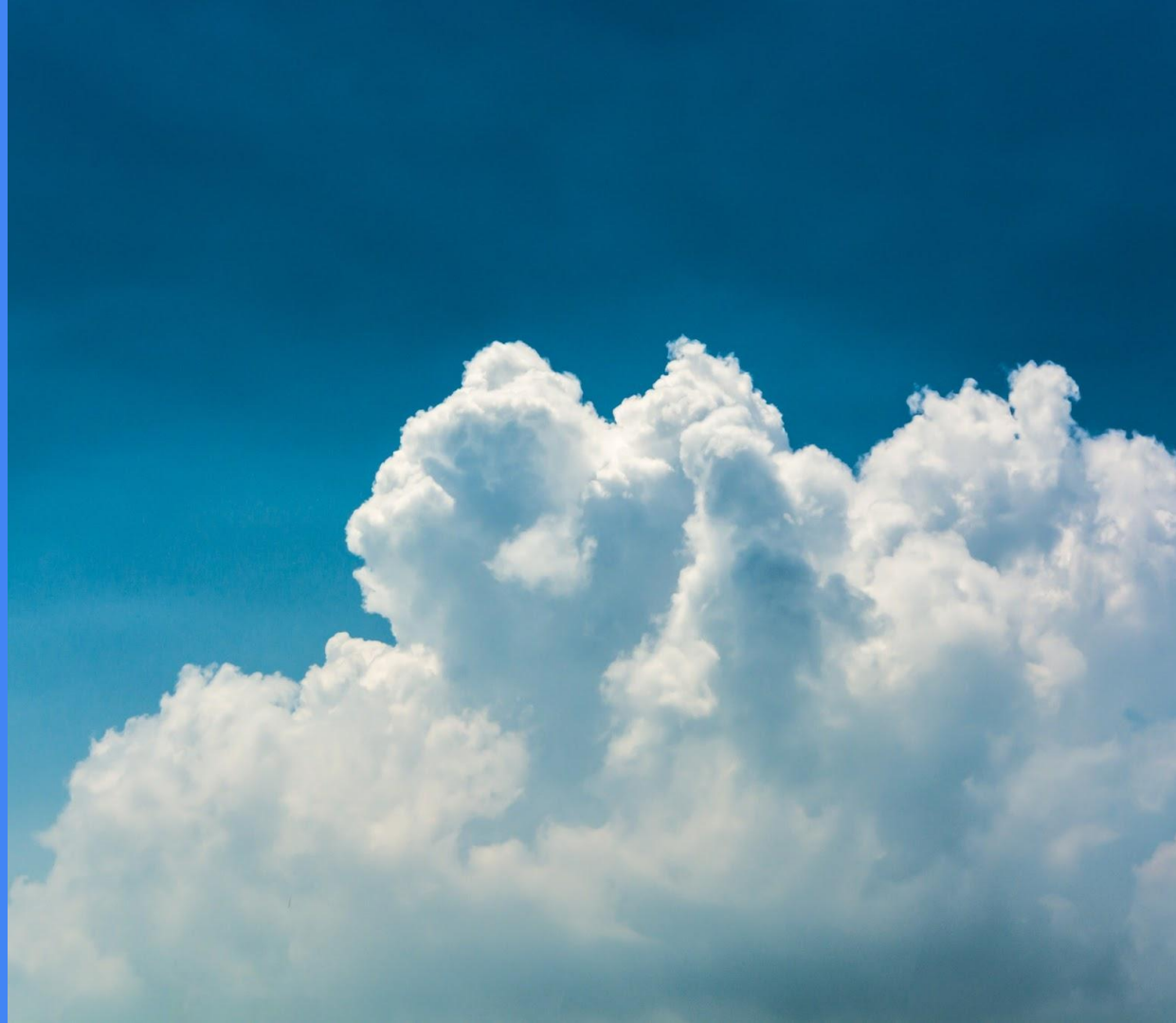
Networked APIs

Gregory Hill

Foreword

Intro to gRPC: A Modern Toolkit for Microservice Communication

Type-Safety



History[1]

1. Simple Object Access Protocol (SOAP) - XML WSDL
2. Asynchronous JavaScript (AJAX -> JSON)
3. Representational State Transfer (REST)
4. GraphQL
5. gRPC

[1] https://dev.to/mikeralphson/a-brief-history-of-web-apis-47k4

Problem

Documentation &
Machine Readability

Schema Driven Development

Advantages

1. Single Source of Truth
2. Maintainable
3. Versionable
4. Developer Friendly
5. Language-Agnostic

Approaches

OpenAPI

- Specification - JSON or YAML
  - Endpoints
  - Operations
  - Inputs
  - Authentication

Swagger

- UI / Editor
- Code Generation

Protocol Buffers

- Interface Definition Language (IDL)
- Serialization Library

gRPC

- RPC Framework - Service
- Unary, Server / Client Streaming & Bidirectional

OpenAPI / REST

- HTTP/1.1
- Interoperable
- Architectural

Comments:
// swagger:meta
// swagger:route GET /users

$ go get -u github.com/go-swagger/go-swagger/cmd/swagger

$ swagger generate client -f ./swagger.json

```
package main

import (
    "context"
    "fmt"
    "log"

    "github.com/package/api-client/client"
    "github.com/package/api-client/client/users"
    httptransport "github.com/go-openapi/runtime/client"
    "github.com/go-openapi/strfmt"
)


func main() {
    transport := httptransport.New("localhost:3000", "", nil)
    api := client.New(transport, strfmt.Default)

    headerToken := httptransport.BearerToken(token.AccessToken)
    params := users.NewGetUsersProfileParams()

    resp, _ := api.Users.GetUsersProfile(params, headerToken)
    fmt.Printf("%v\n", resp.Payload)
}
```

gRPC / Protobuf

- HTTP/2
- Speed / Performance
- Pluggable

Microservices made easy...

$ protoc storage.proto --gogo_out=plugins=grpc:${GOPATH}/src

```
syntax = "proto3";

package storage;

option go_package = "github.com/user/package" ;

message Item {
   // The data that we stored earlier
   bytes Content = 1;
   // The name of the user that uploaded the data
   string Author = 2;
}

service Storage {
   // Returns the number of items stored
   rpc Entries returns (int32);

   // Get the stored item
   rpc Get (string) returns (Item);
}
```

```
...

listener, err := net.Listen(netProtocol, localAddress)
if err != nil {
    return fmt.Errorf("failed to create listener:  %v", err)
}
grpcServer := grpc.NewServer()
RegisterStorageServer(grpcServer, &grpcService)
grpcServer.Serve(listener)
```

//go:generate

Thanks for listening!

Fin.