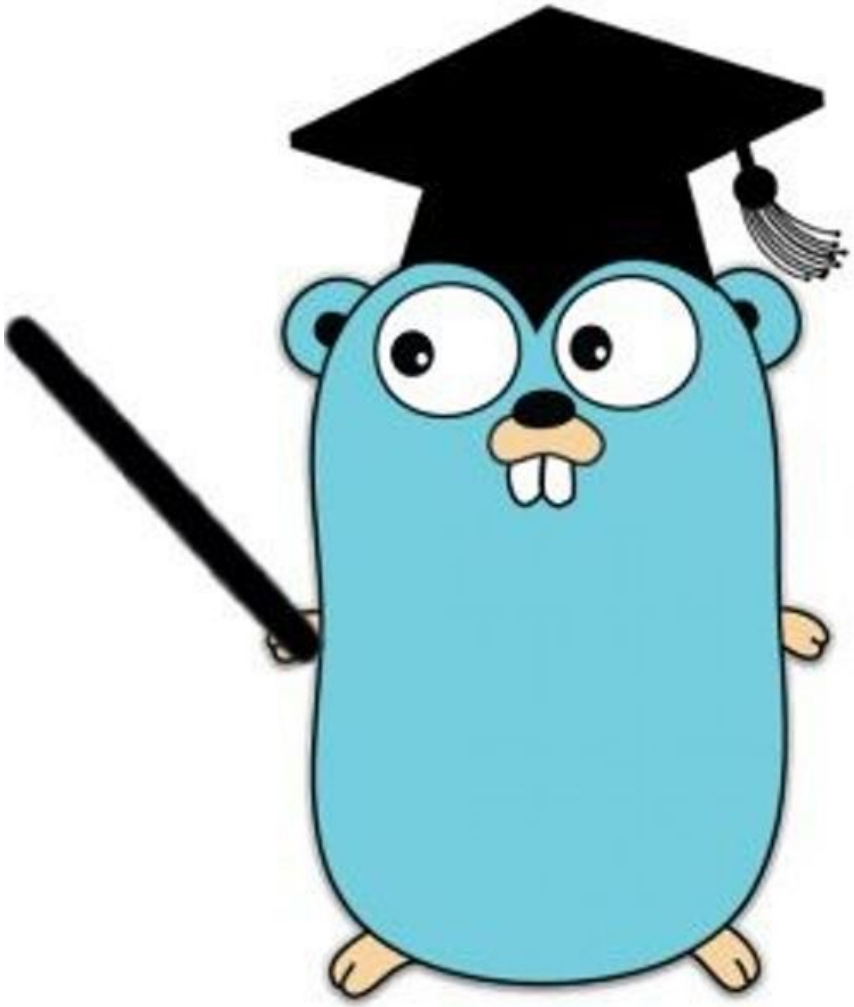
# gRPC microservices are the future ?

## Build scalable gRPC services with go deployed to GKE

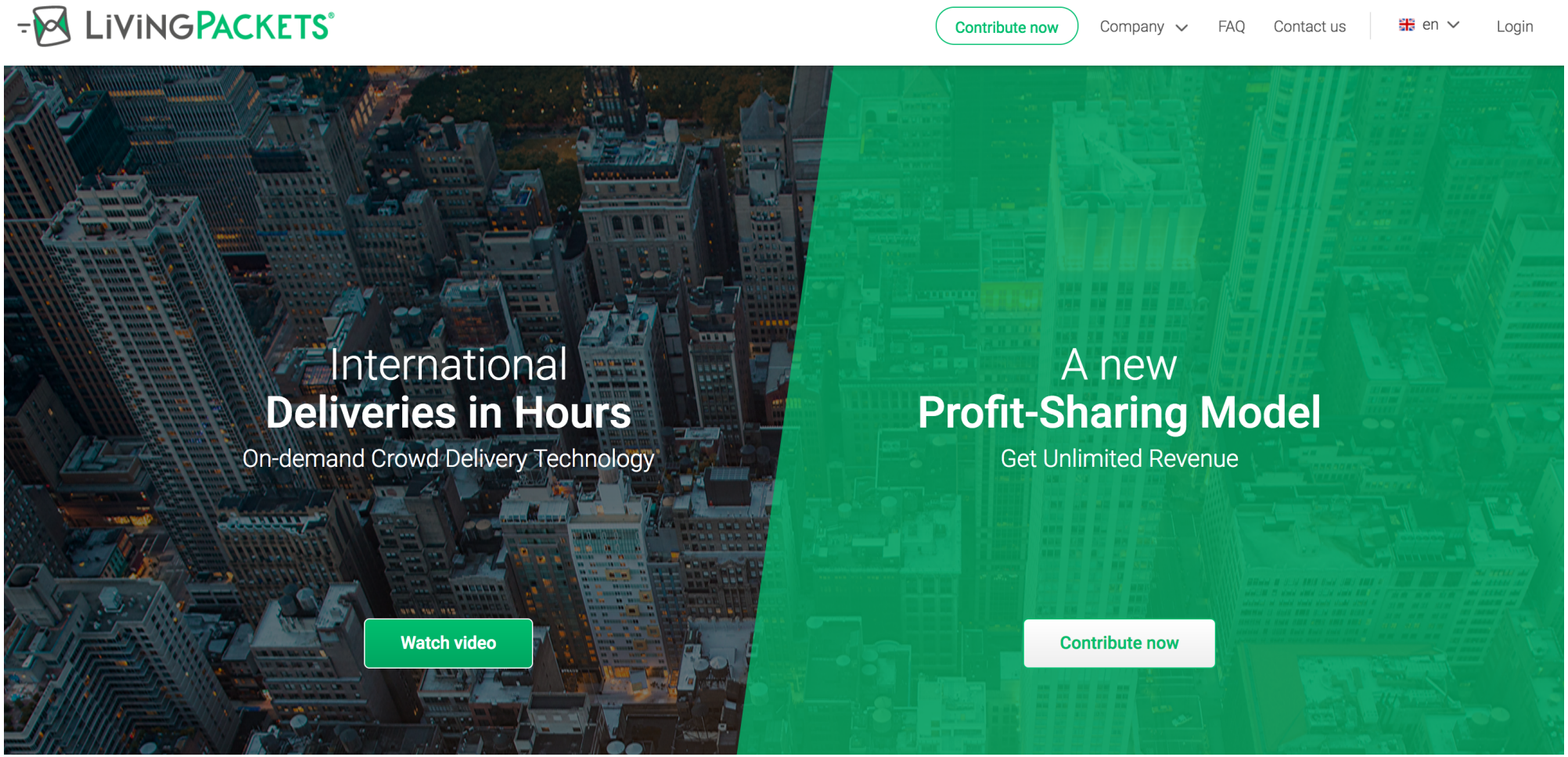
Golang Nantes Meetup
21 September 2017

Cyrille Hemidy
LivingPackets (http://LivingPackets.com)

# [1] Hello Gophers



# [2] @work



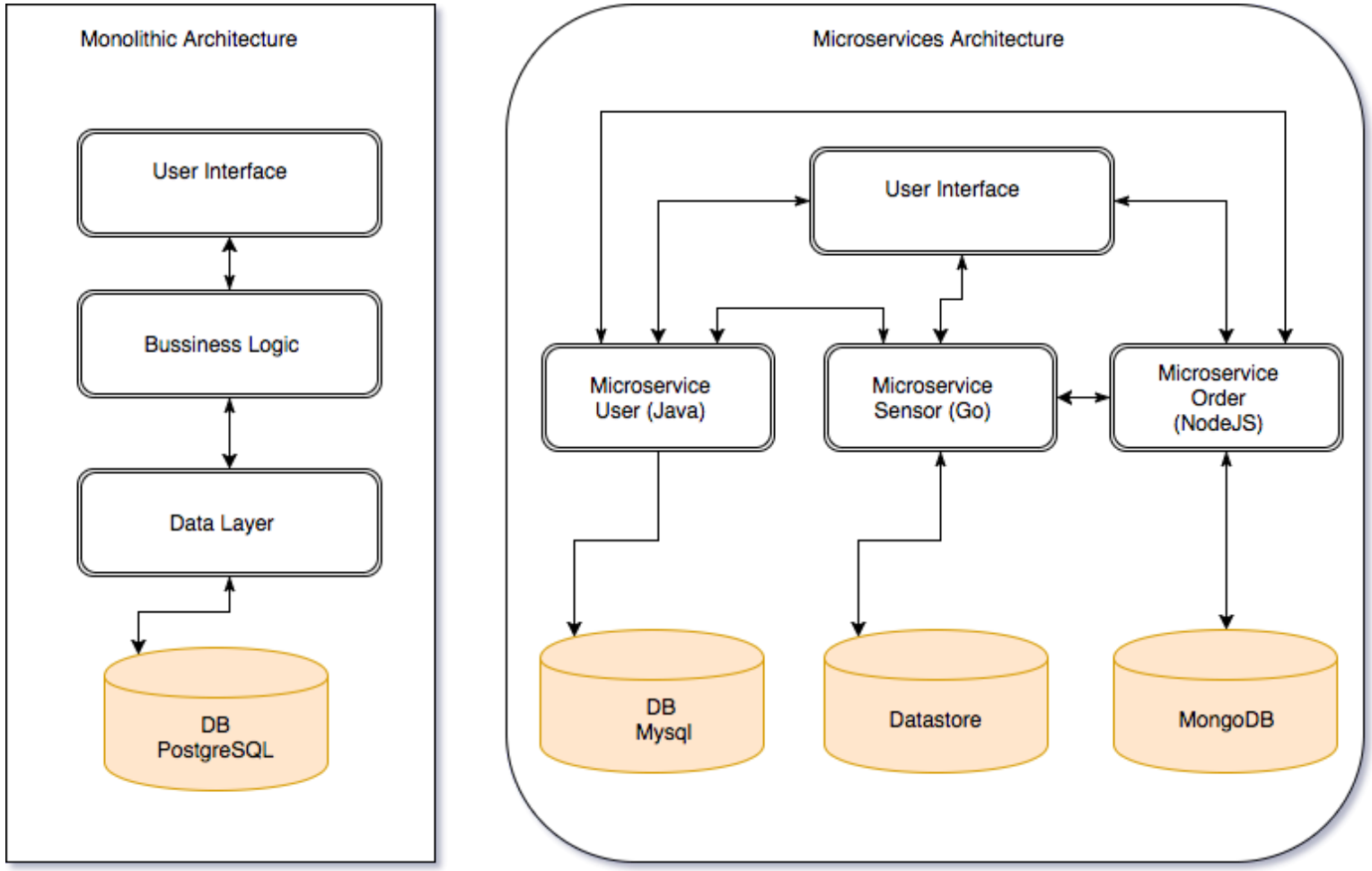

- See LivingPackets.com (https://www.livingpackets.com/en)

# [3] Agenda

- What are microservices, gRPC, Protocol Buffers ?
- A little bit of history
- REST vs gRPC
- Develop gRPC microservices with Go
- Call gRPC microservices with (IOS, Android, Angular)
- Manage local microservices
- Deploy microservices on GKE
- Manage API with Cloud Endpoints
- Build Docker images
- Conclusion

# [4] Let's dive into gRPC microservices ecosystem

# [5] What are microservices ?




- See Microservices are the Future (And Always Will Be) - Josh Holtzman, PayPal

  (https://www.youtube.com/watch?v=lRDgOpsV2zA)

# [6] Monolithic VS microservices ?

Monolith Architecture VS Microservices Architecture

| | Monolith Architecture | Microservices Architecture |
|---|---|---|
| **Architecture** | A single application whith spaghetti code and strong dependencies between services. | A lot of small Services that work independently and communicate with each other (REST, RPC, ...). |
| **Scalability** | Horizontal scale behind a load balancer. | Each service scales independently. Each service scales on demand. |
| **Agility** | Changes to the system is very difficult, need testing and validation of the entire application | Each service can be changed independently. Easy to add new services. |
| **Development** | Application based on a single programming language | Each services can be developed in a different programming language |
| **Maintenance** | Large code bases | Small code bases |

- See Netflix Open Source Software Center (http://netflix.github.io/)
- See Hystrix is a latency and fault tolerance library stop cascading failure (https://github.com/Netflix/Hystrix)
- See Ribbon is a Inter Process Communication (remote procedure calls) library (https://github.com/Netflix/ribbon)

# [7] A little bit of history

- 2011 : Protocol Buffers 2 (https://developers.google.com/protocol-buffers/) => language neutral for serializing structured data
- 2015 : Borg (https://static.googleusercontent.com/media/research.google.com/fr//pubs/archive/43438.pdf) => Large-scale cluster management => Kubernetes (https://kubernetes.io/)
- 2015 : Stubby (https://cloudplatform.googleblog.com/2016/08/gRPC-a-true-Internet-scale-RPC-framework-is-now-1-and-ready-for-production-deployments.html) => A high performance RPC framework => gRPC (https://grpc.io/)
- July 2016 : Protocol Buffers 3.0.0
- Aug 2016 : gRPC 1.0 ready for production
- Sept 2016 : Swift-protobuf (https://github.com/apple/swift-protobuf)
- Jan 2017 : Grpc Swift (https://github.com/grpc/grpc-swift)
- Apr 2017 : Google Endpoints (https://cloud.google.com/endpoints/) => Manage gRPC APIs with Cloud Endpoints
- Sept 2017 : gRPC 1.6.1
- Sept 2017 : Protocol Buffers 3.4.1

# [8] What is gRPC ?


- High performance, open-source RPC (Remote Procedure Call) framework that uses Google Protocol Buffers as the IDL (Interface Description Language)
- Actively developed and production ready
- Current Version is 1.6.1
- See gRPC.io (https://grpc.io/)

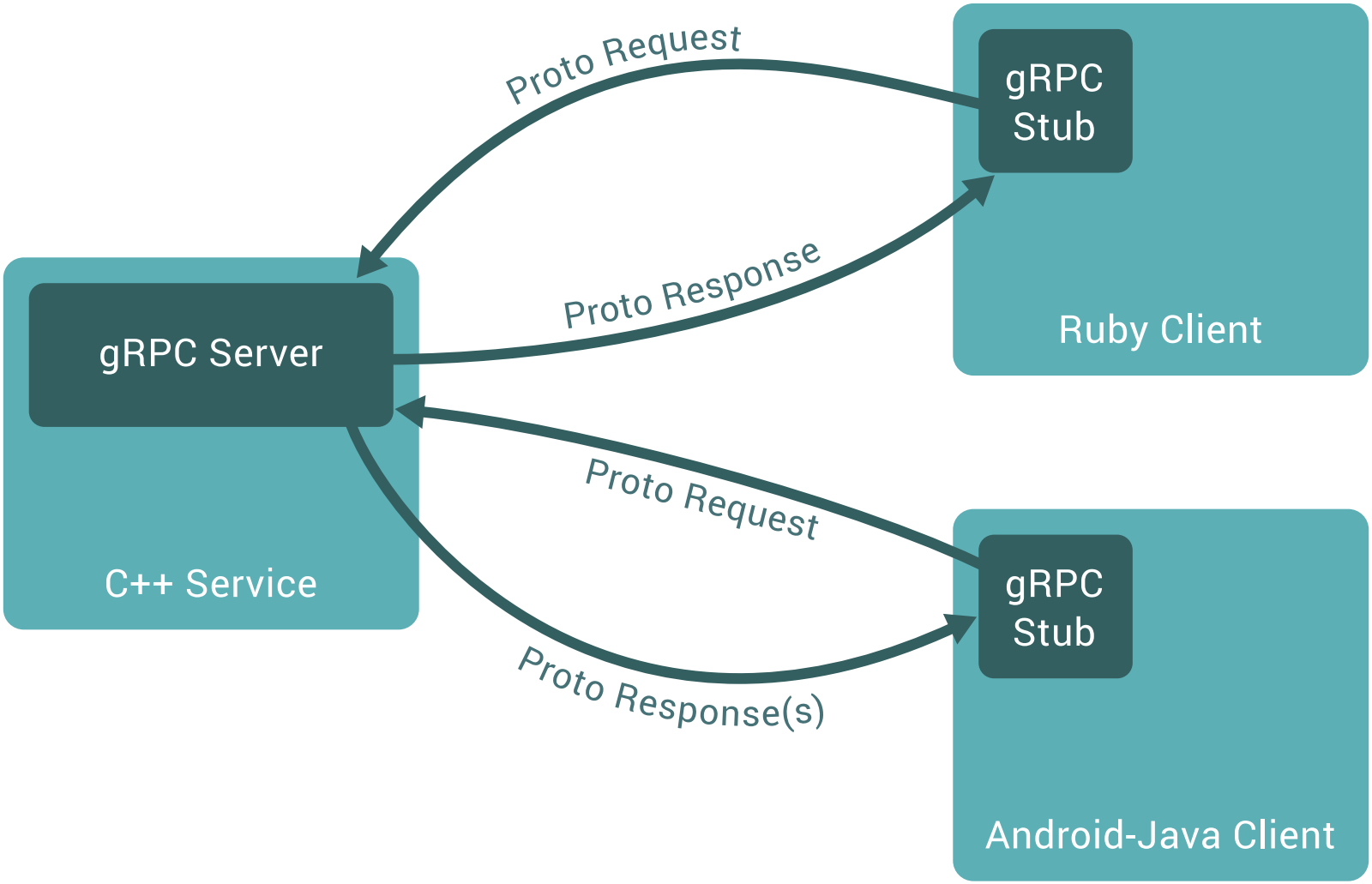
# [9] gRPC speaks a lot of languages

- C/C++
- Java
- Python
- Ruby
- Node.js
- C#
- Go
- PHP
- Objective-C
- Swift (in progress)

# [10] gRPC lets you define 4 kinds of service

- **Unary** : client sends a single request and gets a single response
- **Server streaming** : client sends a request and gets a stream of messages
- **Client streaming** : client sends a stream of messages and gets a single response
- **Bidirectional streaming** : both sides send a sequence of messages using a read-write stream. The two streams operate independently. The order of messages in each stream is preserved

# [11] gRPC Interoperability



# [12] gRPC is designed for future

- Open source
- plugins
- HTTP/2 for transport (Multiplexing, Header Compression, Binary framing)
- Design for fault tolerance (Deadlines, Cancellations)
- Very fast and efficient Protobuf serialization
- Interceptors on client and server
- Tracing
- ...
- See gRPC Community Meeting Working Doc / RoadMap (http://bit.ly/grpcmeetings)

# [13] What is Protocol Buffers ?

- Protocol buffers are Google's language-neutral, platform-neutral, flexible, efficient, extensible mechanism for serializing structured data

```
syntax = "proto3";
package superpower;

option java_package = "com.package.protobuf";
option java_multiple_files = true;
option optimize_for = LITE_RUNTIME;

service GoHeroe {
  rpc List(Filter) returns (SuperPowers) {}
  rpc Add(SuperPower) returns (SuperPowers) {}
}

message SuperPower {
  string name = 1;
  SuperPowerCategory cat = 2;
  bool coolPow = 3;
}

message SuperPowers { repeated SuperPower superPow = 1; }
message Filter { SuperPowerCategory category = 1; }
```

- See Protocol buffers: Avoid these uses (http://www.golangdevops.com/2017/08/16/why-not-to-use-protos-in-code/)

# [14] REST vs gRPC

- JSON vs PROTO

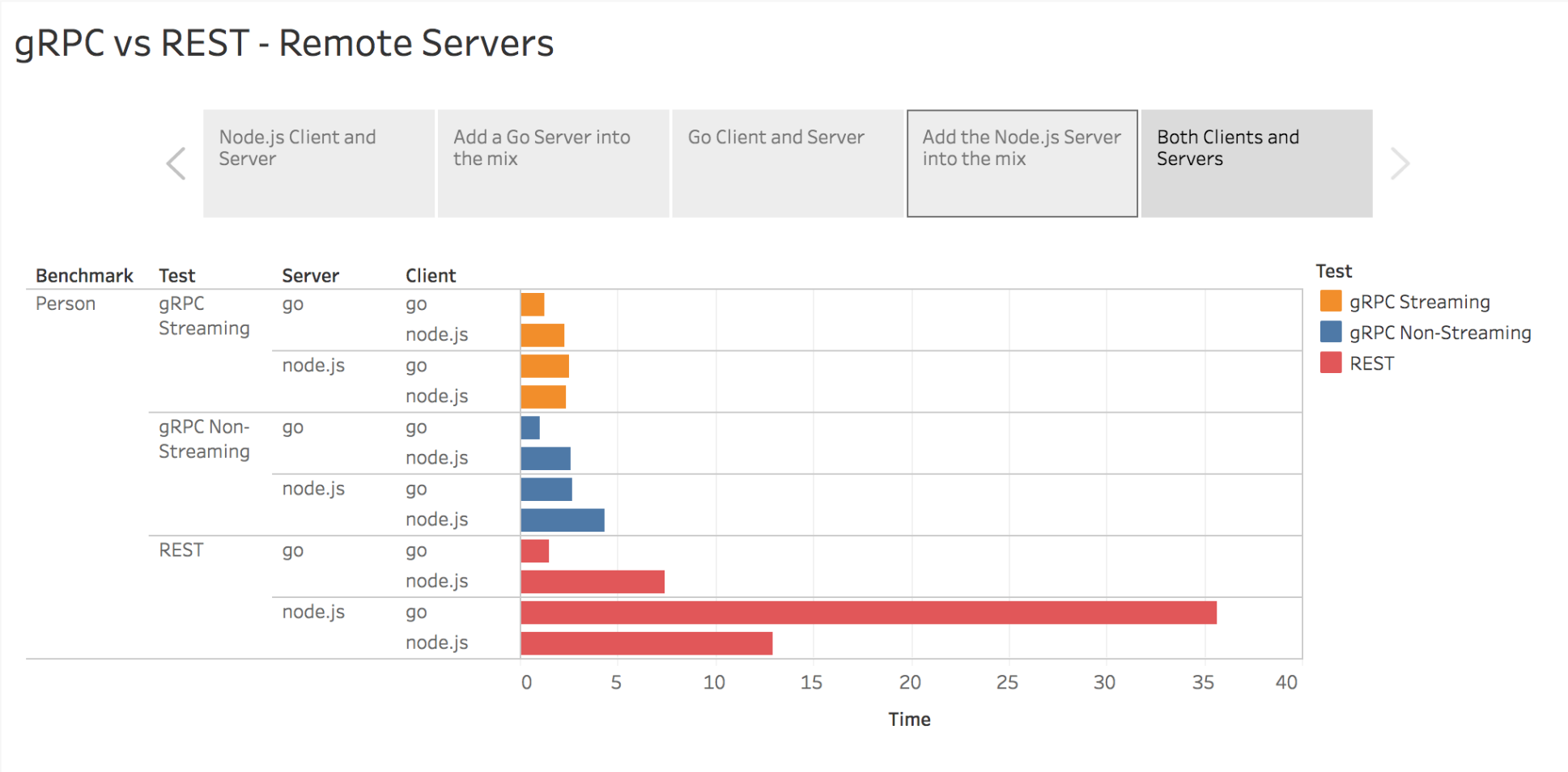
```
- plain text vs binary
- human readable vs binary (compressed)
- (de)serialization vs faster (de)serialization
- Good support vs early adopter
```

- HTTP 1.1 vs HTTP2 (https://http2.github.io/faq/) : Demo HTTP2 (https://http2.golang.org/gophertiles?latency=0)

```
- plain text vs binary (compressed)
- 1 connection by request vs persistent connection
- polling vs streaming
- no TLS by default vs TLS
- Good support vs early adopter
```

- SWAGGER + plugins vs PROTOC + plugins
- Browser support vs no Browser support
- no cascade failure vs cascade failure addressed (https://landing.google.com/sre/book/chapters/addressing-cascading-failures.html)
- N/A vs Tracing, Metadata

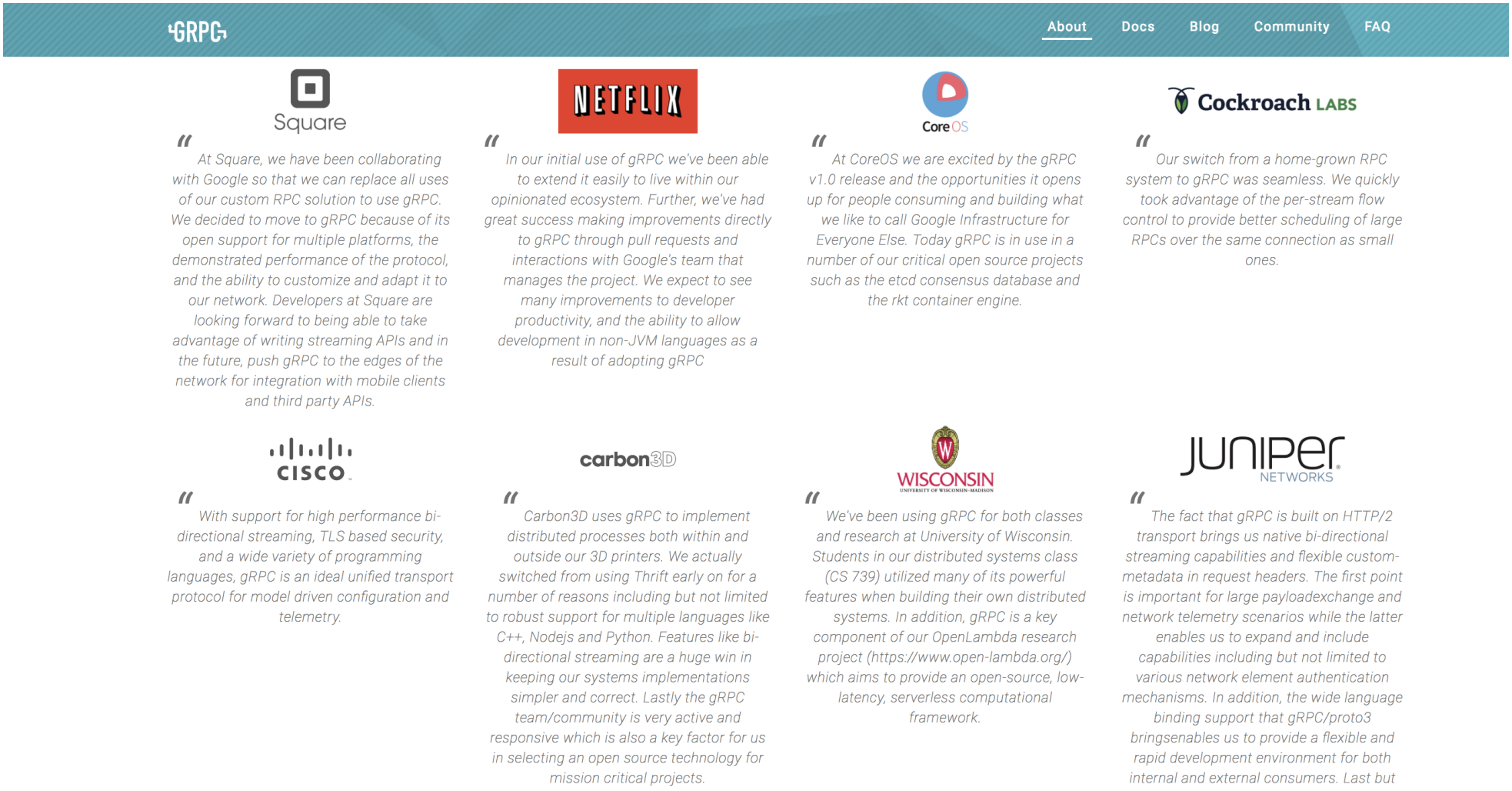
# [15] REST vs gRPC performance




- See gRPC vs REST - Go vs node.js (https://public.tableau.com/profile/sandeep.dinesh#!/vizhome/gRPCvsREST-2016/gRPCvsRest)

# [16] Whos is using gRPC ?

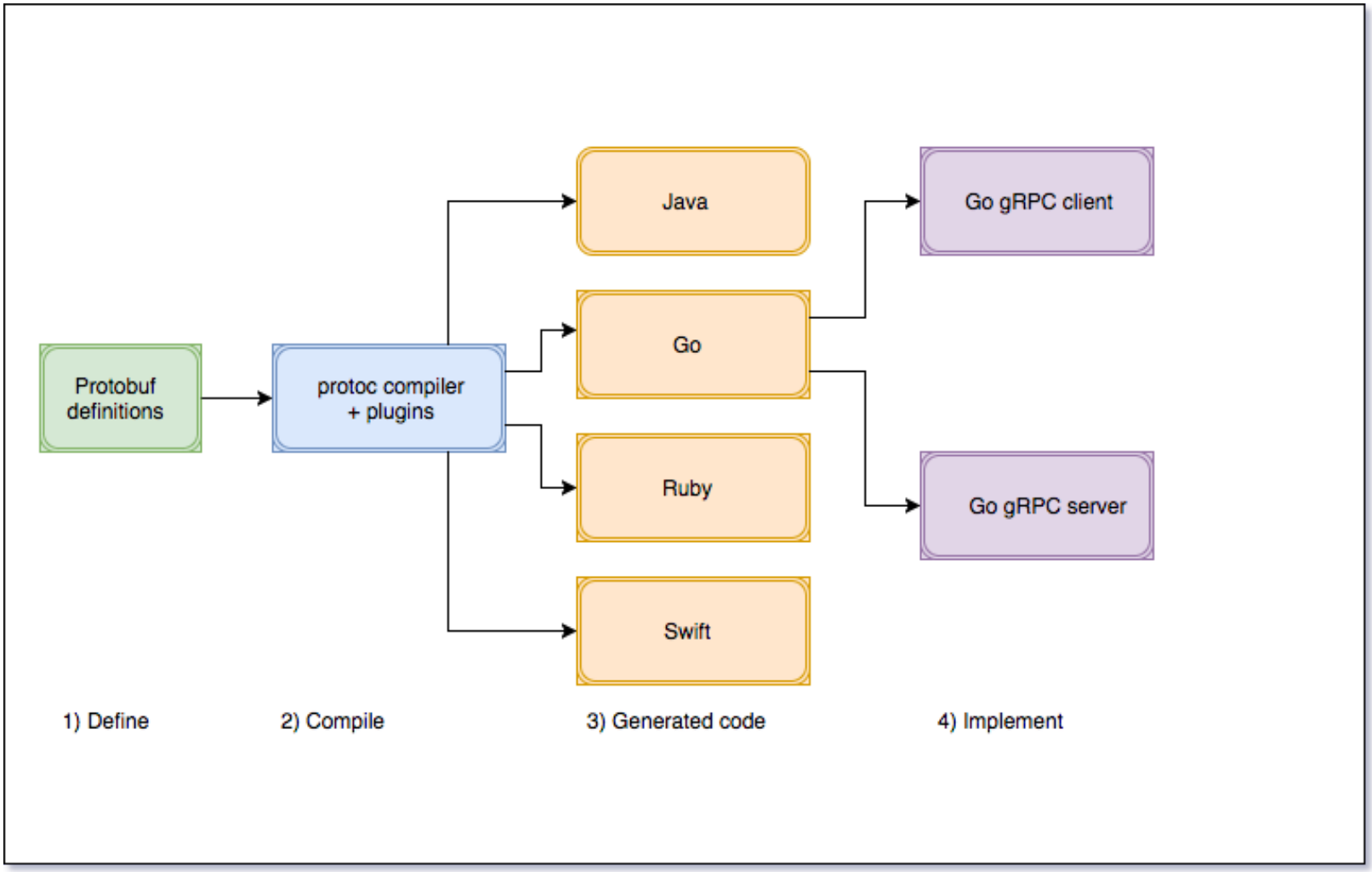
- Google, Kubernetes, Cisco, Docker, ...



# [17] gRPC microservices are the future ?

- Yes !!!
- Yes !!!
- Yes !!!

# [18] Develop gRPC microservices with Go Workflow




- See Book Building Microservices with Go (https://www.amazon.fr/Building-Microservices-Go-Nic-Jackson/dp/1786468662)

# [19] Define services and messages

```
syntax = "proto3";
package superpower;

option java_package = "com.package.protobuf";
option java_multiple_files = true;
option optimize_for = LITE_RUNTIME;

service GoHeroe {
  rpc List(Filter) returns (SuperPowers) {}
  rpc Add(SuperPower) returns (SuperPowers) {}
}

message SuperPower {
  string name = 1;
  SuperPowerCategory cat = 2;
  bool coolPow = 3;
}

message SuperPowers { repeated SuperPower superPow = 1; }
message Filter { SuperPowerCategory category = 1; }
```

- See proto3 Language Guide (https://developers.google.com/protocol-buffers/docs/proto3)

# [20] Generate code with protoc

- Generate Go code with protoc

```
protoc -I ./proto/ ./proto/goheroe.proto --go_out=plugins=grpc:proto
```

- Generate Swift code with protoc

```
protoc -I . backend*.proto \
--proto_path=${GOPATH}/src \
--swift_out=../api/swift \
--swiftgrpc_out=../api/swift \
--swift_opt=Visibility=Public
```

- Generate Java code with protoc

```
protoc -I . backend*.proto \
--proto_path=${GOPATH}/src \
--plugin=protoc-gen-javalite=$(GOPATH)/bin/protoc-gen-javalite \
--plugin=protoc-gen-grpc-java=$(GOPATH)/bin/protoc-gen-grpc-java \
--grpc-java_out=lite:../api/lpGrpc.jar \
--javalite_out=../api/lpGrpc.jar
```

# [21] gRPC server

```
func main() {

        lis, err := net.Listen("tcp", grpcPort)
        if err != nil {
            fmt.Printf("failed to listen: %v\n", err)
        }

        grpcServer := grpc.NewServer()

        srv, err := app.NewGoHeroeServer()
        if err != nil {
            fmt.Printf("failed to create GoHeroeServer : %s\n", err)
        }

        superpower.RegisterGoHeroeServer(grpcServer, srv)

        reflection.Register(grpcServer)

        fmt.Printf("starting server on port : %s\n", grpcPort)
        grpcServer.Serve(lis)
    }
```

# [22] Implement gRPC server

```
// GoHeroeServer implements GoHeroeServer.
type GoHeroeServer struct {
}

// NewGoHeroeServer returns a new GoHeroeServer.
func NewGoHeroeServer() (*GoHeroeServer, error) {
    return &GoHeroeServer{}, nil
}

// List super powers with a filter
func (server GoHeroeServer) List(ctx context.Context, filter *superpower.Filter)
(*superpower.SuperPowers, error) {

    if filter.Category == 0 {
        return &superpower.SuperPowers{SuperPow: heroes}, nil
    }
    return &superpower.SuperPowers{SuperPow: filterHeores(heroes,
        func(heroe *superpower.SuperPower) bool { return heroe.Cat == filter.Category })}, nil
}
```

# [23] Implement gRPC client

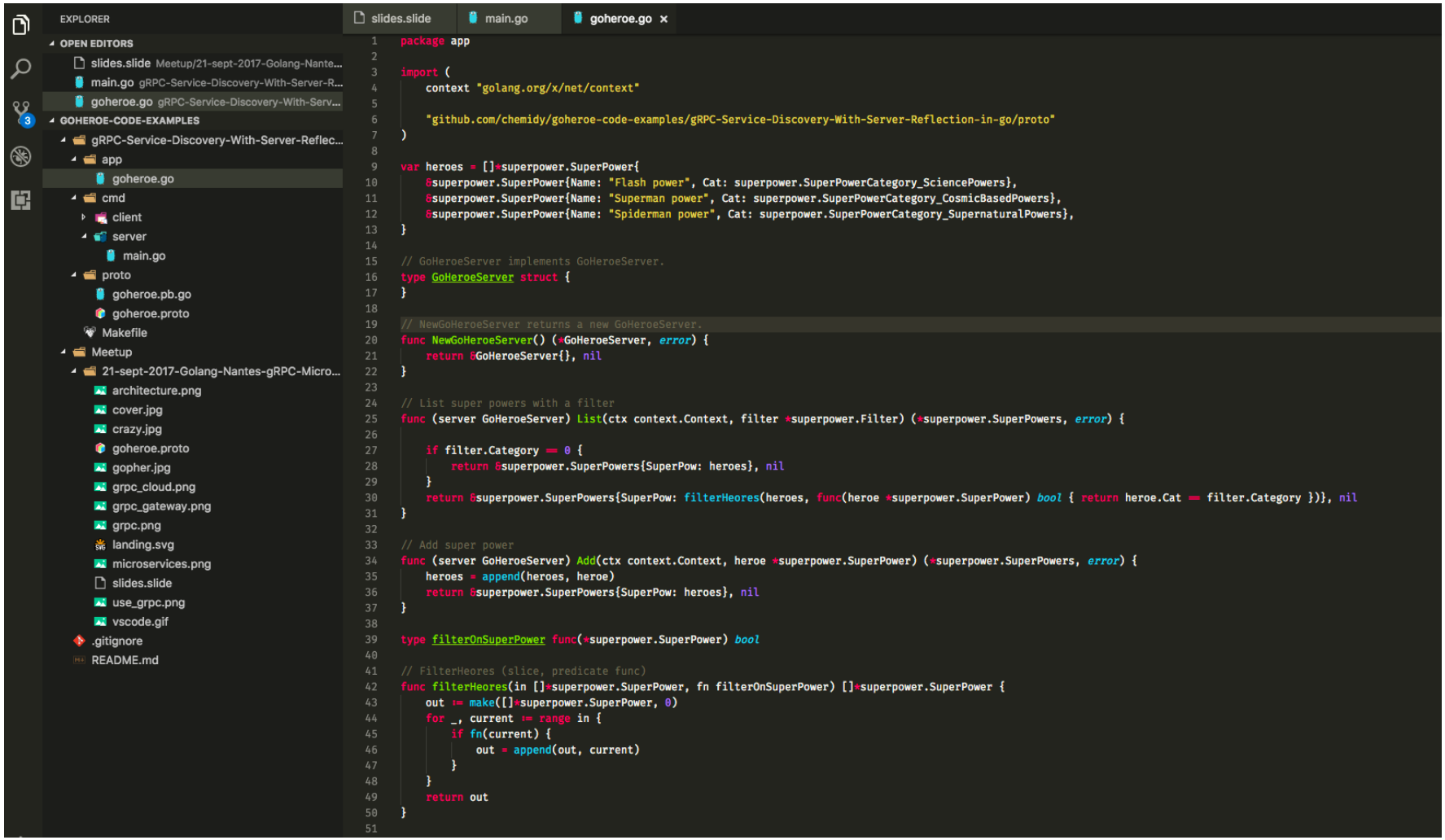
```
func main() {
    conn, err := grpc.Dial("localhost:44444", grpc.WithInsecure())
    if err != nil {
        log.Fatalf("did not connect: %v", err)
    }
    defer conn.Close()
    ctx := context.Background()
    client := superpower.NewGoHeroeClient(conn)
    filter := superpower.Filter{Category: superpower.SuperPowerCategory_CosmicBasedPowers}

    result, err := client.List(ctx, &filter)
    if err != nil {
        log.Fatalf("Error on List : %v", err)
    }
    log.Printf("List: %v", result)

    power := superpower.SuperPower{Name: "mickey power", Cat: 0, CoolPow: true}

    result, err = client.Add(ctx, &power)
    if err != nil {
        log.Fatalf("Error on List : %v", err)
    }
    log.Printf("List: %v", result)
}
```

# [24] Demo protoc



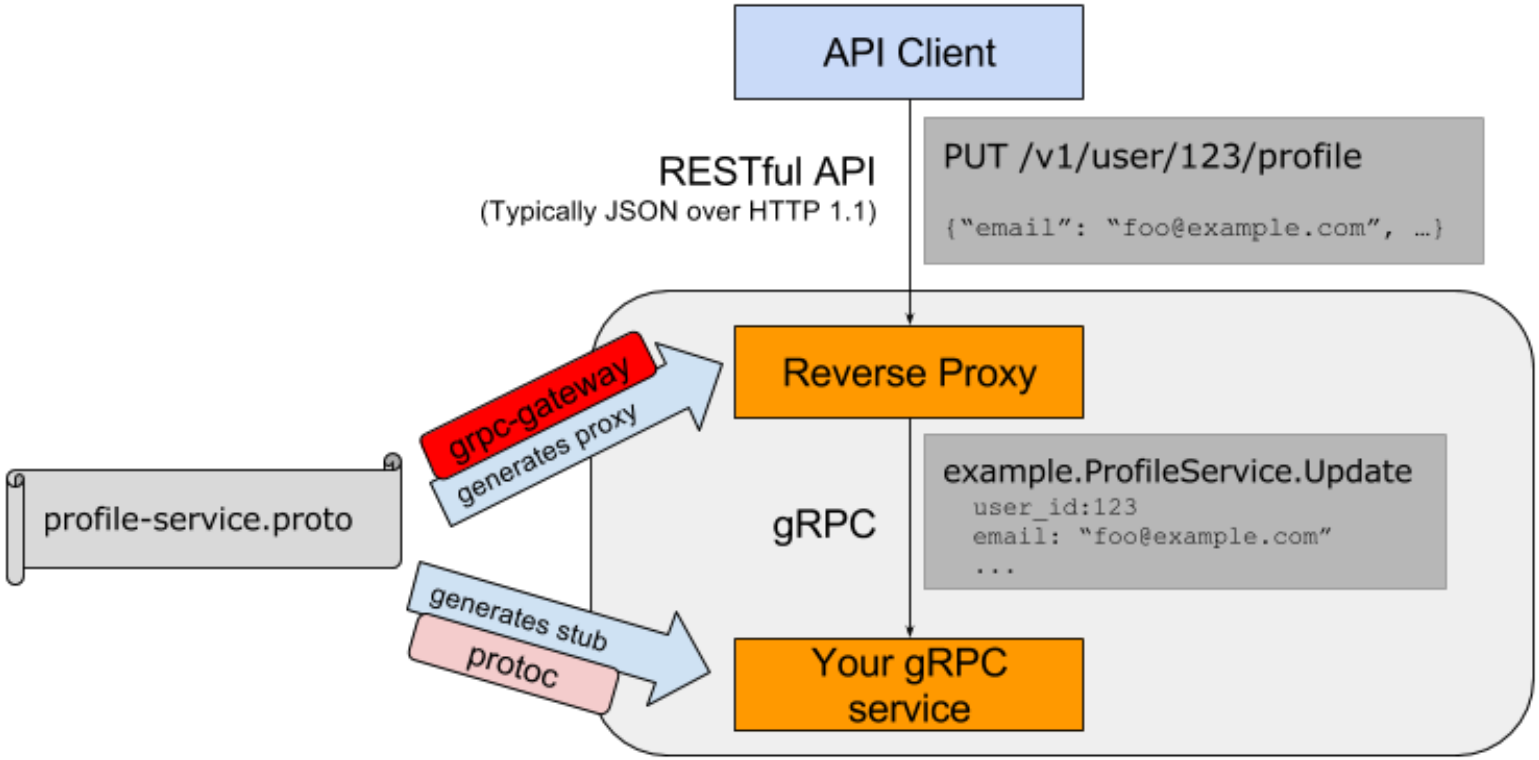
# [25] Call gRPC services from Mobile (IOS,Android)

```
@Override
protected String doInBackground(Void... nothing) {
    try {
        mChannel = ManagedChannelBuilder.forAddress(mHost, mPort)
            .usePlaintext(true)
            .build();
        GreeterGrpc.GreeterBlockingStub stub = GreeterGrpc.newBlockingStub(mChannel);
        HelloRequest message = HelloRequest.newBuilder().setName(mMessage).build();
        HelloReply reply = stub.sayHello(message);
        return reply.getMessage();
    } catch (Exception e) {
        StringWriter sw = new StringWriter();
        PrintWriter pw = new PrintWriter(sw);
        e.printStackTrace(pw);
        pw.flush();
        return String.format("Failed... : %n%s", sw);
    }
}
```

- See gRPC doc for android (https://grpc.io/docs/tutorials/basic/android.html)
- See Hello World for android (https://github.com/grpc/grpc-java/tree/v1.6.x/examples/android)

# [26] Call gRPC services from Angular with a gateway

- grpc-gateway is a plugin of protoc. It reads gRPC service definition, and generates a reverse-proxy server which translates a RESTful JSON API into gRPC




- See Go grpc-gateway : gRPC to JSON proxy generator (https://github.com/grpc-ecosystem/grpc-gateway)

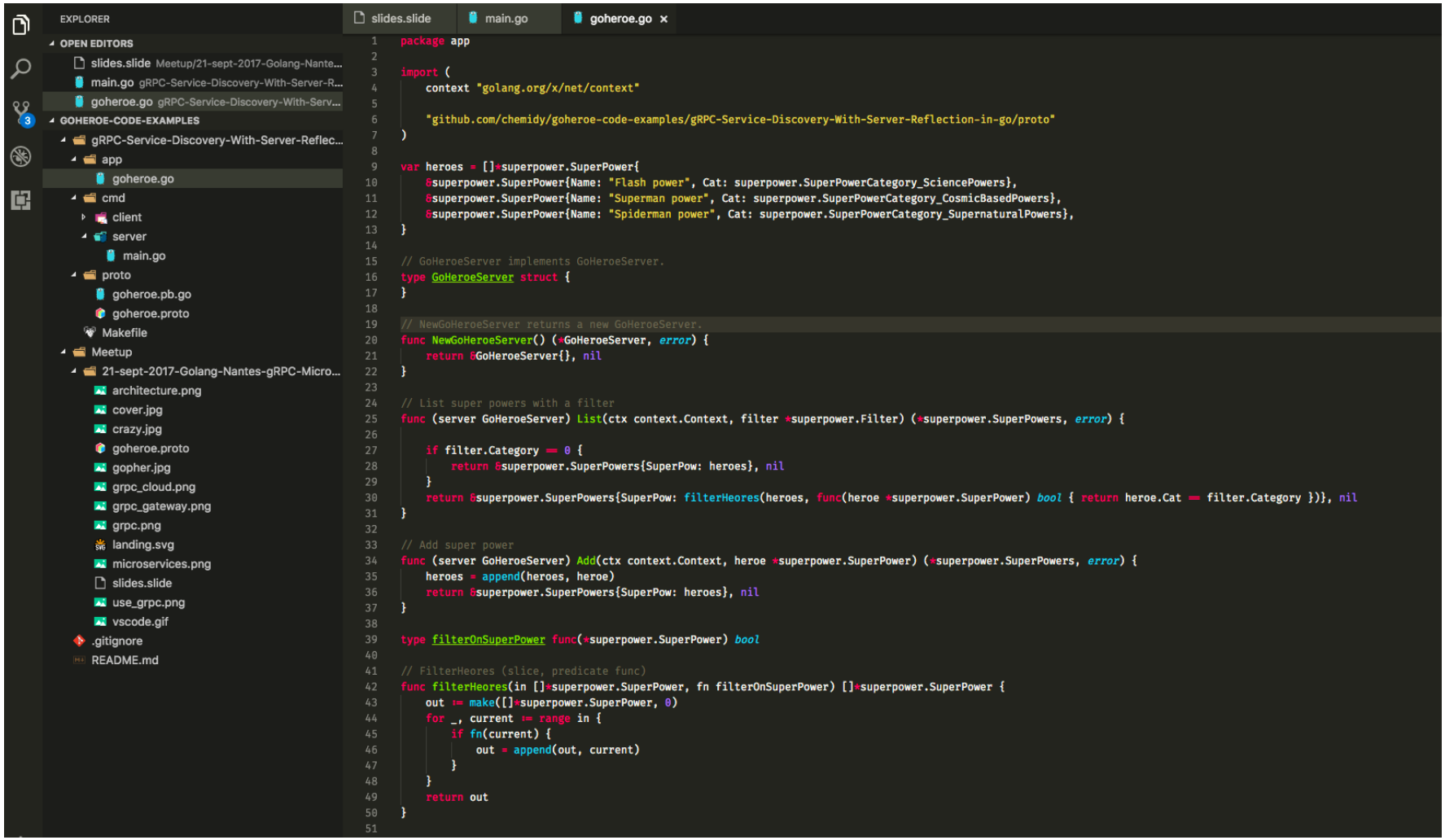
# [27] Manage local microservices

- Edward is a very nice tool to manage a large amount of microservices with automatically rebuild on edit.
- A quick list of features :

```
- Start multiple services (group)
- Restart a service or multiple services
- Check status of running services
- Check logs of running services
- Auto-generate service configuration for: go / Docker
- Auto-restart on
```

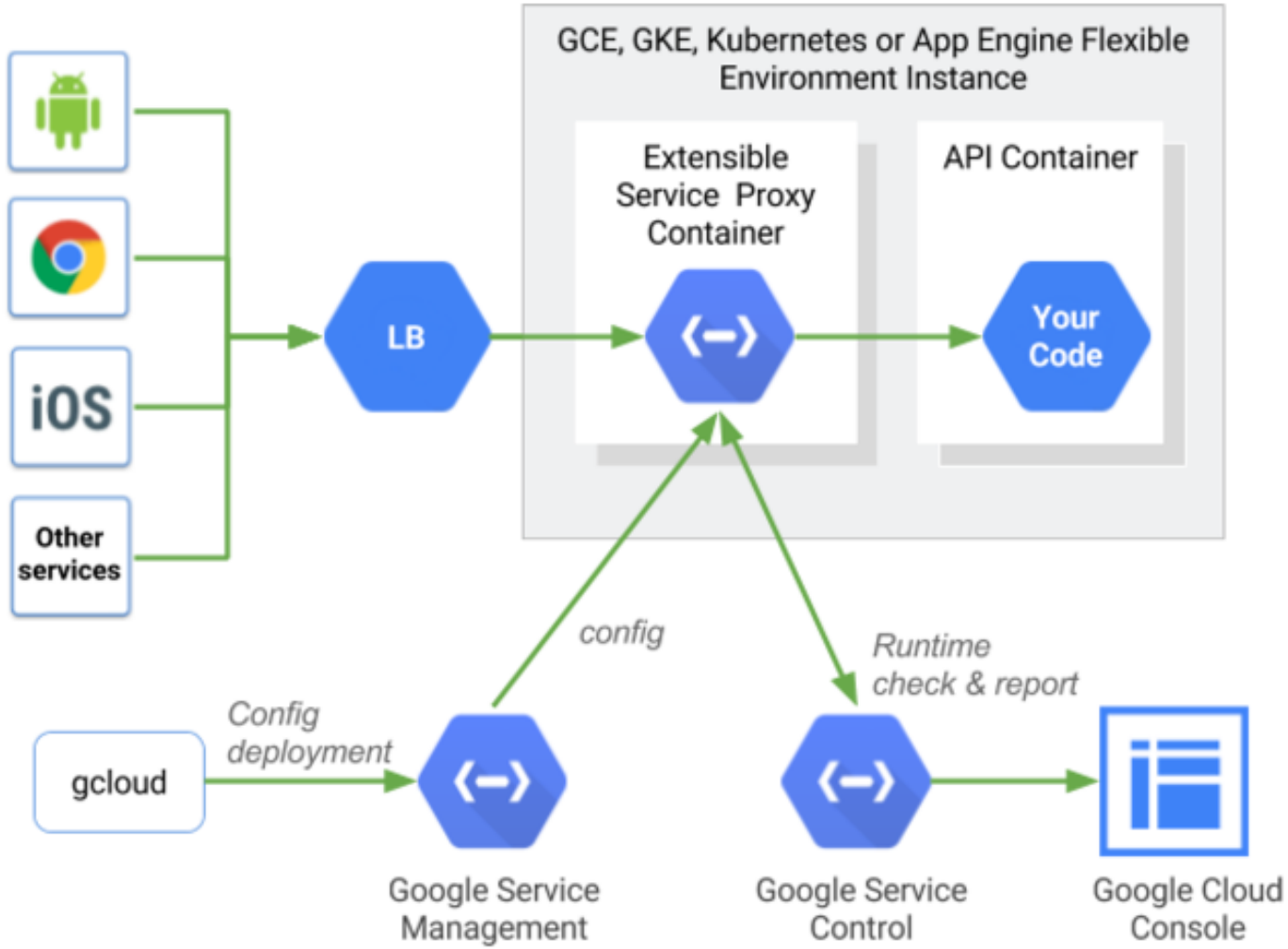
- See Edward Github (https://github.com/yext/edward)
- See Edward docs (http://engblog.yext.com/edward/)

# [28] Demo Edward



# [29] Deploy gRPC microservices on GKE

# [30] Cloud endpoints



# [31] Deploy endpoints

- Generate config file

```
protoc -I proto --include_imports --include_source_info back.proto --descriptor_set_out out.pb
```

- Deploy Google Endpoint

```
gcloud service-management deploy endpoints.pb ../config/endpoints/api_config.yaml
```

- api_config.yaml

```
type: google.api.Service
config_version: 3
name: hello.endpoints.my_google_project.cloud.goog
title: auth gRPC API
apis:
- name: api.Hello
usage:
 rules:
 # Allow unregistered calls for all methods.
 - selector: "*"
   allow_unregistered_calls: true
```

# [32] Build Docker images

- Build image and push to gc.io

```
gcloud container builds submit . \
--config=config/docker/cloudbuild.yaml \
--substitutions _VERSION=${V},_SERVICE=${S}
```

- cloudbuild.yaml

```
steps:
- name: 'gcr.io/cloud-builders/docker'
 args: ['build', '-f', 'config/docker/Dockerfile.${_SERVICE}',
 '-t', 'gcr.io/$PROJECT_ID/go-grpc-${_SERVICE}:${_VERSION}', '.']
 dir: '.'
 images: ['gcr.io/$PROJECT_ID/go-grpc-${_SERVICE}:${_VERSION}']
```

# [33] Deploy to GKE

- deploy Load Balancer

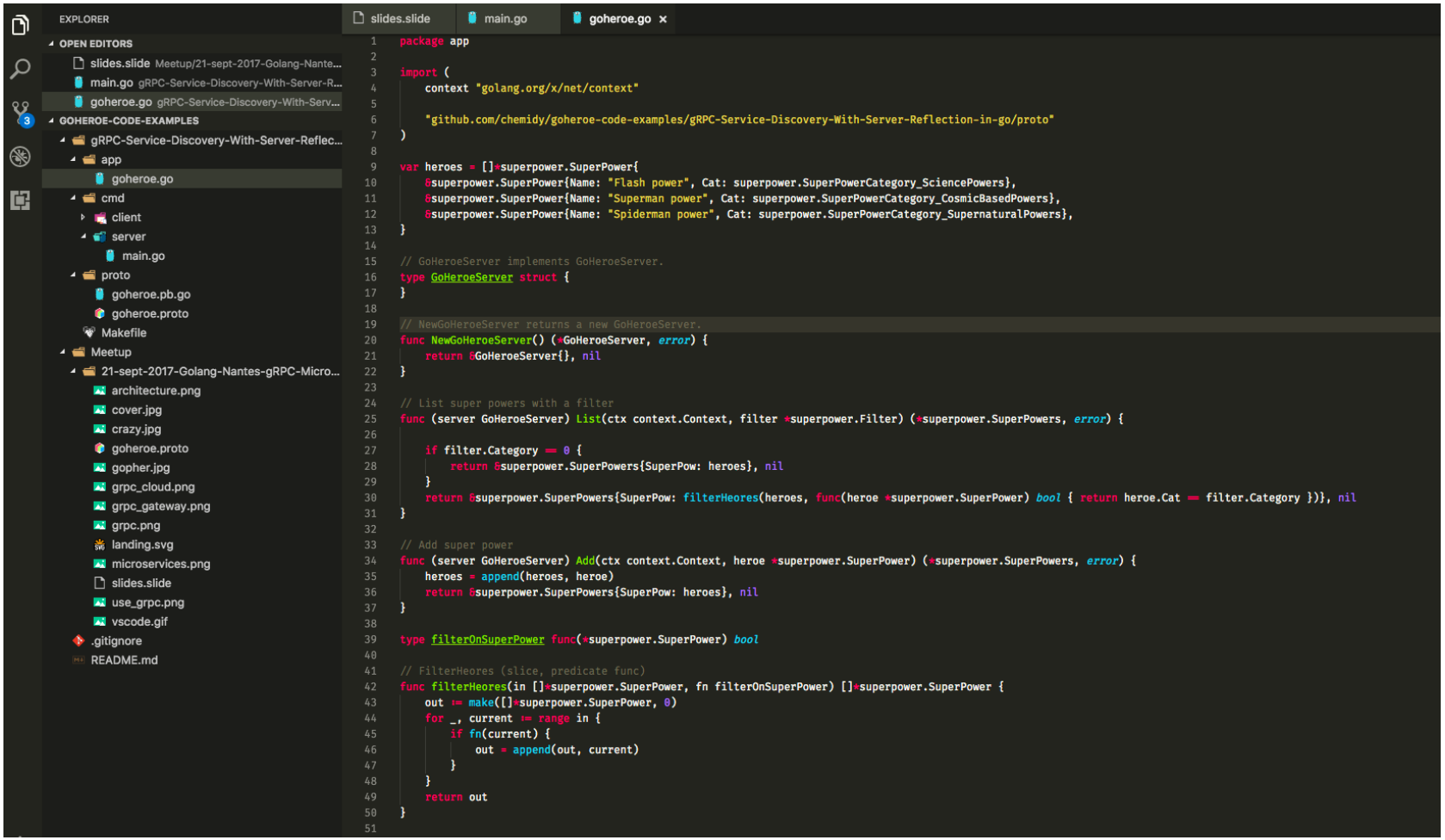
```
kubectl create -f myconfig.yaml
```

- lb.yaml

```
apiVersion: v1
```

```
kind: Service
metadata:
 name: lb-grpc-hello
spec:
 type: LoadBalancer
 loadBalancerIP: 118.218.118.218
 ports:
 - name: http
   port: 80
   targetPort: 9000
   protocol: TCP
 selector:
   app: grpc-hello
```

# [34] Demo K8




- See Building high performance microservices with Kubernetes, Go, and gRPC

  (https://www.youtube.com/watch?v=YiNt4kUnnIM)

# [35] Conclusion

?

# Thank you

Golang Nantes Meetup
21 September 2017

Cyrille Hemidy
LivingPackets (http://LivingPackets.com)

@chemidy (http://twitter.com/chemidy)
+GoHeroe (http://www.goheroe.org)