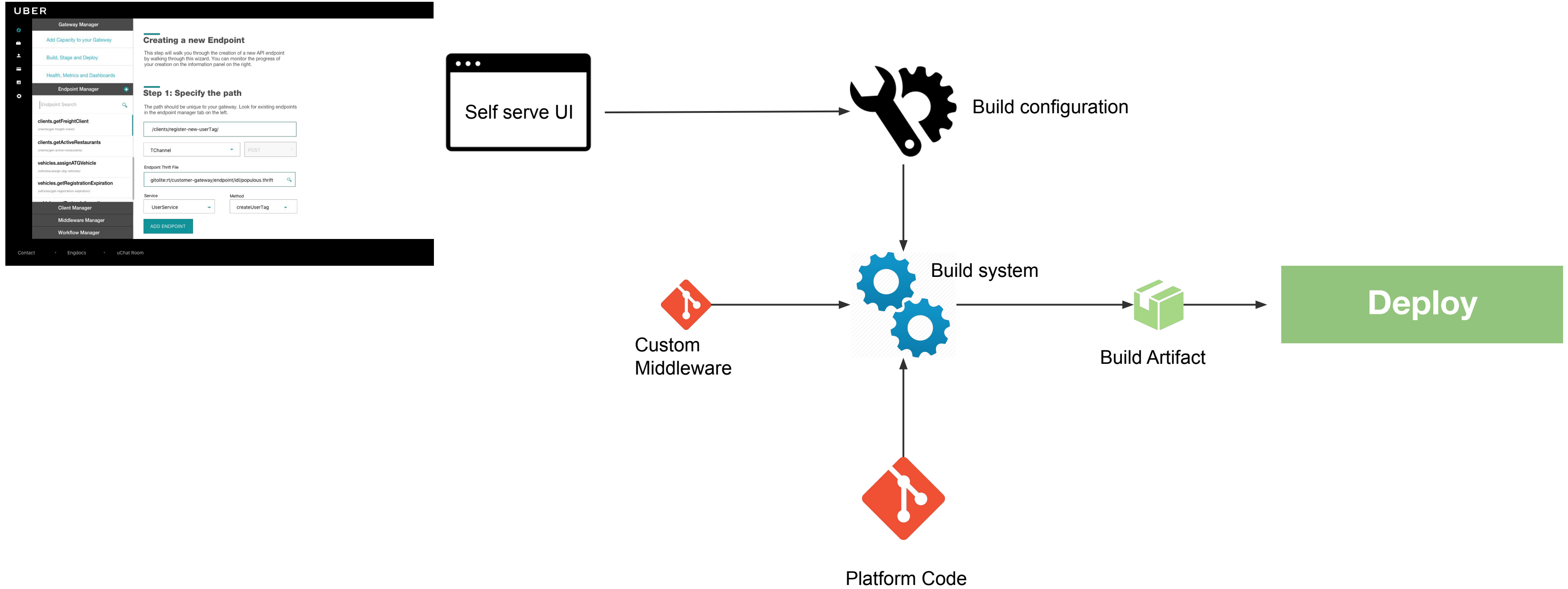
Operations

Tracing
Debugging knobs
Debugging tools
Auto alerts
Monitoring & SLA

API Management

API creation/modification
Request/Response Tx
Protocol transformations
Versioning
Migrations
SDK generations
Document generation

Self Service API Gateway



Principles

Reliability

- API isolation - one does not affect another
- Performance, Tooling and Documentation are first class citizens
- Bad code is prevented with automated processes

Build Artifact

- A build should be reproducible
- Deploy artifact should be 100% generated
- Local development possible

Features

- Less magical features preferred over feature bloat
- Easy to switch from magical automation to full control

Making choices

[1] Language

GoLang
NodeJs
Java

[2] Serialization

JSON <> JSON
JSON <> Thrift
Proto <> Thrift
Proto <> JSON

[3] Protocol

HTTP <> HTTP
HTTP <> Tchannel
HTTP <> gRPC
gRPC <> gRPC
gRPC <> HTTP

[4] Config Store

RDBMS
GIT

[5] RW Access

Incoming Headers
Incoming Body
Response Headers
Response Body

[6] Schema

OpenAPI
Thrift
Proto
YAML

[7] Migration

Move all 1600 APIs
Allow only new APIs

[8] Self-Serve

Fancy UI
Hand written config

[9] Scaling

Single binary
Multiple binary

[10] Features

Uber

API Gateway Platform

[1] Language

Pros

- Dynamic (node) to Static (golang) language
- Significantly performant ~ 1000 QPS/Core
- Aligned with company infrastructure

Cons

- A dynamic environment based on typed language
- No support of generics - too much codegen
- Large binary - compile time & hard failures
- Language naming conventions & reserved keywords
- Goimport non-determinism

[2] Serialization Format

Pros

- Rich & versatile gateway
- Proto (binary) ingress reduces payload size
- Migrate old APIs without migration

Cons

- Fundamental incompatibility in representations
  - Union/Set/List/Map Thrift representation in JSON & GoLang
  - UInt32Value, UInt64Value, Timestamp, Int32Value, in JSON?
- Versioning the payload
- Debug tooling for proto

[3] Protocol

Pros

- Rich gateway compatible with 99% of Uber protocols

Cons

- No application headers in Tchannel & gRPC
- No notion of REST verbs in gRPC
- Error conversion from one protocol to another
- HTTP -> gRPC transcends free form to typed requests
- gRPC Stream incompatible with other protocols
- The overhead to know the different compatibilities during config

[4] Config Store

Pros

- Clear version history & rollback
- Federation of config is straightforward
- Generated code & config live together
- Ability to test generated binary before config is committed

Cons

- Merge conflicts
- Slow to git commit and push (numerous systems)
- Hacky GIT binding with no native control from GoLang
- Slow checkout to apply a new config
- Polluted review of config with generated code

GIT

RDBMs

[5] RW Access

Pros

- Rich features
- Protocol conversion possible
  - Header to Body
  - Error mapping
- Complex middlewares
  - Filter fields, localization, rate limiting based on request

Cons

- Cost of serialization/deserialization
- Numerous bugs in mapping logic (e.g., nested list of map or sets)
- Magical auto-mapping based on field name resulted in bugs

[6] Schema

Pros

- Strict - a single field cannot be number & string like OpenAPI
- Expressive
- Annotation support to allow building complex representations

Cons

- Fundamental mismatch between schemas
    - Union, i64, i32 in thrift to JSON/HTTP
- No notion to represent Headers, Query, Path parameters
- What is ok in Thrift is not ok in GoLang
- Proto only services now needed a thrift file too (ouch)
- No support for meta validation like OpenAPI
- No doc generation support
- Nested imports are hard to resolve & maintain green build

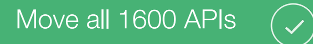
[7] Migration

Pros

- Deprecate the old API gateway with no maintenance overhead
- Significantly performant API gateway
- Validation that the new API gateway is feature rich
- Security & latest fixes
- An opportunity to deprecate

Cons

- 1.5 years of migration support & company wide effort
- Forced to support features we did not want in the new gateway



[8] Self-Serve

Pros

- Guides the user across tons of configurations & validations
- Fancy* & magical
- Less option for hacking features from customers (engineers)
- Feature rich - alerts, search fields, api docs, management
- Provided sessions for each user - sharable error states

Cons

- Batch/Multi edit flows became complex via the UI
- Surfacing complex errors was hard and at the mercy of the UI
- Dedicated frontend eng
- Introduced RDBMs for UI states & sessions

Fancy UI

Hand written config

[9] Scaling

Pros

- As we scale to 1000s of more APIs, we can deploy distinct binary that are smaller
- Faster isolated deploy for in development API binary

Cons

- Significant config overhead to resolve module dependencies
- Maintenance of multiple binaries

Multiple binary

Binary binary

[10] Features

Pros

- Extremely feature rich
  - Rate limit by body/query
  - Encrypt request
  - Encrypt response
  - Transform Request
  - Transform Response
  - Load shedding
  - Localization
  - Auth per API call
  - Streaming APIs
  - UI Features

Cons

- Development & maintenance overhead

Extensive features

Limited features

Here we are ...

[1] Language

GoLang
NodeJs
Java

[2] Serialization

JSON <> JSON
JSON <> Thrift
Proto <> Thrift
Proto <> JSON

[3] Protocol

HTTP <> HTTP
HTTP <> Tchannel
HTTP <> gRPC
gRPC <> gRPC
gRPC <> HTTP

[4] Config Store

GIT
RDBMS

[5] RW Access

Incoming Headers
Incoming Body
Response Headers
Response Body

[6] Schema

Thrift
Proto
OpenAPI
YAML

[7] Migration

Move all 1600 APIs
Allow only new APIs

[8] Self-Serve

Fancy UI
Hand written config

[9] Scaling

Multiple binary
Single binary

[10] Features

Uber

API Gateway Platform

Learnings

- Stick with a single protocol across the company
- Stick with a single serialization across the company
- Decision to serialize/deserialize is not for the faint hearts
- UI can significantly assist users but add lot of maintenance overhead
- Developers love faster iteration - choose the tech stack wisely
- GIT for configuration is a double edged sword
- There is no perfect programming language to build a API gateway system
- Scale :

1.5k

endpoints

1M rps

requests to the API layer

Thank you!

- Madan Thangavelu
  @hackinghabits
- Ankit Srivastava