OZON tech

Пишем микросервисы на Go как в BigTech с нуля

Леонид Ченский
Руководитель команды DBA Scylla

Леонид Ченский

LinkedIn

lmoguchev@ozon.ru

Полгода в Ozon Платформа

Team Lead

- Департамент DBaaS, команда Scylla DBA

3 года в Ozon Логистика

Middle Golang Developer → Team Lead

- Сервисы тарификации: расчёт доступности товаров для пользователя, сроков доставки и подбор способов доставки

Веду свои курсы и блог

https://t.me/leoscode

AGENDA

01 Трудности создания MSA с нуля

02 gRPC и gRPC-Gateway

03 Buf

04 rk-boot

05 Примеры

06 Заключение

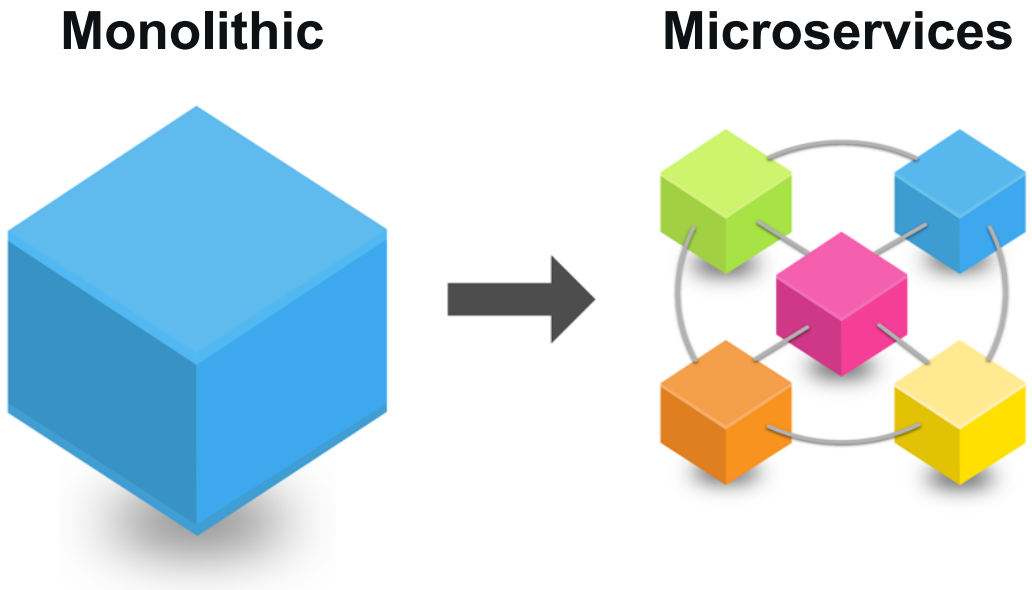
ozontech

01

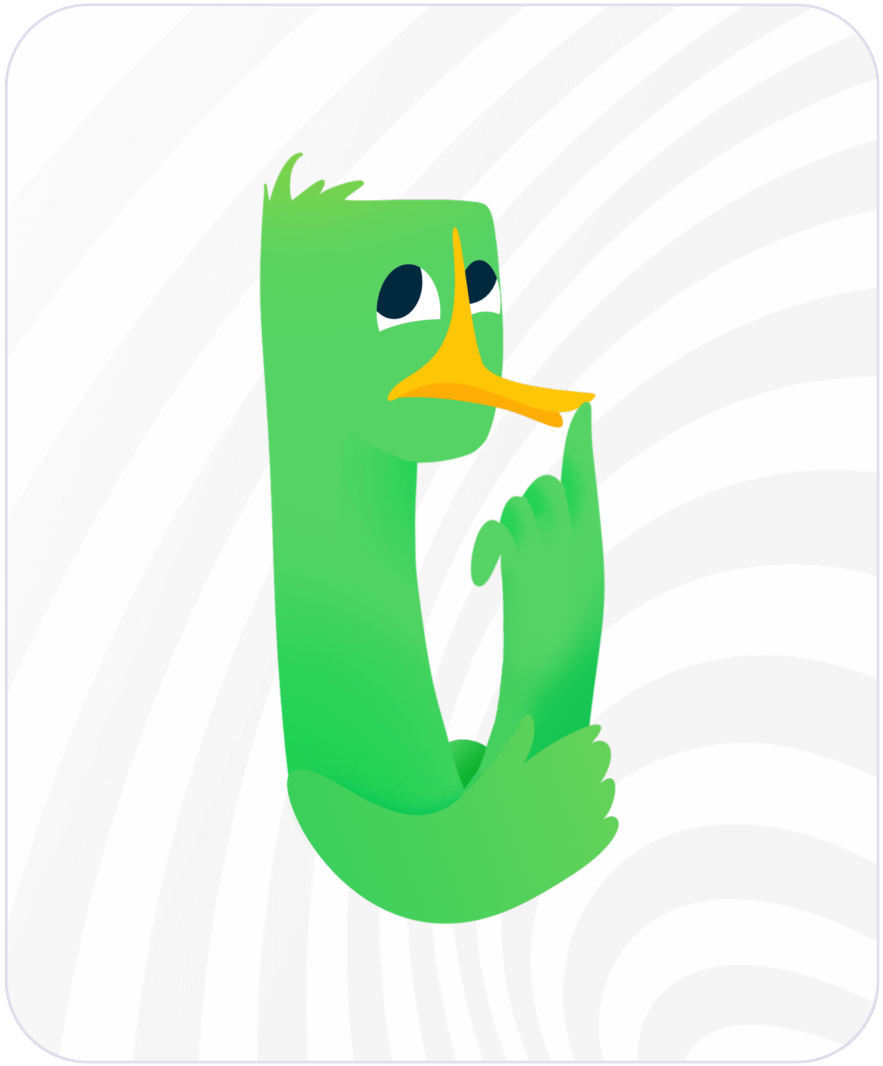
Трудности создания микросервисов с нуля

Почему микросервисы стали стандартом в BigTech?

- Разделение больших системы на независимые, легко управляемые компоненты
- Параллельная разработка над отдельными частями системы
- Большая отказоустойчивость

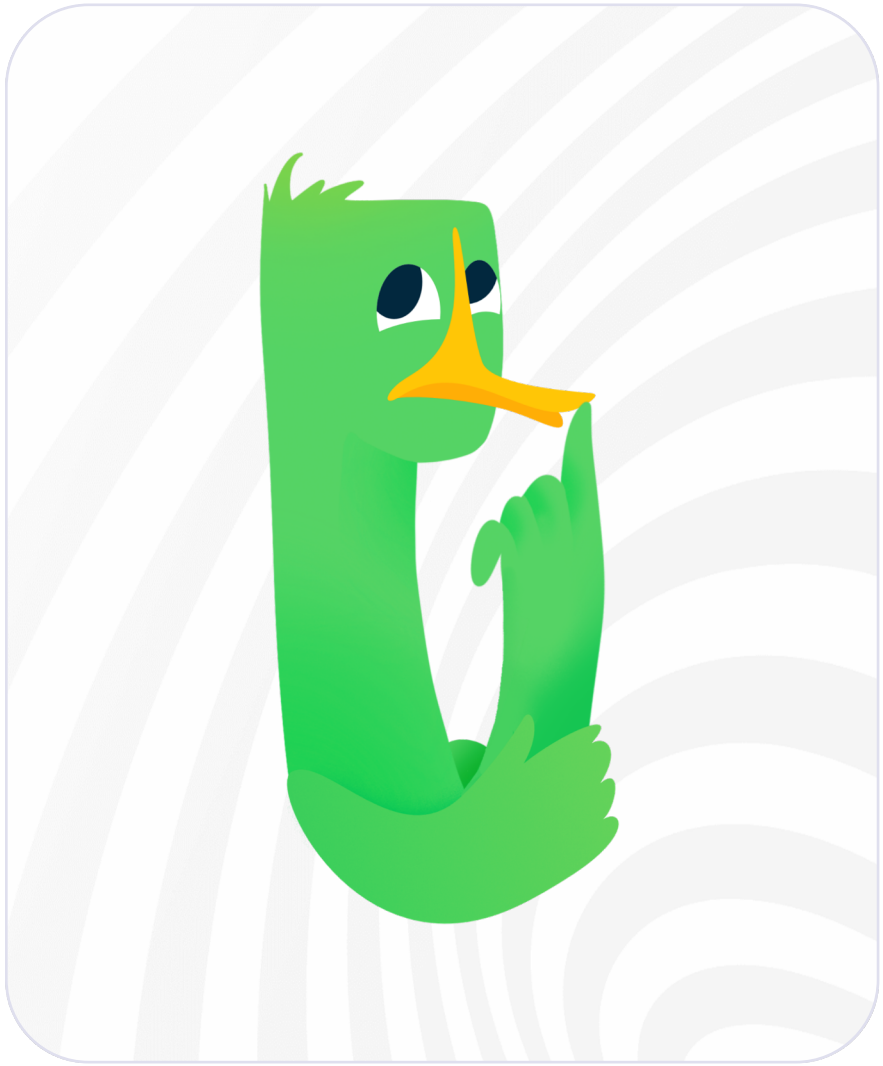


Трудности создания микросервисов с нуля



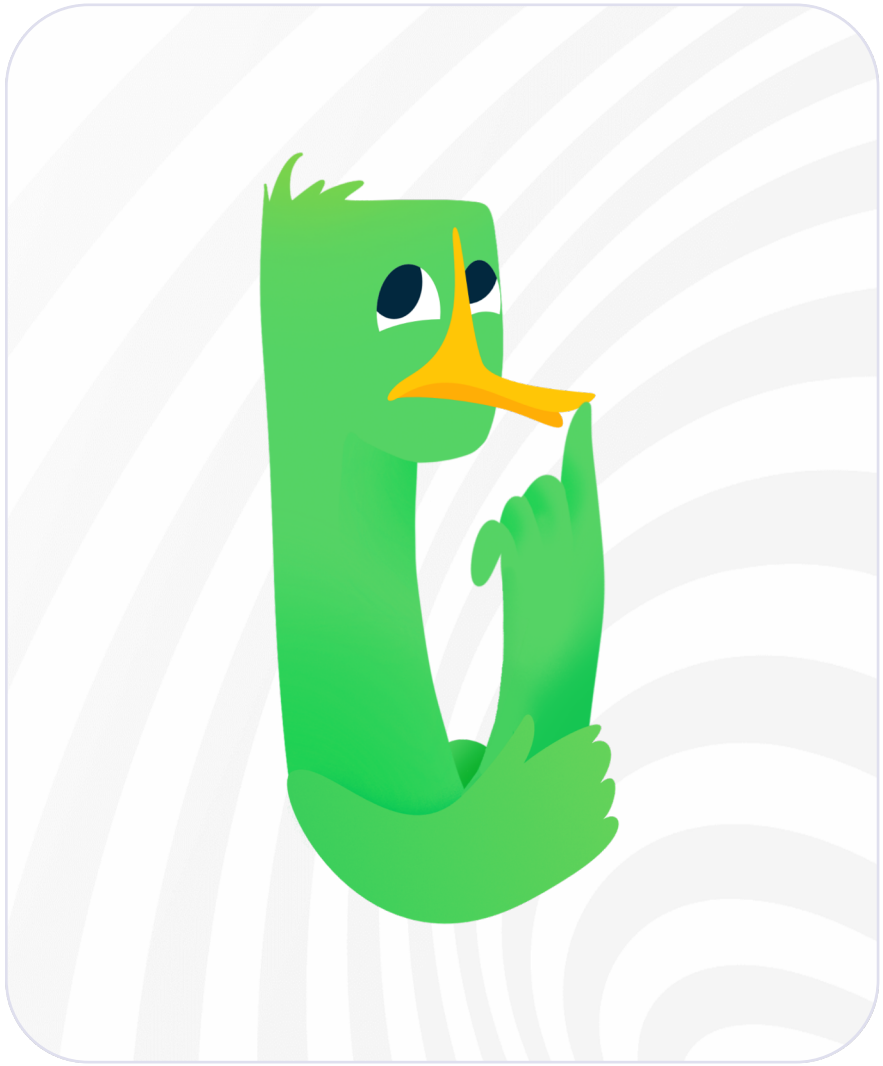
1. Межсервисное взаимодействие

Трудности создания микросервисов с нуля



1. Межсервисное взаимодействие
2. Долгое время разработки с нуля

Трудности создания микросервисов с нуля



1. Межсервисное взаимодействие
2. Долгое время разработки с нуля
3. Сложности в поддержке

Как в BigTech компаниях удается легко работать с MSA?

Как в BigTech компаниях удается легко работать с MSA?

01

Соблюдение единых стандартов разработки

Как в BigTech компаниях удается легко работать с MSA?

01

Соблюдение единых стандартов разработки

02

Автоматизация рутинных задач

Как в BigTech компаниях удается легко работать с MSA?

01

Соблюдение единых стандартов разработки

02

Автоматизация рутинных задач

03

Уделение внимания долгосрочной поддержке сервисов

Как в BigTech компаниях удается легко работать с MSA?

01

Соблюдение единых стандартов разработки

02

Автоматизация рутинных задач

03

Уделение внимания долгосрочной поддержке сервисов

04

Использование высокопроизводительных решений

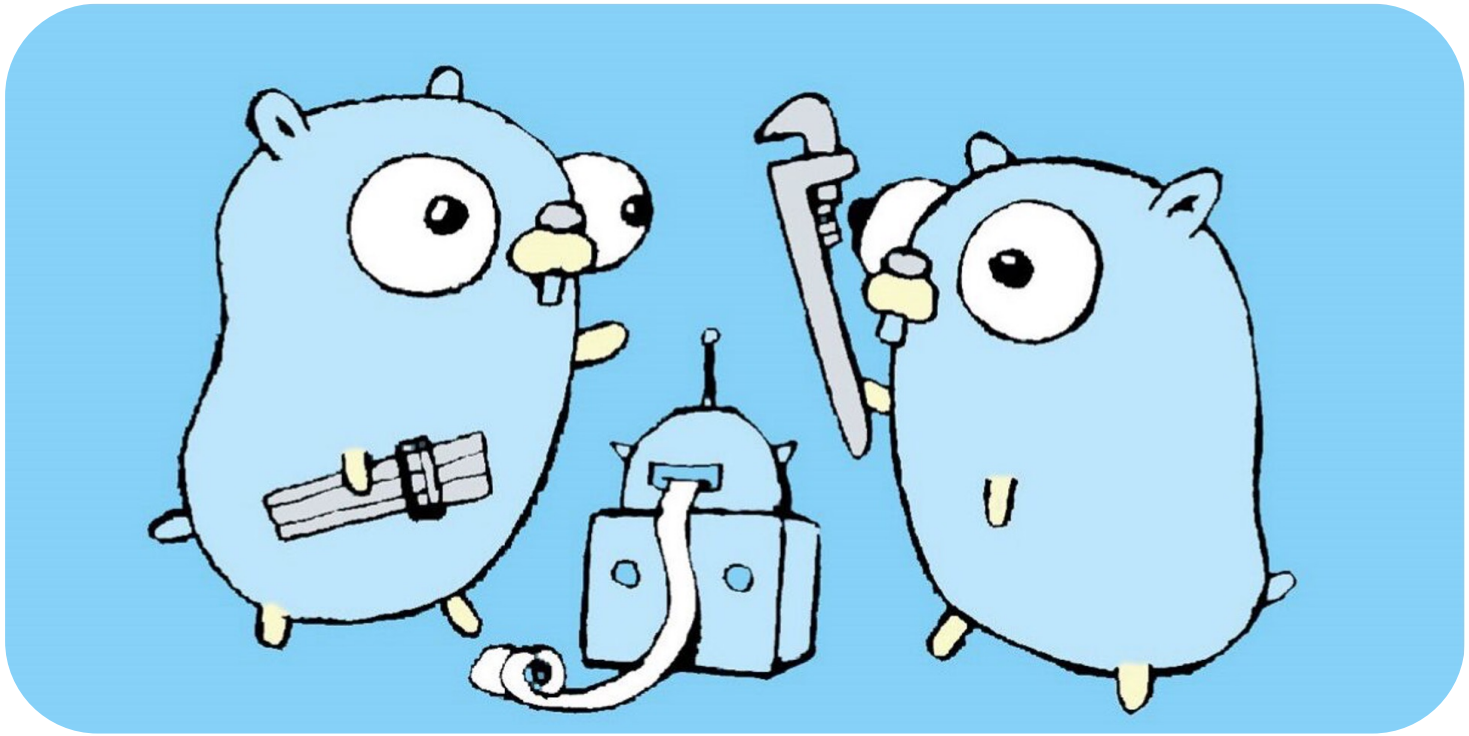
Требования к инструментам и библиотекам

Свойства

- Универсальность и стандартизация
- Высокая производительность
- Масштабируемость и простая поддержка

Цель доклада

Показать, как быстро и эффективно стартовать с микросервисной архитектурой на Go, используя современные opensource инструменты

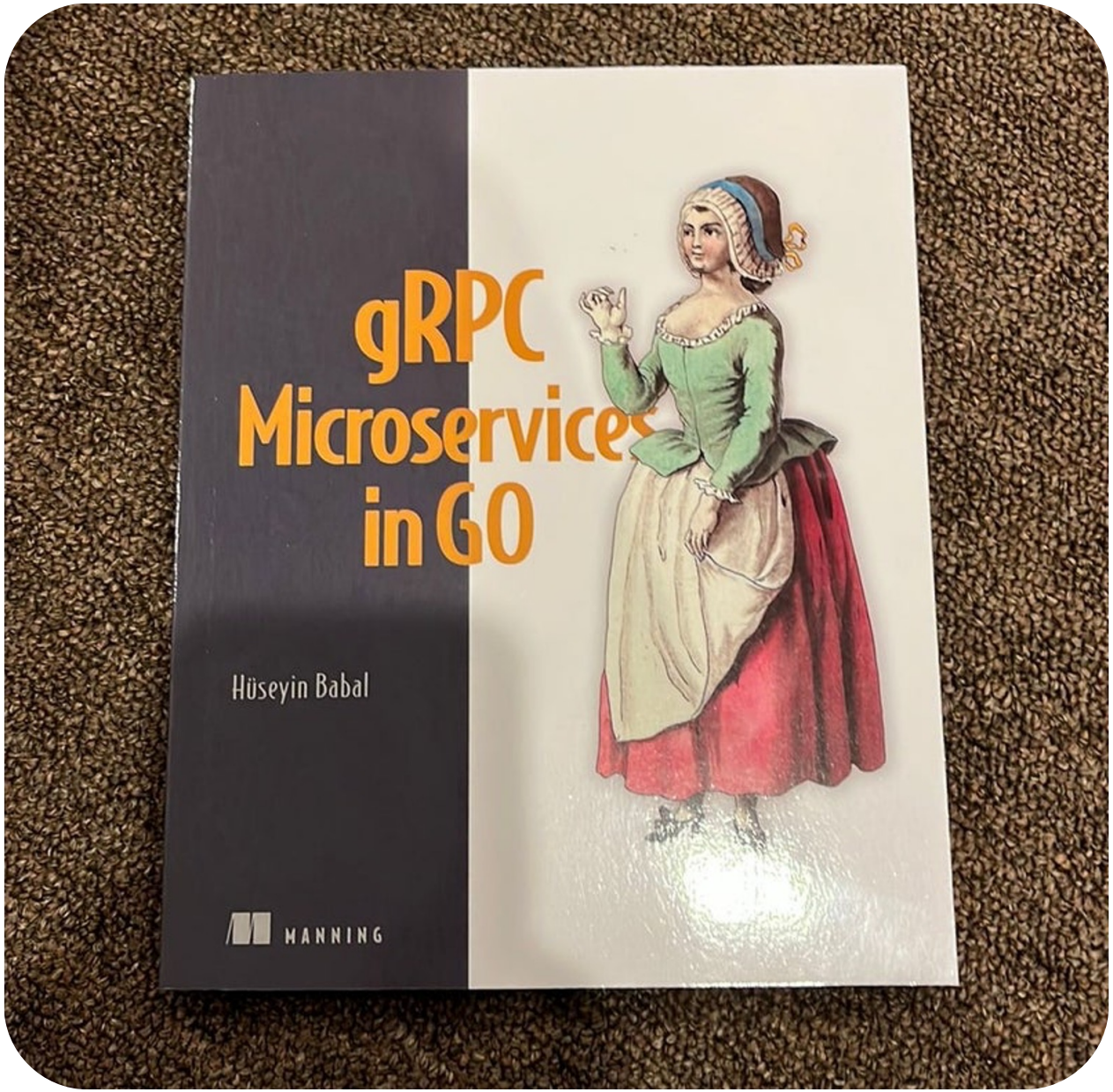


02 gRPC

gRPC

gRPC (Google Remote Procedure Call) —

это высокопроизводительный фреймворк, представляющий собой фреймворк удалённого вызова процедур (RPC), разработанный Google



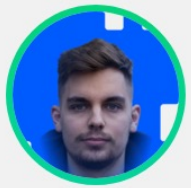
https://grpc.io/docs/languages/go/quickstart

Другие доклады на тему Protobuf и gRPC

Protobuf в Go

Владислав Сидоров
Ozon

Облегчаем жизнь разработчикам при помощи плагинов protoc



Святослав Петров
Ozon

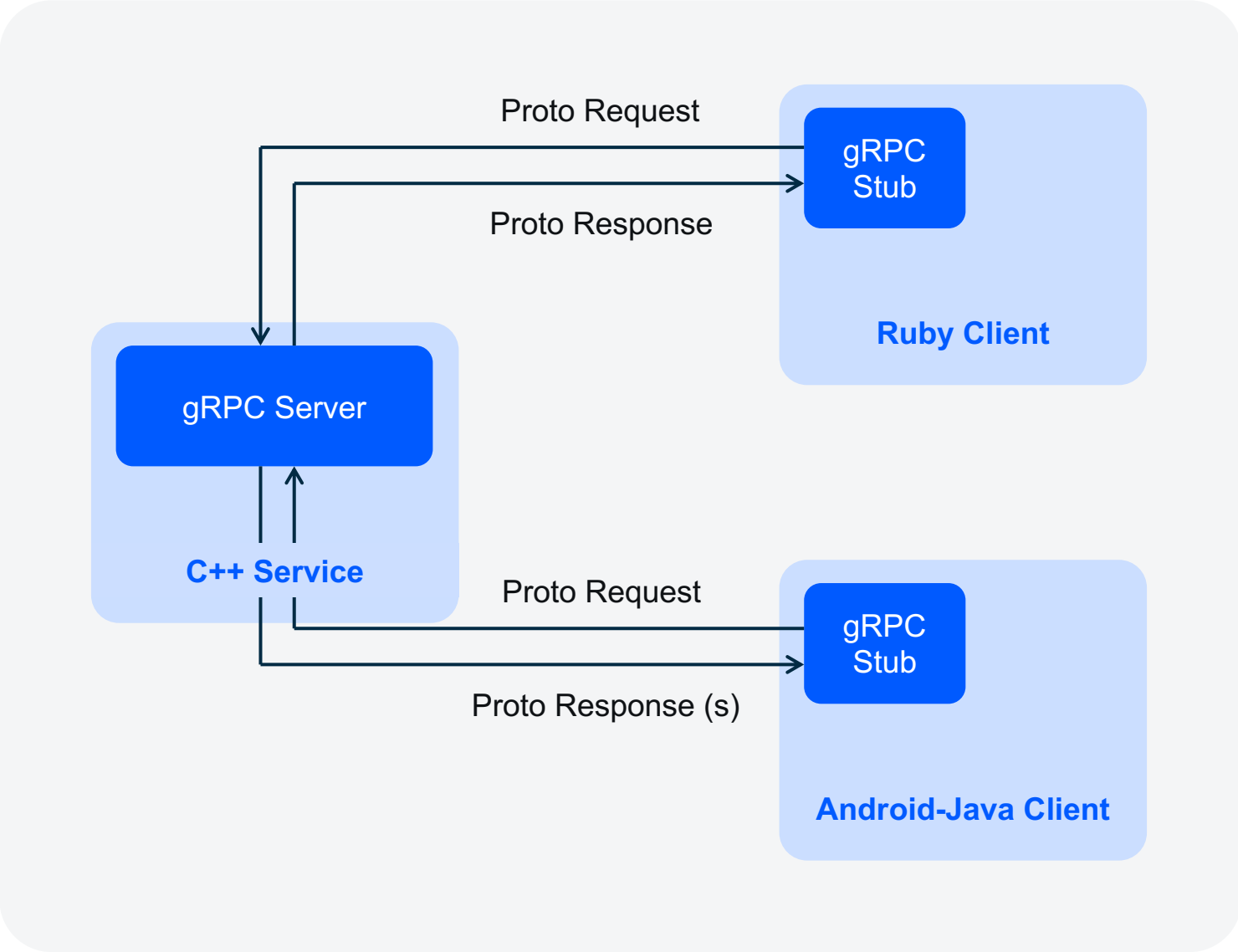
Универсальность и стандартизация

01 **Protocol Buffers** – строгая типизация

Универсальность и стандартизация

01 Protocol Buffers – строгая типизация

02 **Мультиплатформенность**


Proto Request
gRPC Stub
Proto Response
Ruby Client
gRPC Server
C++ Service
Proto Request
gRPC Stub
Proto Response (s)
Android-Java Client

Универсальность и стандартизация

01 Protocol Buffers – строгая типизация

02 Мультиплатформенность

03 Встроенные механизмы SSL/TLS

Встроенные механизмы SSL/TLS

```
// TLS/SSL
tlsConfig := &tls.Config{ /* ... */ }
grpcServer := grpc.NewServer(
    grpc.Creds(credentials.NewTLS(tlsConfig)),
)
```

Высокая производительность

01 Protocol Buffers – бинарный протокол передачи данных и HTTP/2

Высокая производительность

01 Protocol Buffers – бинарный протокол передачи данных

02 Поддержка bi-directional streaming

Масштабируемость и простая поддержка

01 Автоматическая генерация кода

Автоматическая генерация кода

```go
const GRPC_PORT = ":80"

func main() {
    lis, err := net.Listen("tcp", GRPC_PORT)
    if err != nil {
        log.Fatalf("failed to listen: %v", err)
    }

    srv := examplev1.NewExampleServiceServerImplementation()

    // gRPC
    grpcServer := grpc.NewServer()
    pb.RegisterExampleServiceServer(grpcServer, srv)

    reflection.Register(grpcServer)

    if err := grpcServer.Serve(lis); err != nil {
        log.Fatalf("serve: %v", err)
    }
}
```

Масштабируемость и простая поддержка

01 Автоматическая генерация кода

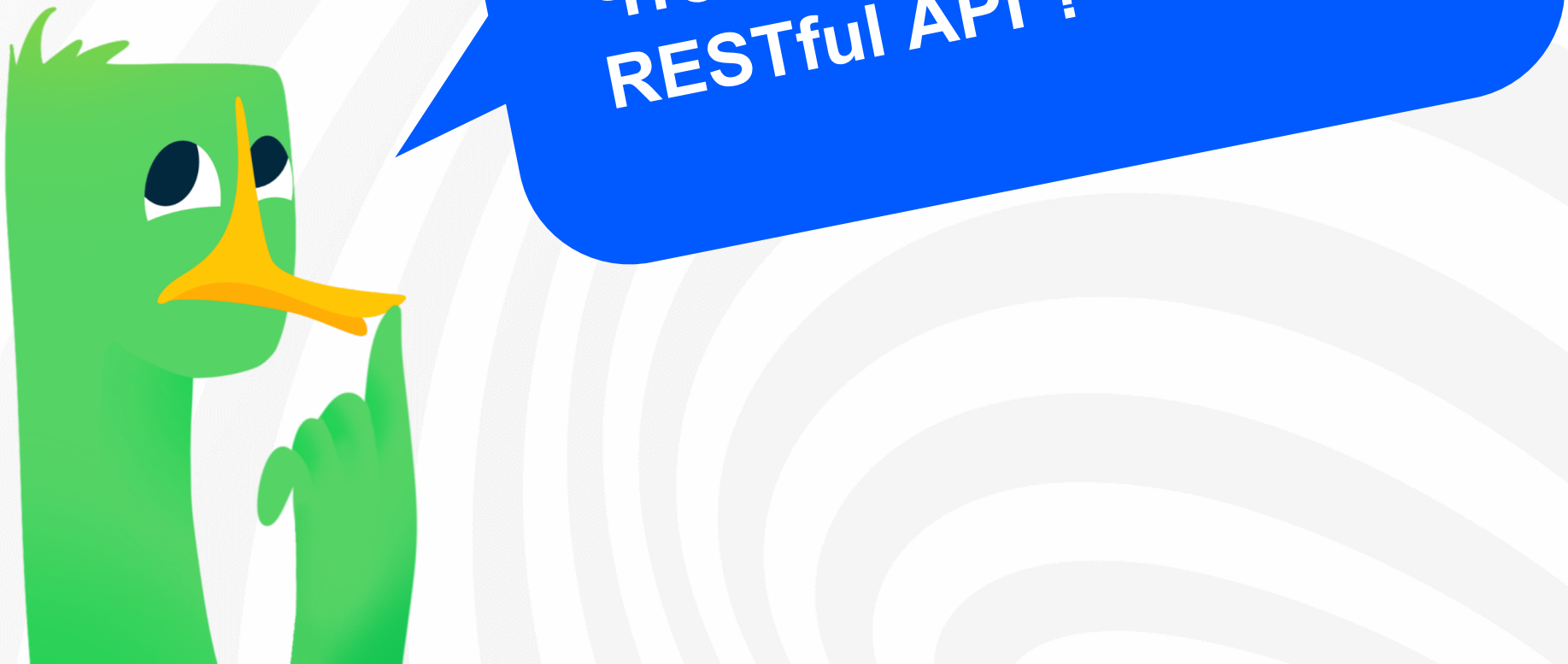
02 gRPC – расширяемый

Автогенерация валидации gRPC-запросов

```proto
import "buf/validate/validate.proto"; // внешние зависимости (buf.validate)

// Note - note message
message Note {
  // title - название заметки
  string title = 1 [
    (buf.validate.field).string = {
      min_len: 3
      max_len: 256
    },
  ];

  // ...
}
```

Автогенерация валидации gRPC-запросов

```go
// WithProtovalidateUnaryServerInterceptor - Валидация (Вариант 2)
func WithProtovalidateUnaryServerInterceptor(v *protovalidate.Validator) grpc.UnaryServerInterceptor {
    return func(ctx context.Context, req any,
        _ *grpc.UnaryServerInfo, handler grpc.UnaryHandler) (any, error) {

        msg, ok := req.(proto.Message)
        if ok {
            if err := v.Validate(msg); err != nil {
                return nil, status.Error(codes.InvalidArgument, err.Error())
            }
        }

        return handler(ctx, req)
    }
}
```

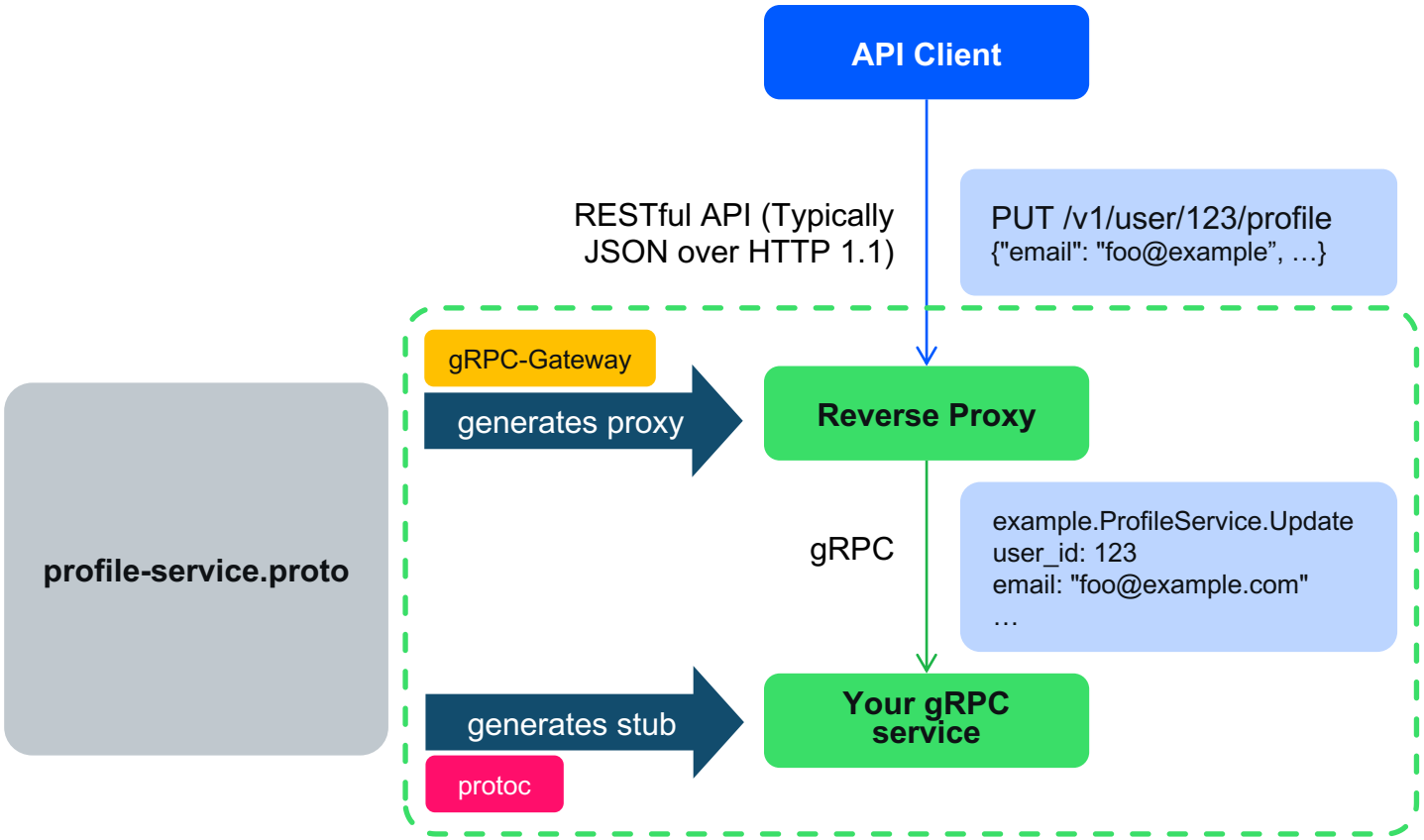

Что на счёт поддержки RESTful API ?

02

gRPC-Gateway

gRPC-Gateway

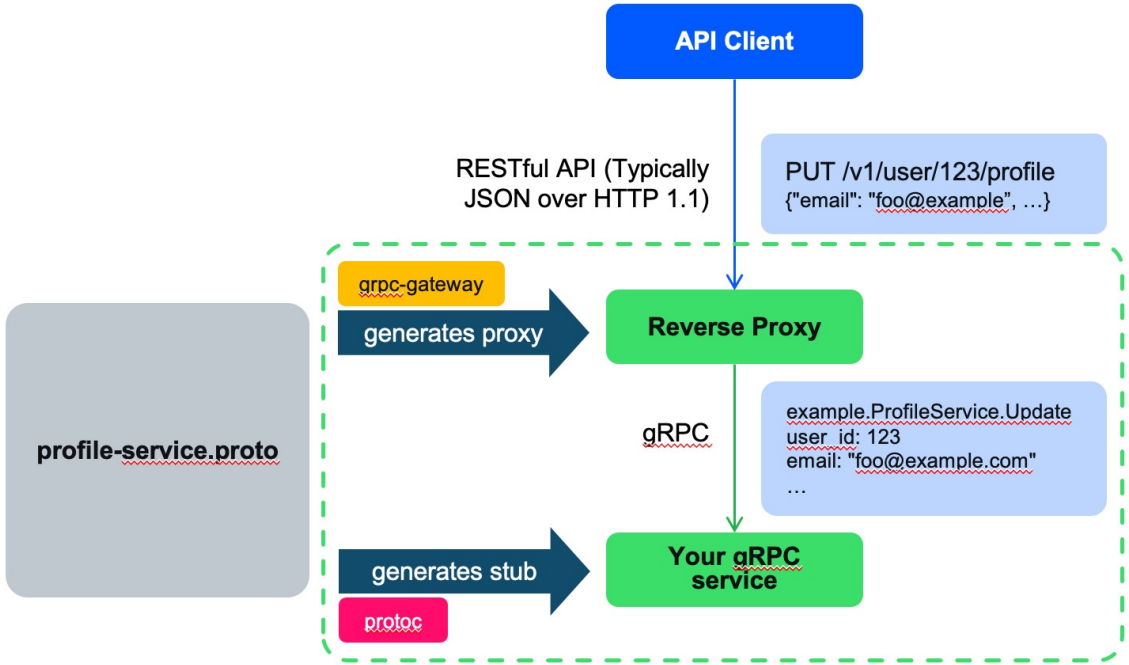
gRPC-Gateway — это инструмент, который позволяет автоматически генерировать RESTful API из gRPC сервисов



gRPC-Gateway

Плюсы

- **Обеспечение совместимости с существующими RESTful клиентами**



gRPC-Gateway

Плюсы

- Обеспечение совместимости с существующими RESTful клиентами
- **gRPC-Gateway автоматически генерирует RESTful API из существующих gRPC сервисов**

gRPC-Gateway

```go
// TLS/SSL
tlsConfig := &tls.Config{ /* ... */ }

srv := examplev1.NewExampleServiceServerImplementation()

// gRPC-Gateway
mux := runtime.NewServeMux()
err = pb.RegisterExampleServiceHandlerServer(context.TODO(), mux, srv)
if err != nil {
    log.Fatalf("failed to register handler: %v", err)
}
httpServer := &http.Server{
    Handler:   mux,
    Addr:      GRPC_PORT,
    TLSConfig: tlsConfig,
}
go httpServer.ListenAndServe()
```

gRPC-Gateway

Плюсы

- Обеспечение совместимости с существующими RESTful клиентами
- gRPC-Gateway автоматически генерирует RESTful API из существующих gRPC сервисов
- **Единая реализация бизнес-логики**

gRPC-Gateway

Плюсы

- Обеспечение совместимости с существующими RESTful клиентами
- gRPC-Gateway автоматически генерирует RESTful API из существующих gRPC сервисов
- Единая реализация бизнес-логики
- **Поддержка Swagger/OpenAPI**

Генерация OpenAPI

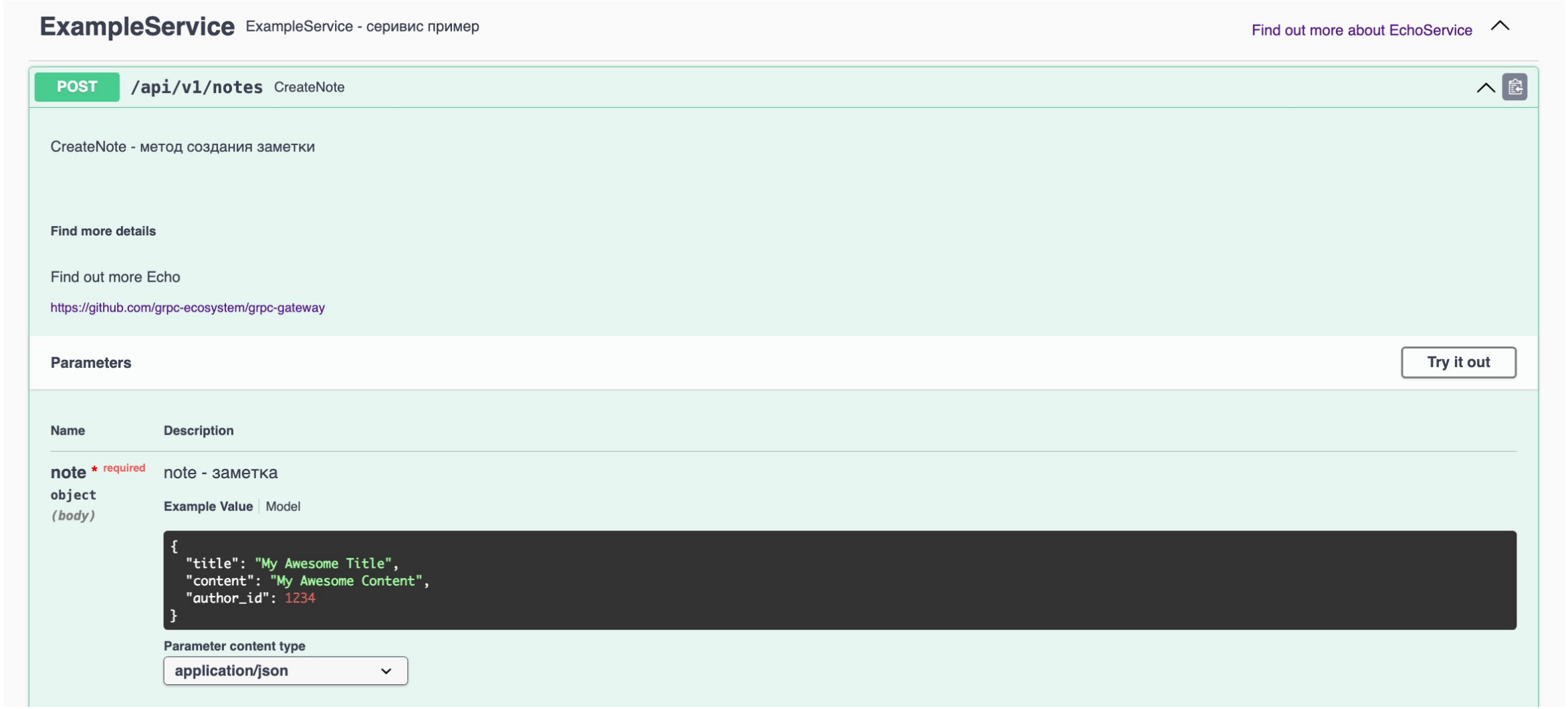
```
import "google/api/annotations.proto";
import "protoc-gen-openapiv2/options/annotations.proto";

// ExampleService - серивис пример
service ExampleService {
  // CreateNote - метод создания заметки
  rpc CreateNote(CreateNoteRequest) returns (CreateNoteResponse) {
    option (google.api.http) = {// from "google/api/annotations.proto"
      post: "/api/v1/notes"
      body: "note"
    };
    option (grpc.gateway.protoc_gen_openapiv2.options.openapiv2_operation) = {//
from "protoc-gen-openapiv2/options/annotations.proto"
      description: "CreateNote - метод создания заметки"
      summary: "CreateNote"
      external_docs: {
        url: "https://github.com/grpc-ecosystem/grpc-gateway"
        description: "Find out more Echo"
      }
    };
  }
}
```

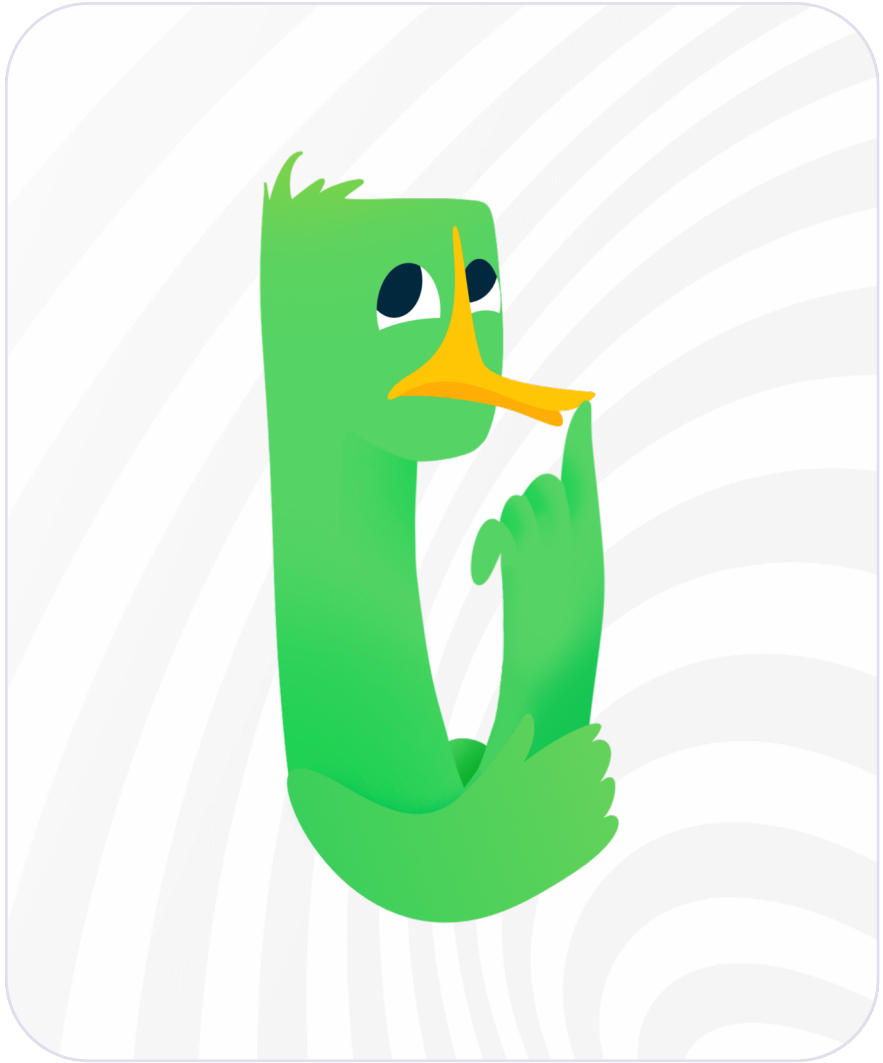
https://github.com/grpc-ecosystem/grpc-gateway/blob/main/examples/internal/proto/examplepb/a_bit_of_everything.proto

Генерация OpenAPI

```
// title - название заметки
string title = 1 [
  json_name = "title",
  (google.api.field_behavior) = REQUIRED,
  (buf.validate.field).string = {
    min_len: 3
    max_len: 256
  },
  (grpc.gateway.protoc_gen_openapiv2.options.openapiv2_field) = {
    title: "title"
    description: "название заметки"
    example: "\"My Awesome Title\""
    min_length: 3
    max_length: 256
    pattern: "^[0-9a-zA-Z]$"
    type: STRING
    format: "string"
  }
];
```

Генерация OpenAPI



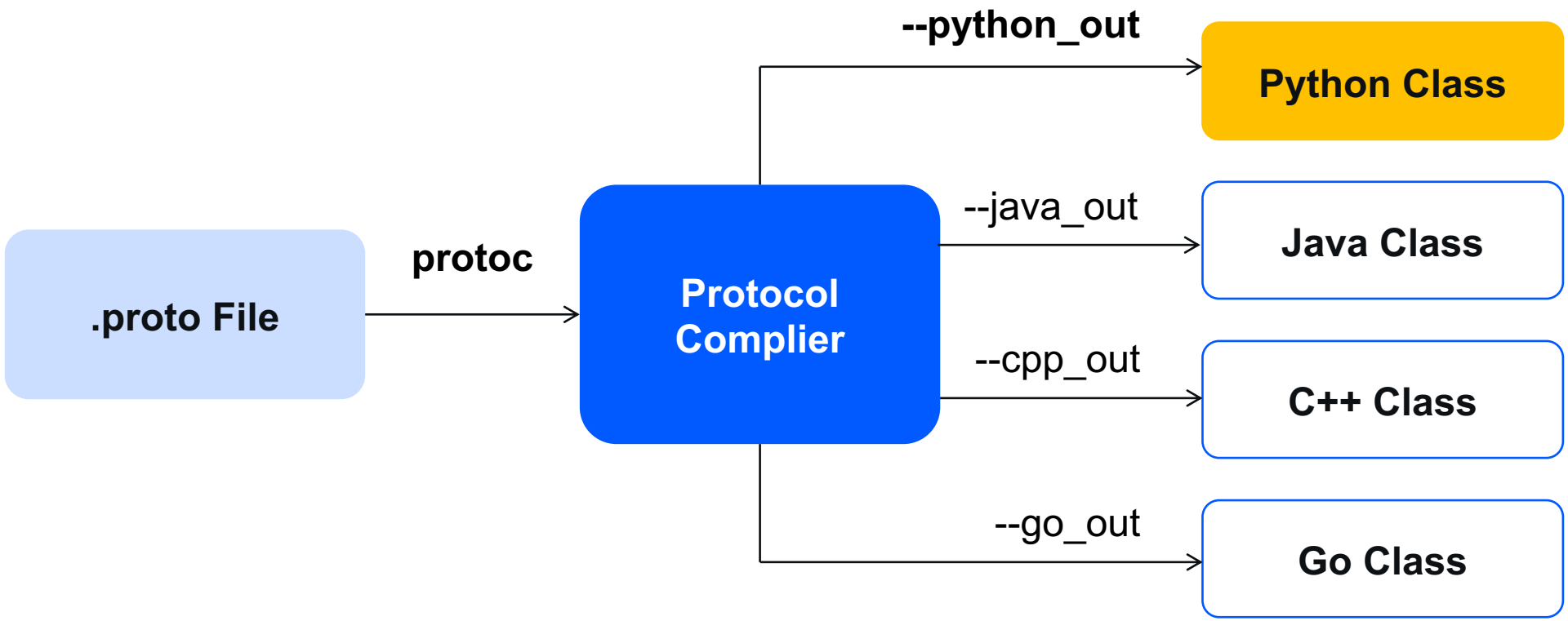


Так ли проста **генерация кода**?

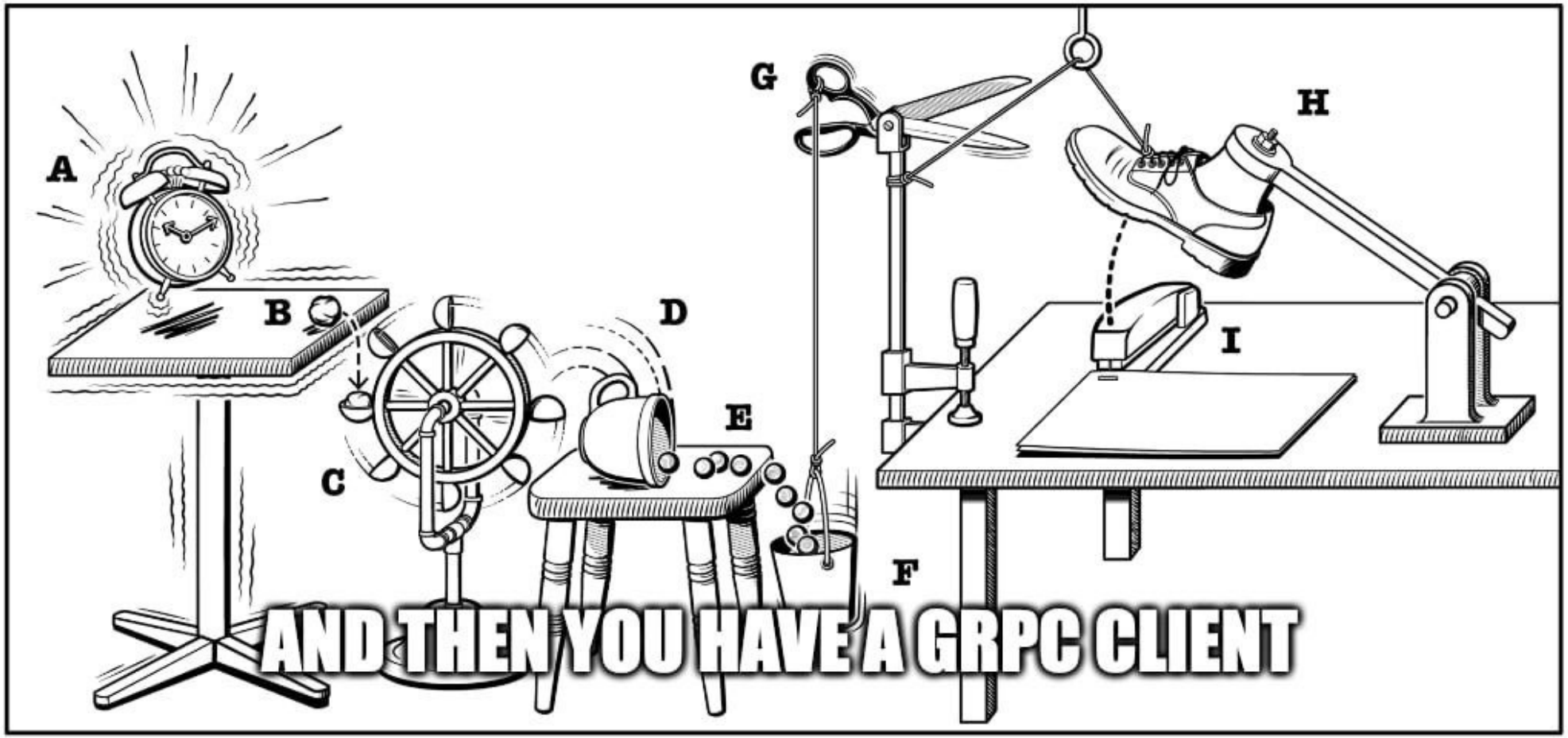
Что насчет **линтинга** и **форматирования protobuf**?

А как **версионировать схемы** и **работать с зависимостями**?

protoc


.proto File
protoc
Protocol Complier
--python_out
Python Class
--java_out
Java Class
--cpp_out
C++ Class
--go_out
Go Class

protoc



03 Buf

gRPC

Buf — набор инструментов для работы с Protocol Buffers (protobuf), разработанный для улучшения управления схемами и API в проектах, использующих gRPC и другие технологии, основанные на protobuf.

Какие проблемы решает Buf?

01

Оптимизация генерации кода

Генерация кода с помощью Buf

Устанавливаем Buf

```
go install github.com/bufbuild/buf/cmd/buf@v1.36.0
```

Генерация кода с помощью Buf

Инициализируем конфиг

Генерация кода с помощью Buf

Указываем путь до наших protobuf файлов в ***buf.yaml***

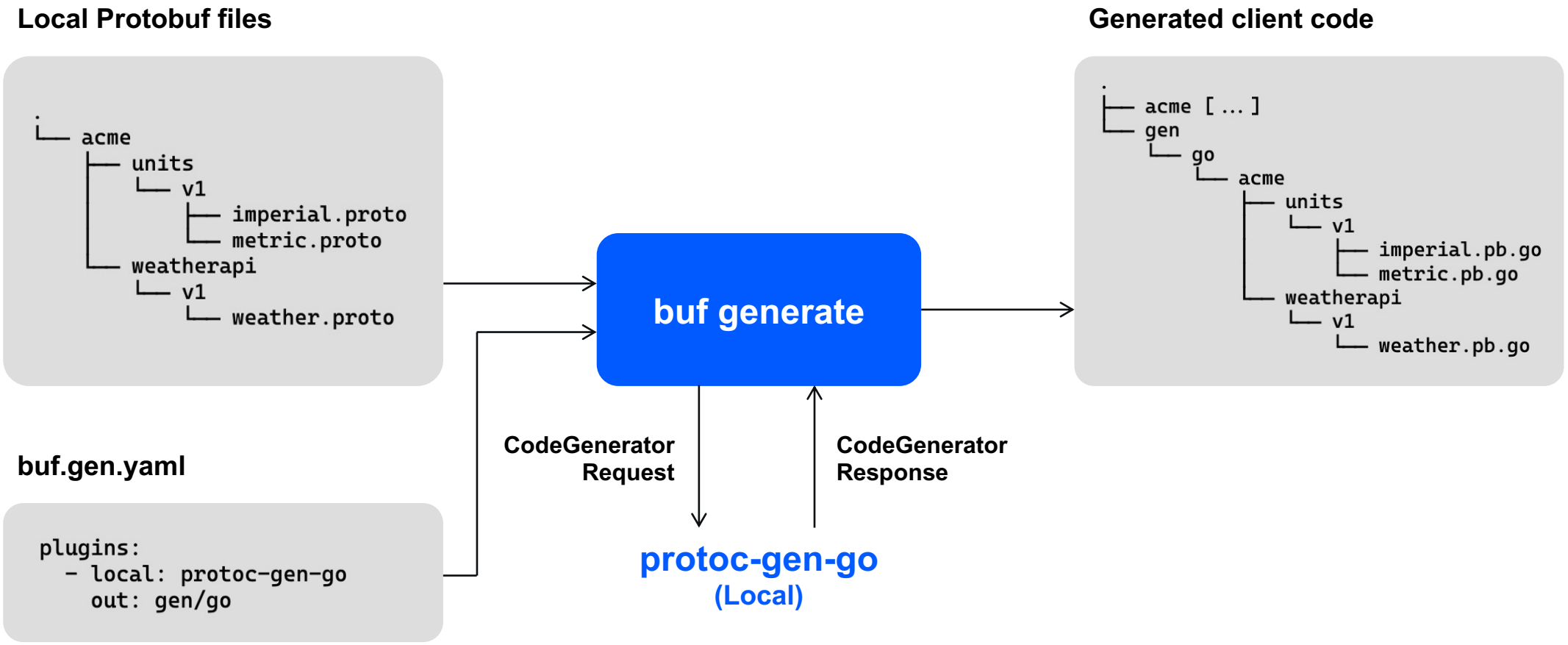
```yaml
version: v2
modules:
  - path: proto
```

Генерация кода с помощью Buf

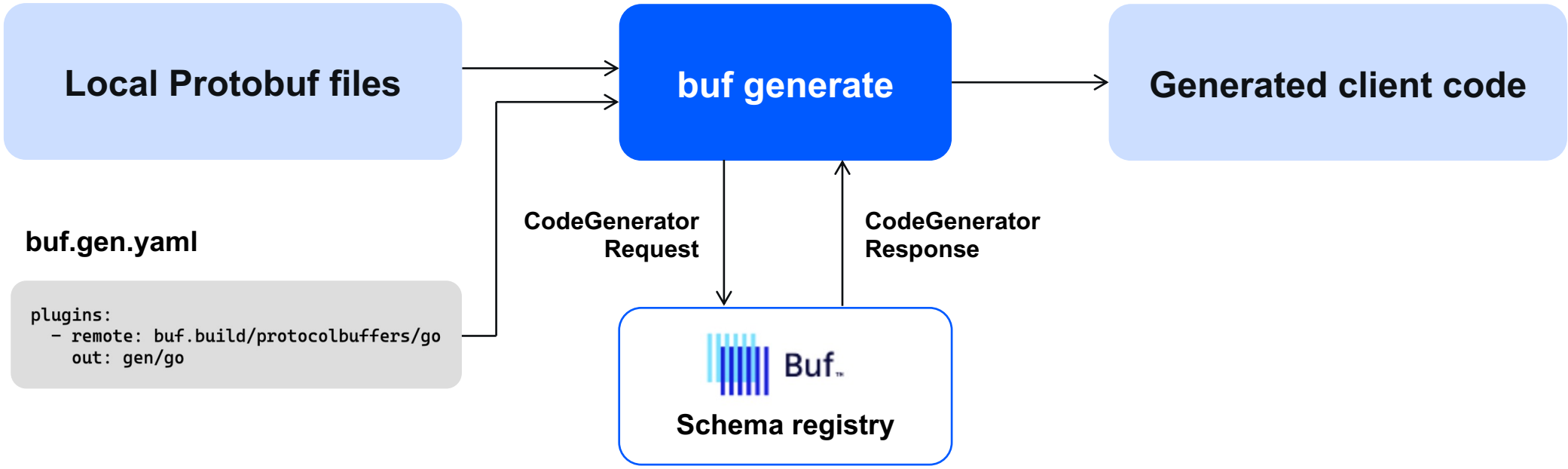
Создадим конфигурацию ***buf.gen.yaml*** для генерации кода

```yaml
version: v2
managed:
  enabled: true
  disable:
    - path: vendor.protobuf
plugins:
  - local: protoc-gen-go
    out: pkg
    opt: [...]
  - local: protoc-gen-go-grpc
    out: pkg
    opt: [...]
  - local: protoc-gen-grpc-gateway
    out: pkg
    opt: [...]
  - local: protoc-gen-openapiv2
    out: swagger
    strategy: all
    opt: [...]
inputs:
  - directory: proto
```

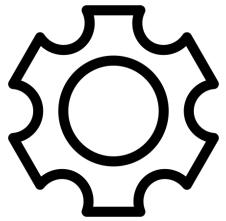
Генерация кода с помощью **local plugins**



Генерация кода с помощью **remote plugins**



Работаем с protobuf-зависимостями

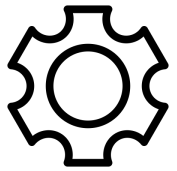


Вендоринг protobuf файлов

- Рутинная операция, для которой придется писать свои скрипты и самим отслеживать версии

Работаем с protobuf зависимостями (vendor)

Вручную (или с помощью скриптов) скачиваем необходимые protobuf-файлы

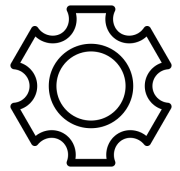


```
# Устанавливаем proto описания google/protobuf
.vendor-google-protobuf:
    git clone -b main --single-branch -n --depth=1 --filter=tree:0 \
        https://github.com/protocolbuffers/protobuf $(VENDOR_PROTO_PATH)/protobuf &&\
    cd $(VENDOR_PROTO_PATH)/protobuf &&\
    git sparse-checkout set --no-cone src/google/protobuf &&\
    git checkout
    mkdir -p $(VENDOR_PROTO_PATH)/google/protobuf
    find $(VENDOR_PROTO_PATH)/protobuf/src/google/protobuf -maxdepth 1 \
    -type f -exec mv {} $(VENDOR_PROTO_PATH)/google/protobuf \;
    rm -rf $(VENDOR_PROTO_PATH)/protobuf
```

Пример

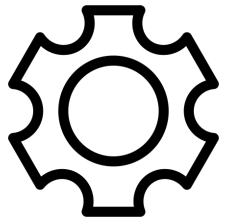
Работаем с protobuf зависимостями (vendor)

Указываем путь до скаченных protobuf файлов в ***buf.yaml***



```
modules:
  - path: proto
  - path: vendor.protobuf
```

Работаем с protobuf-зависимостями



Вендоринг protobuf файлов

- Рутинная операция, для которой придется писать свои скрипты и самим отслеживать версии

Buf

Buf Schema Registry

- Централизованный реестр protobuf схем
- Недоступен в России

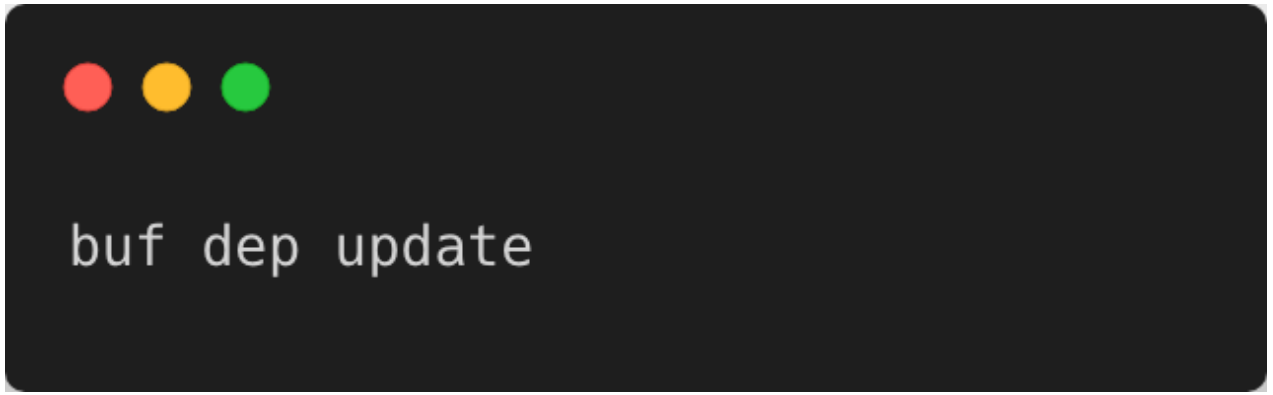
Устанавливаем protobuf зависимости (BSR)

Указываем зависимости из Buf Schema Registry (BSR) в ***buf.yaml***

```
deps:
  - buf.build/googleapis/googleapis
  - buf.build/grpc-ecosystem/grpc-gateway
  - buf.build/bufbuild/protovalidate
```

Устанавливаем protobuf зависимости (BSR)

Фиксируем protobuf-зависимости нашего модуля

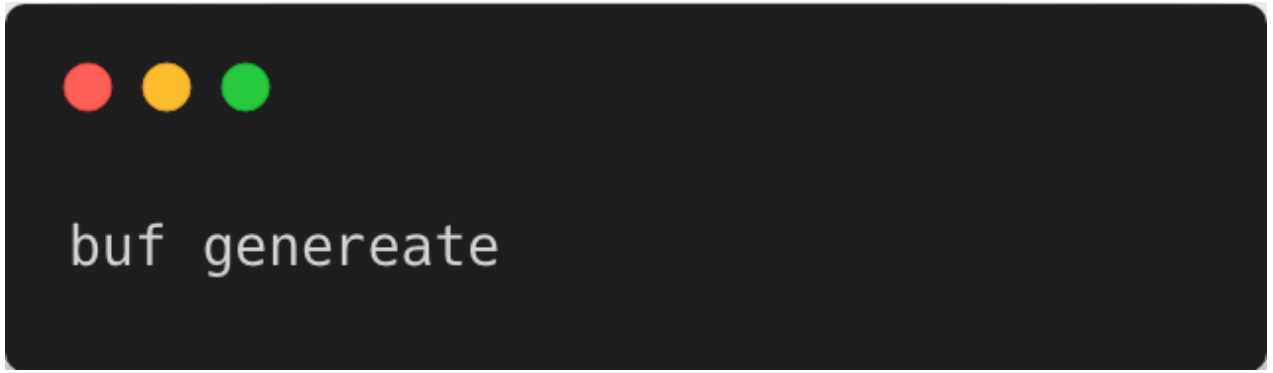


Устанавливаем protobuf зависимости (BSR)

Проверяем, что все необходимые зависимости корректно установлены

Устанавливаем protobuf зависимости (BSR)

Генерация Go-кода



Какие проблемы решает Buf?

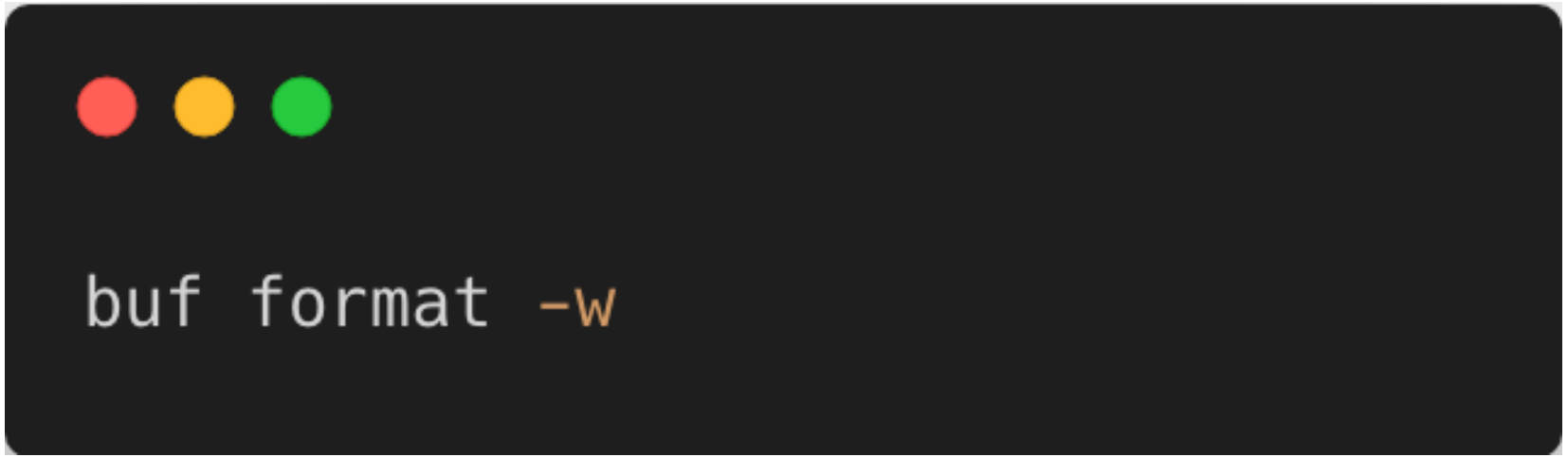
01

Оптимизация генерации кода

02

Форматирование и линтинг protobuf схем

Форматирование protobuf -хем



Форматирование protobuf-схем

Автоматическое форматирование в VSCode

- Установите форматирование Buf
- Выберете Buf в качестве форматера по умолчанию:

```
⌘ + Shift + P -> Fromat Document -> Configure Default Formatter
```

- Добавьте следующие настройки в settings.json файл для форматирования protobuf-файлов при сохранении:

```
"[proto]": {
  "editor.formatOnSave": true
}
```

Линтер protobuf-схем

Линтер protobuf-схем

Настраиваем правила линтера

- Linting Overview
- Rules and Categories
- В buf.yaml указываем необходимые нам настройки:

```
lint:
  use:
    - DEFAULT
    - COMMENTS
```

- В случае вендоринга сторонних protobuf файлов вручную, стоит указать их в разделе ignore (не всегда работает)

Какие проблемы решает Buf?

01

Оптимизация генерации кода

02

Форматирование и линтинг protobuf схем

03

Проверка совместимости protobuf схем

Проверка ломающих изменений protobuf

Проверка ломающих изменений protobuf

```
proto/api/example/v1/messages.proto:26:3:Previously present field "1" with
name "title" on message "Note" was deleted.
proto/api/example/v1/messages.proto:26:3:Previously present field "1" with
name "title" on message "Note" was deleted without reserving the name
"title".
proto/api/example/v1/messages.proto:26:3:Previously present field "1" with
name "title" on message "Note" was deleted without reserving the number "1".
```

Какие проблемы решает Buf?

01

Оптимизация генерации кода

02

Форматирование и линтинг protobuf схем

03

Проверка совместимости protobuf схем

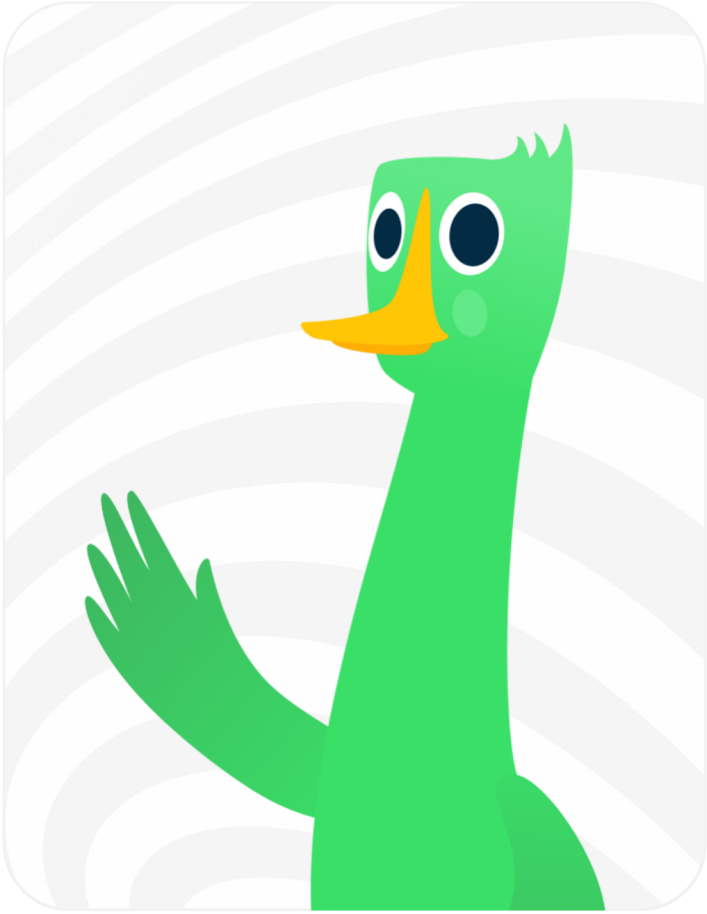
04

Централизованное управление схемами

Достаточно ли gRPC в качестве базового фреймворка для микросервисов?

А что еще нужно для создания микросервисов на Go?

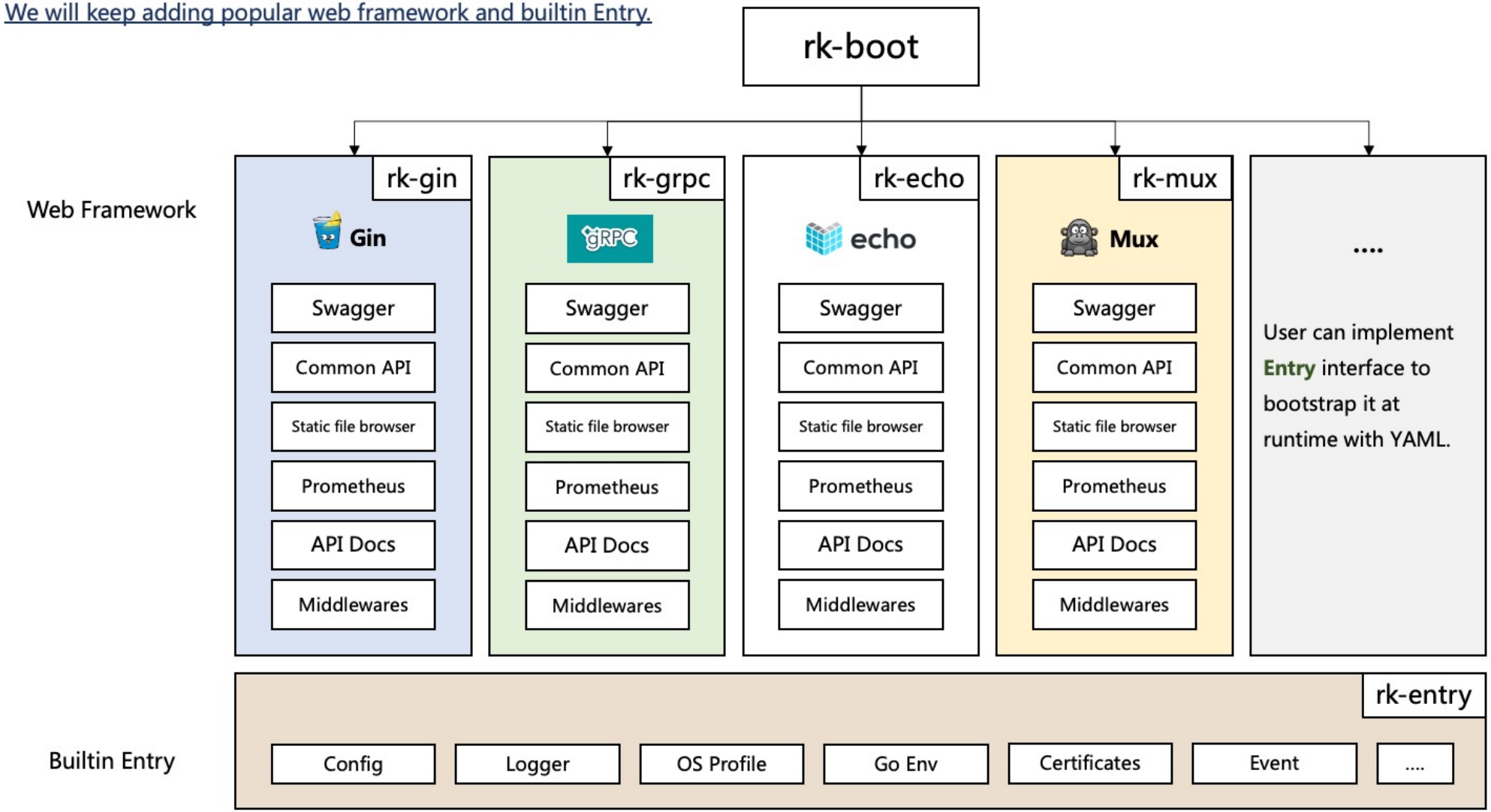
- Управление конфигурациями приложения
- Observability
- Безопасность
- Стандартизация, унификация инициализации сервиса



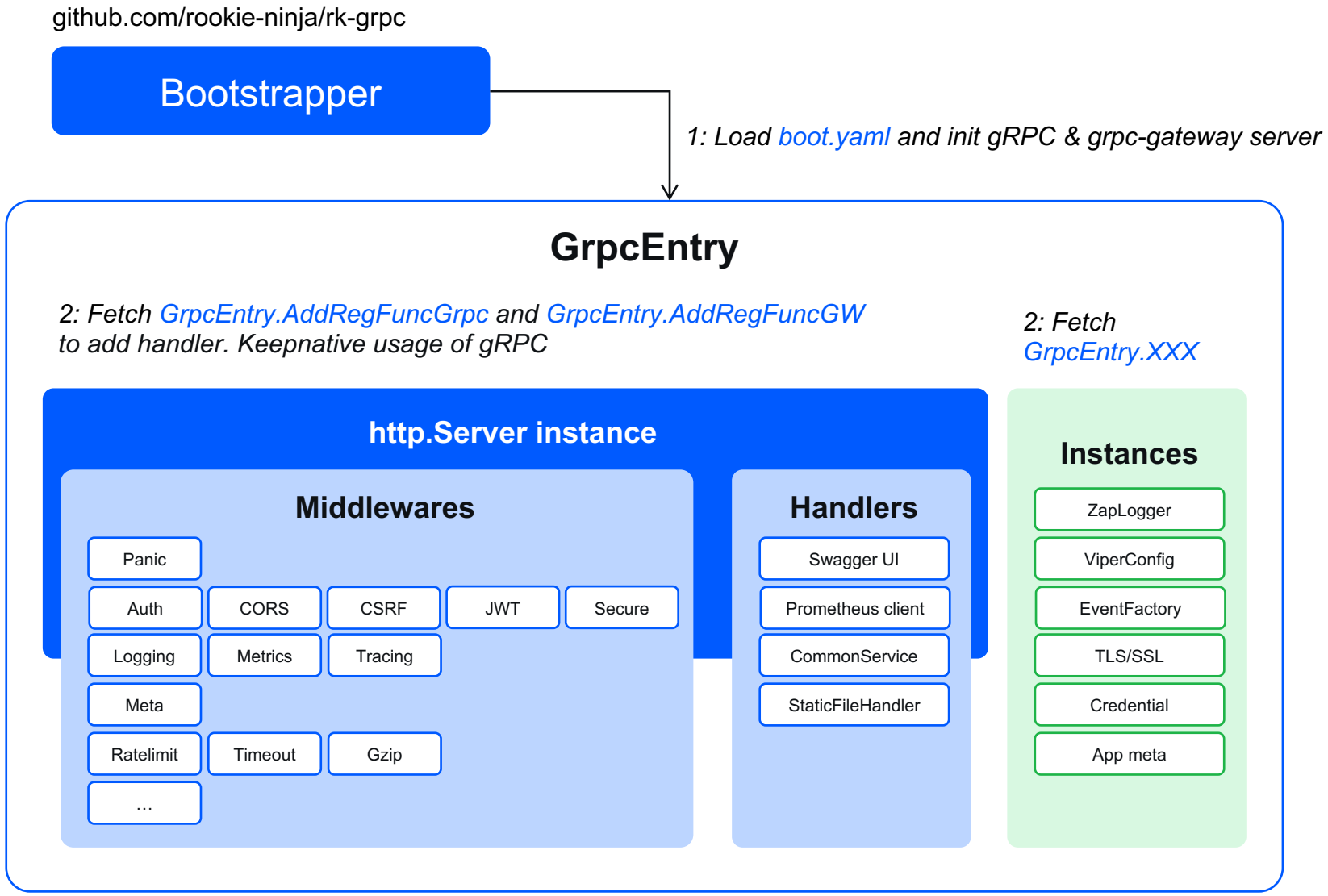
04 rk-boot

Golang

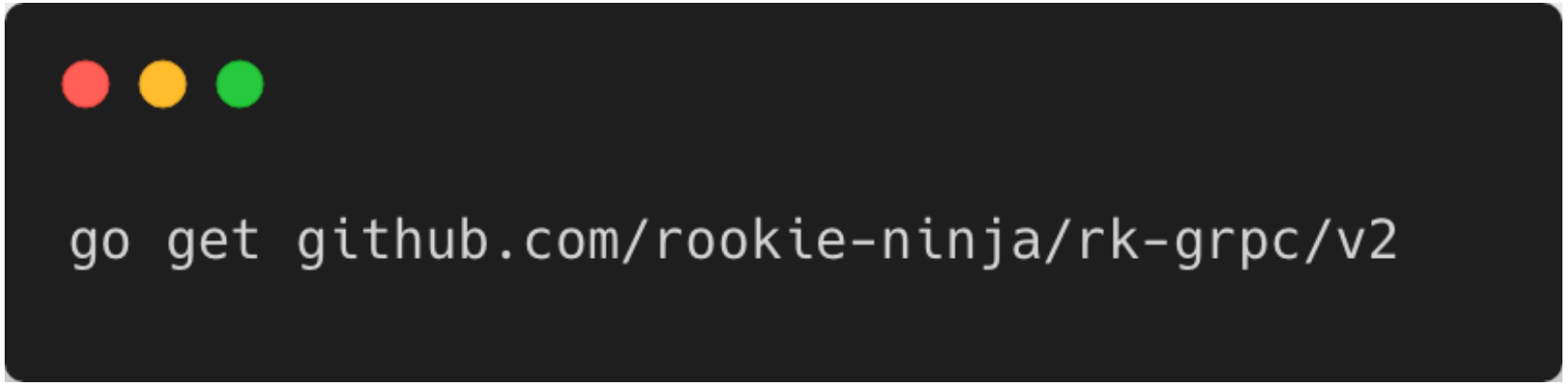
We will keep adding popular web framework and builtin Entry.



rk-grpc


github.com/rookie-ninja/rk-grpc
Bootstrapper
1: Load boot.yaml and init gRPC & grpc-gateway server
GrpcEntry
2: Fetch GrpcEntry.AddRegFuncGrpc and GrpcEntry.AddRegFuncGW to add handler. Keepnative usage of gRPC
2: Fetch GrpcEntry.XXX
http.Server instance
Middlewares
Panic
Auth
CORS
CSRF
JWT
Secure
Logging
Metrics
Tracing
Meta
Ratelimit
Timeout
Gzip
...
Handlers
Swagger UI
Prometheus client
CommonService
StaticFileHandler
Instances
ZapLogger
ViperConfig
EventFactory
TLS/SSL
Credential
App meta

rk-grpc



boot.yaml

```yaml
grpc:
  - name: example # Название нашего entry
    description: "example server"
    enabled: true # Можем отключить entry при необходимости
    port: 82 # Порт на котором будем принимать входящие gRPC запросы
    enableReflection: true # Включить gRPC reflection (в основном нужно для grpcurl, grpcli, postman)
    gwPort: 80 # Порт на котором будем принимать входящие HTTP запросы в gRPC-Gateway
    gwOption: # Настройки опций gRPC-Gateway
      marshal: # https://pkg.go.dev/google.golang.org/protobuf/encoding/protojson#MarshalOptions
        multiline: false
        emitUnpopulated: true
        indent: ""
        allowPartial: false
        useProtoNames: false
        useEnumNumbers: false
      unmarshal: # https://pkg.go.dev/google.golang.org/protobuf/encoding/protojson#UnmarshalOptions
        allowPartial: false
        discardUnknown: true
```

config.go

```go
package config

import (
    _ "embed"
)

//go:embed boot.yaml
var Boot []byte
```

main.go

```go
func main() {
    // Инициализация нашего RPC обработчика
    srv, err := examplev1.NewExampleServiceServerImplementation()
    if err != nil {...}

    // Загрузжаем entries из конфигурации (boot.yaml).
    boot := rkboot.NewBoot(
        rkboot.WithBootConfigRaw(config.Boot),
    )

    // Получение GrpcEntry
    grpcEntry := rkgrpc.GetGrpcEntry("example") // название entry
    // Регистрация gRPC сервера
    grpcEntry.AddRegFuncGrpc(func(server *grpc.Server) { pb.RegisterExampleServiceServer(server, srv) })
    // Регистрация gRPC-Gateway proxy
    grpcEntry.AddRegFuncGw(pb.RegisterExampleServiceHandlerFromEndpoint)

    // Bootstrap entry
    boot.Bootstrap(ctx)

    // Ждем сигнала выключения
    boot.WaitForShutdownSig(ctx)
}
```

main.go

```go
func main() {
    // Инициализация нашего RPC обработчика
    srv, err := examplev1.NewExampleServiceServerImplementation()
    if err != nil {...}

    // Загрузжаем entries из конфигурации (boot.yaml).
    boot := rkboot.NewBoot(
        rkboot.WithBootConfigRaw(config.Boot),
    )

    // Получение GrpcEntry
    grpcEntry := rkgrpc.GetGrpcEntry("example") // название entry
    // Регистрация gRPC сервера
    grpcEntry.AddRegFuncGrpc(func(server *grpc.Server) { pb.RegisterExampleServiceServer(server, srv) })
    // Регистрация gRPC-Gateway proxy
    grpcEntry.AddRegFuncGw(pb.RegisterExampleServiceHandlerFromEndpoint)

    // Bootstrap entry
    boot.Bootstrap(ctx)

    // Ждем сигнала выключения
    boot.WaitForShutdownSig(ctx)
}
```

main.go

```go
func main() {
    // Инициализация нашего RPC обработчика
    srv, err := examplev1.NewExampleServiceServerImplementation()
    if err != nil {...}

    // Загрузжаем entries из конфигурации (boot.yaml).
    boot := rkboot.NewBoot(
        rkboot.WithBootConfigRaw(config.Boot),
    )

    // Получение GrpcEntry
    grpcEntry := rkgrpc.GetGrpcEntry("example") // название entry
    // Регистрация gRPC сервера
    grpcEntry.AddRegFuncGrpc(func(server *grpc.Server) { pb.RegisterExampleServiceServer(server, srv) })
    // Регистрация gRPC-Gateway proxy
    grpcEntry.AddRegFuncGw(pb.RegisterExampleServiceHandlerFromEndpoint)

    // Bootstrap entry
    boot.Bootstrap(ctx)

    // Ждем сигнала выключения
    boot.WaitForShutdownSig(ctx)
}
```

Run

```
2024-09-15T15:38:17.026+0300    INFO    boot/grpc_entry.go:1060 Bootstrap grpcEntry     {"eventId":
"f435d5da-ea33-4d95-8e11-e247c3ae2dce", "entryName": "example", "entryType": "gRPCEntry"}
2024-09-15T15:38:17.026+0300    INFO    boot/grpc_entry.go:761  gRPC_port:82
2024-09-15T15:38:17.026+0300    INFO    boot/grpc_entry.go:762  gateway_port:80
```

04

rk-boot

Конфиг приложения

Config

```yaml
config: # https://github.com/spf13/viper
  - name: config
    description: "Description of entry"
    domain: "*"
    # path: "config/config.yaml"
    envPrefix: ""
    content:
      key: value
      key1: value1
      key2: value2
```

Config

```
// Конфиг приложения
cfg := rkentry.GlobalAppCtx.GetConfigEntry("config")

logger.Info("read config",
    zap.String("key", cfg.GetString("key")),
    zap.String("key1", cfg.GetString("key1")),
    zap.String("key2", cfg.GetString("key2")),
)
```

04

rk-boot

Документация API

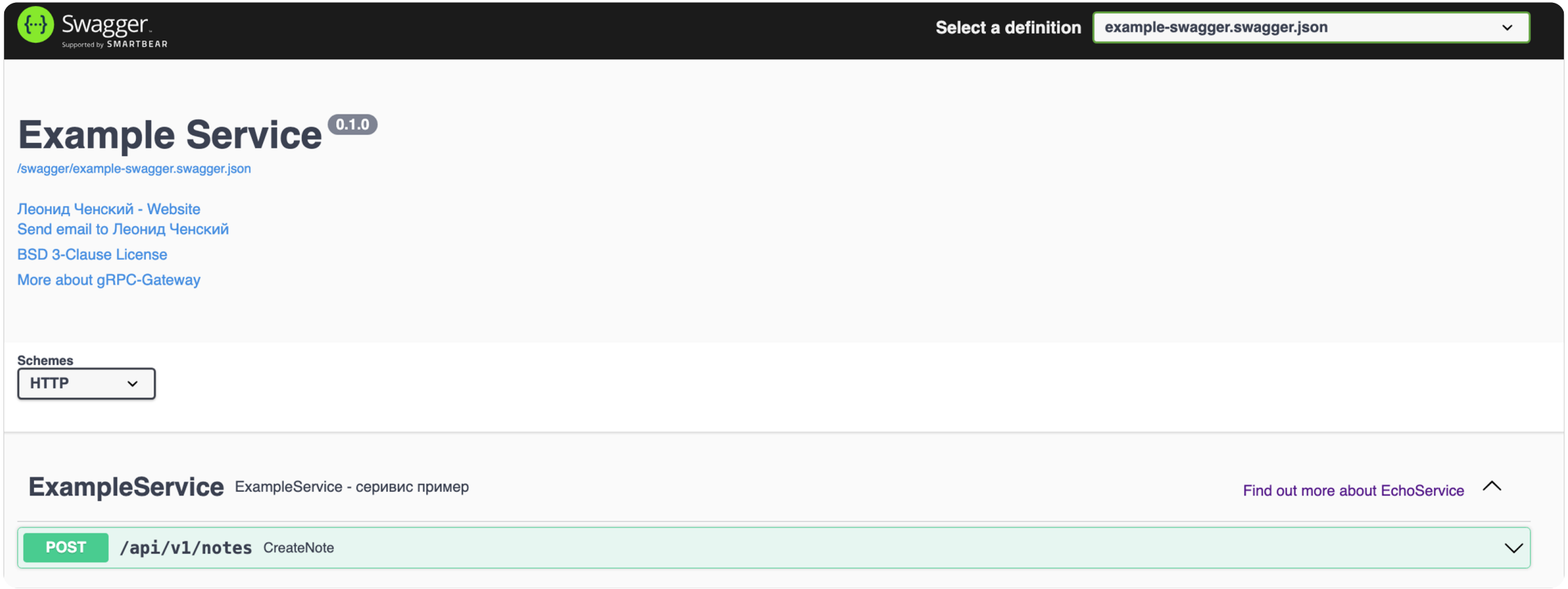
API документация

```yaml
commonService: # Swagger UI клиент для RK сервиса
  enabled: true
sw: # Swagger UI клиент: https://github.com/swagger-api/swagger-ui
  enabled: true
  path: "swagger"
  jsonPaths:
    - swagger
  headers: []
docs: # Встроенный экземпляр RapiDoc https://github.com/rapi-doc/RapiDoc, который можно использовать вместо Swagger
  enabled: true
  path: "docs"
  specPath: "swagger"
  headers: []
  style:
    theme: "light"
  debug: true
```

Run

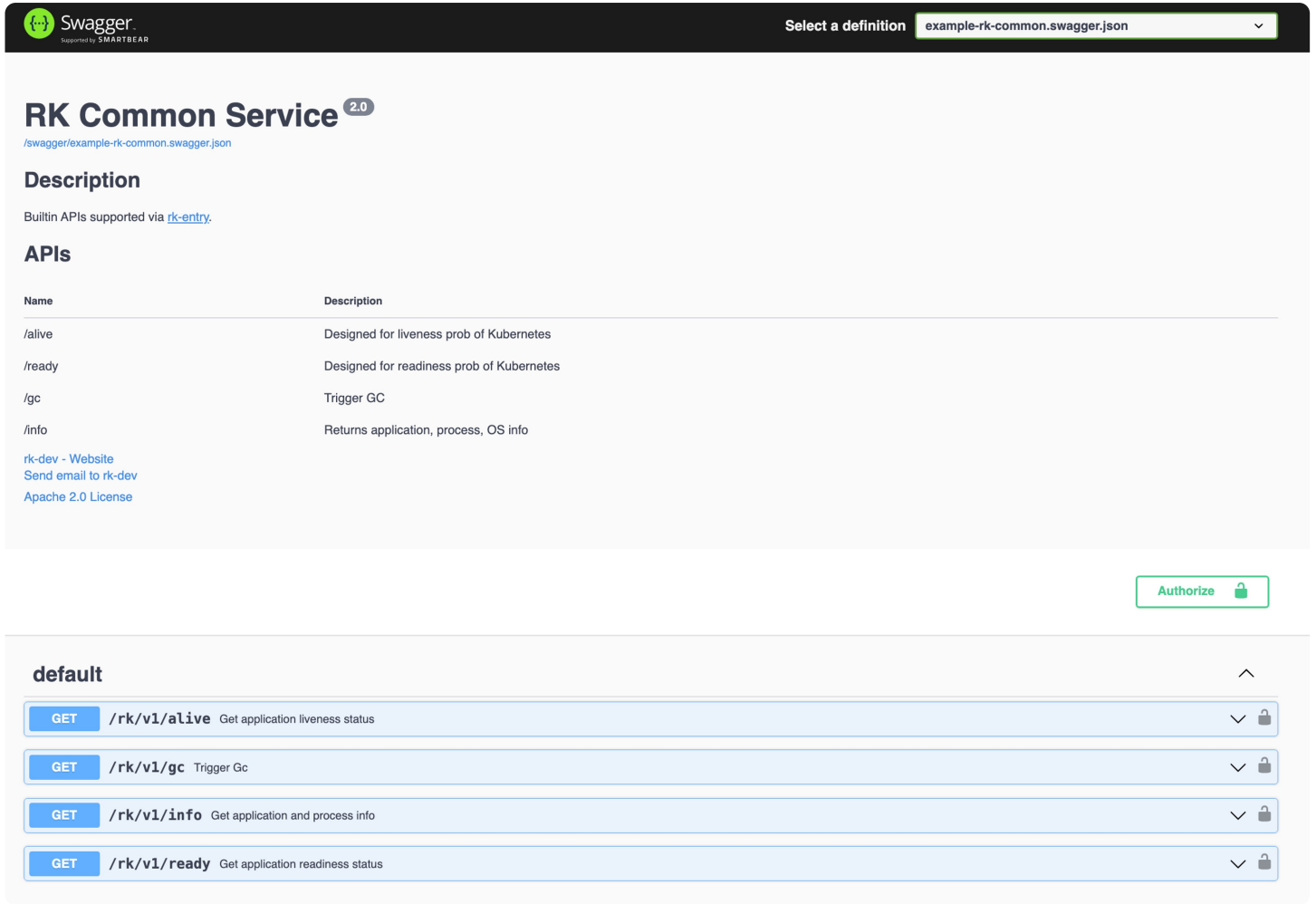
```
2024-09-15T16:08:42.262+0300    INFO    boot/grpc_entry.go:1060 Bootstrap grpcEntry     {"eventId":
"a636bbf6-afd0-4b06-80e8-0f0f87951e44", "entryName": "example", "entryType": "gRPCEntry"}
2024-09-15T16:08:42.262+0300    INFO    boot/grpc_entry.go:761  gRPC_port:82
2024-09-15T16:08:42.262+0300    INFO    boot/grpc_entry.go:762  gateway_port:80
2024-09-15T16:08:42.262+0300    INFO    boot/grpc_entry.go:765  SwaggerEntry: http://localhost:80/swagger/
2024-09-15T16:08:42.262+0300    INFO    boot/grpc_entry.go:768  DocsEntry: http://localhost:80/docs/
2024-09-15T16:08:42.262+0300    INFO    boot/grpc_entry.go:783  CommonSreviceEntry:
http://localhost:80/rk/v1/ready, http://localhost:80/rk/v1/alive, http://localhost:80/rk/v1/info
```

Swagger UI



http://localhost:8080/swagger/

Common API



http://localhost:8080/swagger/

04

rk-boot

Observability

Метрики

boot.yaml

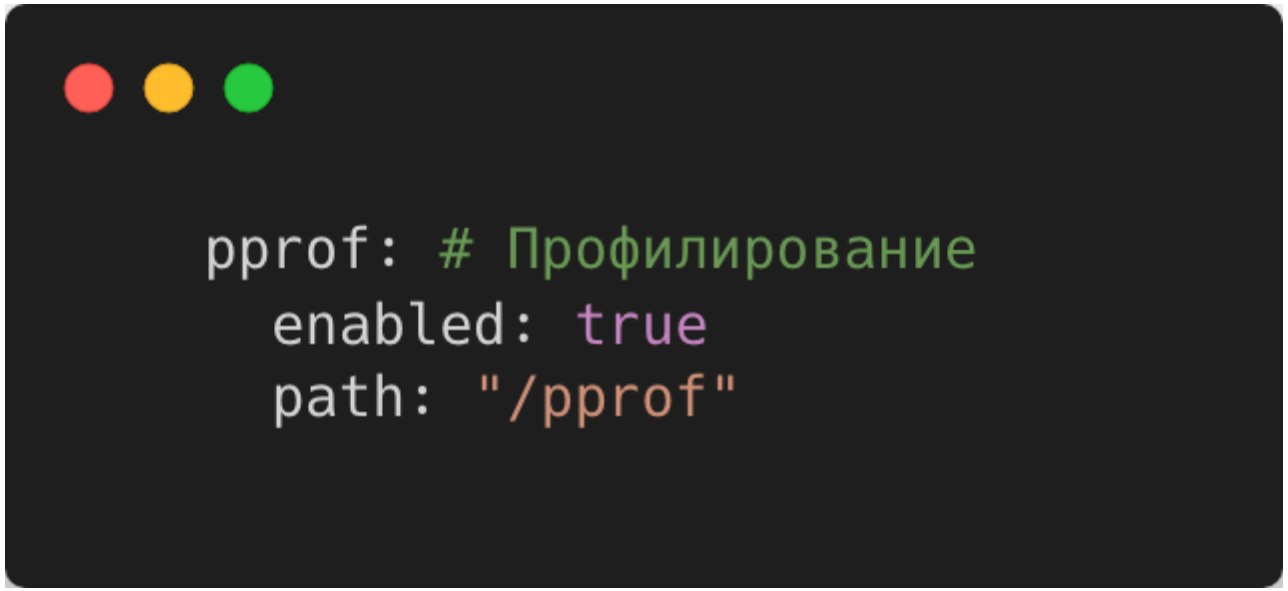
```yaml
prom: # prometheus client /metrics
  enabled: true
middleware:
  prom: # метрики запросов
    enabled: true
```

Метрики

```
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 2.2834e-05
go_gc_duration_seconds{quantile="0.25"} 4.1084e-05
go_gc_duration_seconds{quantile="0.5"} 9.1709e-05
go_gc_duration_seconds{quantile="0.75"} 0.000255
go_gc_duration_seconds{quantile="1"} 0.000335459
go_gc_duration_seconds_sum 0.000746086
go_gc_duration_seconds_count 5
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 23
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.22.5"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 7.501696e+06
# HELP go_memstats_alloc_bytes_total Total number of bytes allocated, even if freed.
# TYPE go_memstats_alloc_bytes_total counter
go_memstats_alloc_bytes_total 1.4918768e+07
```

http://localhost/metrics

Профилирование



```yaml
pprof: # Профилирование
  enabled: true
  path: "/pprof"
```

Профилирование

/debug/pprof/

Set debug=1 as a query parameter to export in legacy text format

Types of profiles available:

Count	Profile
10	allocs
0	block
0	cmdline
14	goroutine
10	heap
0	mutex
0	profile
15	threadcreate
0	trace

full goroutine stack dump

Profile Descriptions:

- allocs: A sampling of all past memory allocations
- block: Stack traces that led to blocking on synchronization primitives
- cmdline: The command line invocation of the current program
- goroutine: Stack traces of all current goroutines. Use debug=2 as a query parameter to export in the same format as an unrecovered panic.
- heap: A sampling of memory allocations of live objects. You can specify the gc GET parameter to run GC before taking the heap sample.
- mutex: Stack traces of holders of contended mutexes
- profile: CPU profile. You can specify the duration in the seconds GET parameter. After you get the profile file, use the go tool pprof command to investigate the profile.
- threadcreate: Stack traces that led to the creation of new OS threads
- trace: A trace of execution of the current program. You can specify the duration in the seconds GET parameter. After you get the trace file, use the go tool trace command to investigate the trace.

Логирование

```
2024-09-15T16:08:42.262+0300    INFO    boot/grpc_entry.go:1060 Bootstrap grpcEntry     {"eventId":
"a636bbf6-afd0-4b06-80e8-0f0f87951e44", "entryName": "example", "entryType": "gRPCEntry"}
2024-09-15T16:08:42.262+0300    INFO    boot/grpc_entry.go:761  gRPC_port:82
2024-09-15T16:08:42.262+0300    INFO    boot/grpc_entry.go:762  gateway_port:80
2024-09-15T16:08:42.262+0300    INFO    boot/grpc_entry.go:765  SwaggerEntry: http://localhost:80/swagger/
2024-09-15T16:08:42.262+0300    INFO    boot/grpc_entry.go:768  DocsEntry: http://localhost:80/docs/
2024-09-15T16:08:42.262+0300    INFO    boot/grpc_entry.go:783  CommonSreviceEntry:
http://localhost:80/rk/v1/ready, http://localhost:80/rk/v1/alive, http://localhost:80/rk/v1/info
```

Логирование

```yaml
logger: # https://github.com/rookie-ninja/rk-logger
  - name: zap
    description: "ZAP"
    domain: "*"
    default: true
    zap:
      level: debug
      development: true
      disableCaller: false
      disableStacktrace: true
      encoding: json # console, json, flatten
      outputPaths: ["stdout"]
      errorOutputPaths: ["stderr"]
      encoderConfig:
      sampling:
      initialFields:
```

Логирование


```
{"level":"INFO","ts":"2024-09-
16T21:41:21.213+0300","caller":"boot/grpc_entry.go:1060","msg":"Bootstrap
grpcEntry","key":"value","eventId":"fa1f0ec1-5421-4315-a5a0-
a37f0fe58c70","entryName":"example","entryType":"gRPCEntry"}
{"level":"INFO","ts":"2024-09-
16T21:41:21.214+0300","caller":"boot/grpc_entry.go:761","msg":"gRPC_port:82","key":"value"}
{"level":"INFO","ts":"2024-09-
16T21:41:21.214+0300","caller":"boot/grpc_entry.go:762","msg":"gateway_port:80","key":"value"}
{"level":"INFO","ts":"2024-09-
16T21:41:21.214+0300","caller":"boot/grpc_entry.go:765","msg":"SwaggerEntry:
http://localhost:80/swagger/","key":"value"}
{"level":"INFO","ts":"2024-09-
16T21:41:21.214+0300","caller":"boot/grpc_entry.go:768","msg":"DocsEntry:
http://localhost:80/docs/","key":"value"}
{"level":"INFO","ts":"2024-09-
16T21:41:21.214+0300","caller":"boot/grpc_entry.go:771","msg":"PromEntry:
http://localhost:80/metrics","key":"value"}
{"level":"INFO","ts":"2024-09-
16T21:41:21.214+0300","caller":"boot/grpc_entry.go:783","msg":"CommonSreviceEntry:
http://localhost:80/rk/v1/ready, http://localhost:80/rk/v1/alive,
http://localhost:80/rk/v1/info","key":"value"}
{"level":"INFO","ts":"2024-09-
16T21:41:21.214+0300","caller":"boot/grpc_entry.go:786","msg":"PProfEntry:
http://localhost:80/pprof/","key":"value"b
```

Логирование запросов

Настройка middleware-логирования

```yaml
middleware:
  logging: # логирование https://github.com/rookie-ninja/rk-query
    enabled: true
    ignore: [""]
    loggerEncoding: "json" # console, json, flatten
    loggerOutputPaths: ["stdout"]
    eventEncoding: "json" # console, json, flatten
    eventOutputPaths: ["stdout"]
```

Логирование запросов


```
{"endTime": "2024-09-16T21:35:34.779+0300", "startTime": "2024-09-
16T21:35:34.764+0300", "elapsedNano": 15128750, "timezone": "MSK", "ids":
{"eventId":"3692a4d7-00d0-4c9d-bab3-
ac800c514dd1","traceId":"1e778e0712c57f25878d534fbbade1c4"}, "app":
{"appName":"rk","appVersion":"local","entryName":"example","entryType":"gRPCEntry"},
"env": {"arch":"arm64","domain":"*","hostname":"MacBook-Pro-
3.local","localIP":"192.168.105.1","os":"darwin"}, "payloads":
{"apiMethod":"","apiPath":"/api.example.v1.ExampleService/CreateNote","apiProtocol":"",
"apiQuery":"","grpcMethod":"CreateNote","grpcService":"api.example.v1.ExampleService","
grpcType":"UnaryServer","gwMethod":"","gwPath":"","gwScheme":"","gwUserAgent":"","userA
gent":""}, "error": {}, "counters": {}, "pairs": {}, "timing": {}, "remoteAddr":
"127.0.0.1:63656", "operation": "/api.example.v1.ExampleService/CreateNote",
"eventStatus": "Ended", "resCode": "OK"}
```

Трейсинг

```yaml
middleware:
  trace: # Трейсинг
    enabled: true
    ignore: [""]
    exporter:
      file:
        enabled: true
        outputPath: "logs/trace.log"
      jaeger:
        agent:
          enabled: false
          host: "localhost"
          port: 6831
        collector:
          enabled: false
          endpoint: "http://localhost:14268/api/traces"
          username: ""
          password: ""
```

Трейсинг

```yaml
middleware:
  trace: # Трейсинг
    enabled: true
    ignore: [""]
    exporter:
      file:
        enabled: true
        outputPath: "logs/trace.log"
      jaeger:
        agent:
          enabled: false
          host: "localhost"
          port: 6831
        collector:
          enabled: false
          endpoint: "http://localhost:14268/api/traces"
          username: ""
          password: ""
```

Трейсинг

```
{
  "Name": "/api.example.v1.ExampleService/CreateNote",
  "SpanContext": {
    "TraceID": "2cf323594e03e56d6db6a0dac5d42069",
    "SpanID": "305e2faff5b189e0",
    "TraceFlags": "01",
    "TraceState": "",
    "Remote": false
  },
  "Parent": {},
  "SpanKind": 2,
  "StartTime": "2024-09-17T21:42:28.106028+03:00",
  "EndTime": "2024-09-17T21:42:28.122781125+03:00",
  ...
}
```

04

rk-boot

Прочее

Timeout

```yaml
middleware:
  timeout:
    enabled: true
    ignore: [""] # black list
    timeoutMs: 5000
    paths:
      - path: "/api.example.v1.ExampleService/CreateNote"
        timeoutMs: 50
```

Timeout

Code	Details
499	Error: status code 499 **Response body**

```
{
  "code": 1,
  "message": "Request timed out!",
  "details": [
    {
      "@type": "type.googleapis.com/rk.api.v1.ErrorDetail",
      "code": 1,
      "status": "Canceled",
      "message": "Request timed out!"
    }
  ]
}
```

Ratelimit

```yaml
middleware:
  rateLimit:
    enabled: true
    ignore: [""] # black list
    algorithm: "leakyBucket"
    reqPerSec: 1000
    paths:
      - path: "/api.example.v1.ExampleService/CreateNote"
        reqPerSec: 1
```

Прочее

Middleware	Описание
Panic	Recover from panic for RPC requests and log it.
Auth	Support [Basic Auth] and [API Key] authorization types.
CORS	Server side CORS validation.
JWT	Server side JWT validation.
Secure	Server side secure validation.
CSRF	Server side CSRF validation.

Итоги

- gRPC + rk-boot закрывают большинство потребностей на старте в микросервисах
- rk-boot и gRPC расширяемые
- Делать стандартные миркосервисы на Go – легко

Проект с исходным кодом

https://github.com/moguchev/gofunc_autumn_2024

OZON tech

Спасибо за внимание!

Леонид Ченский
Руководитель команды DBA Scylla
lmoguchev@ozon.ru