

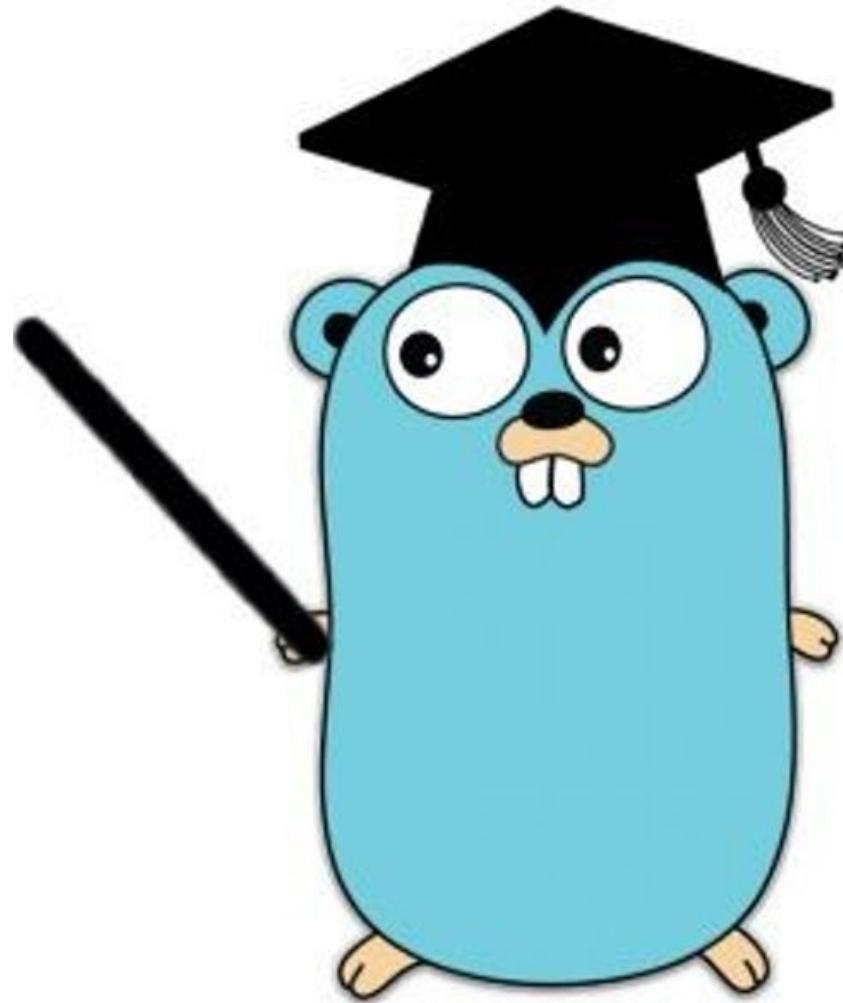
gRPC microservices are the future ?

Build scalable gRPC services with go deployed to GKE

Golang Nantes Meetup
21 September 2017

Cyrille Hemidy
[LivingPackets](http://LivingPackets.com) (<http://LivingPackets.com>)

[1] Hello Gophers



[2] @work

[Contribute now](#)[Company](#) ▾[FAQ](#)[Contact us](#) [en](#) ▾[Login](#)The landing page for LivingPACKETS features a large, high-angle photograph of a city skyline at dusk, likely New York City, with numerous skyscrapers and streetlights visible.

International Deliveries in Hours
On-demand Crowd Delivery Technology

[Watch video](#)

A new Profit-Sharing Model
Get Unlimited Revenue

[Contribute now](#)

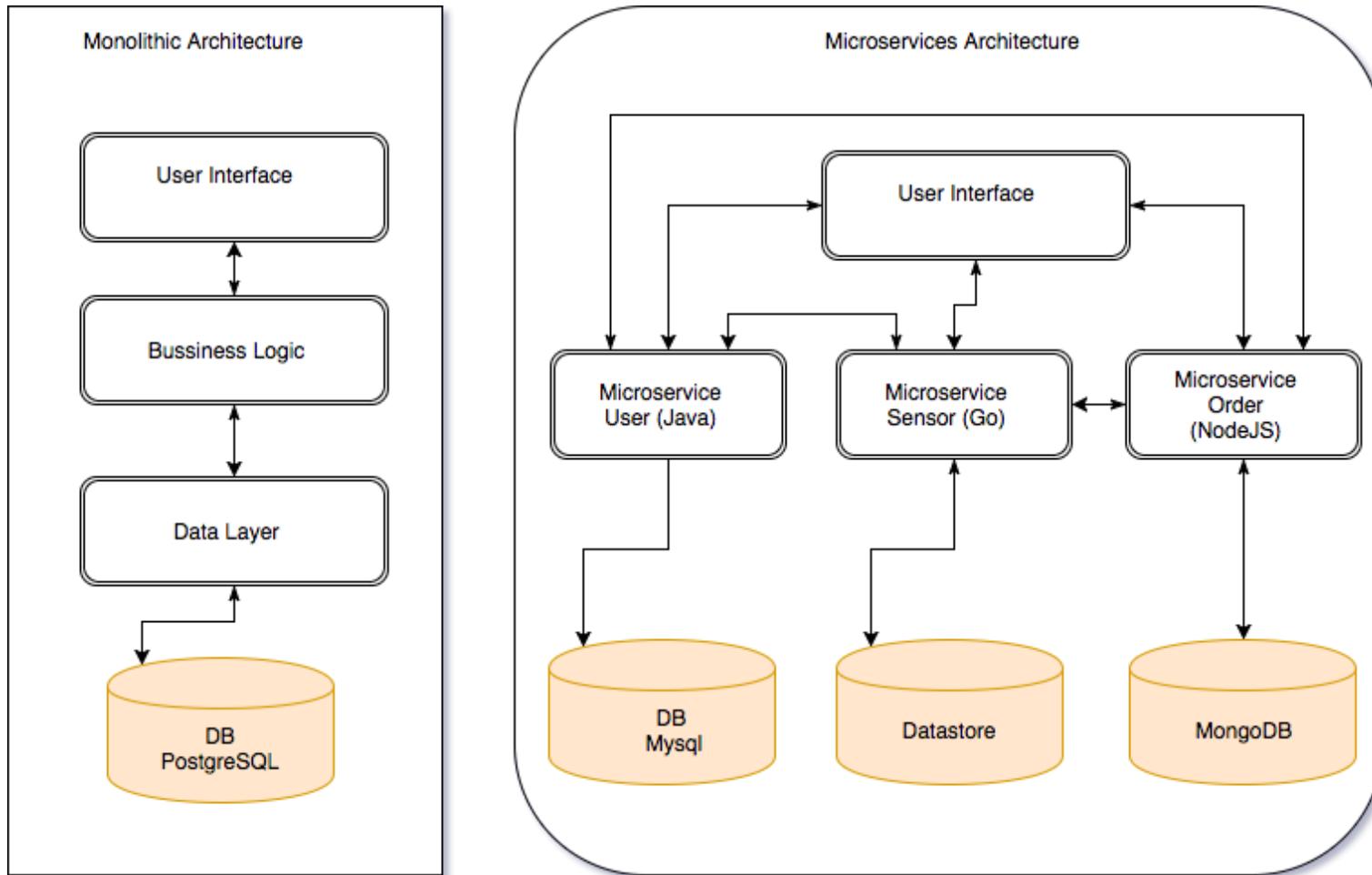
- See [LivingPACKETS.com](https://www.livingpackets.com/en) (<https://www.livingpackets.com/en>)

[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

Architecture

Monolith Architecture

A single application with spaghetti code and strong dependencies between services.

Microservices Architecture

A lot of small Services that work independently and communicate with each other (REST, RPC, ...).

Scalability

Each service scales independently.
Each service scales on demand.

Agility

Each service can be changed independently.
Easy to add new services.

Development

Each service can be developed in a different programming language

Maintenance

Small code bases

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

[7] A little bit of history

- 2011 : [Protocol Buffers 2](https://developers.google.com/protocol-buffers/) (<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) (<https://static.googleusercontent.com/media/research.google.com/fr//pubs/archive/43438.pdf>) => Large-scale cluster management => [Kubernetes](https://kubernetes.io/) (<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) (<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/) (<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) (<https://github.com/apple/swift-protobuf>)
- Jan 2017 : [Grpc Swift](https://github.com/grpc/grpc-swift) (<https://github.com/grpc/grpc-swift>)
- Apr 2017 : [Google Endpoints](https://cloud.google.com/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/) (<https://grpc.io/>)

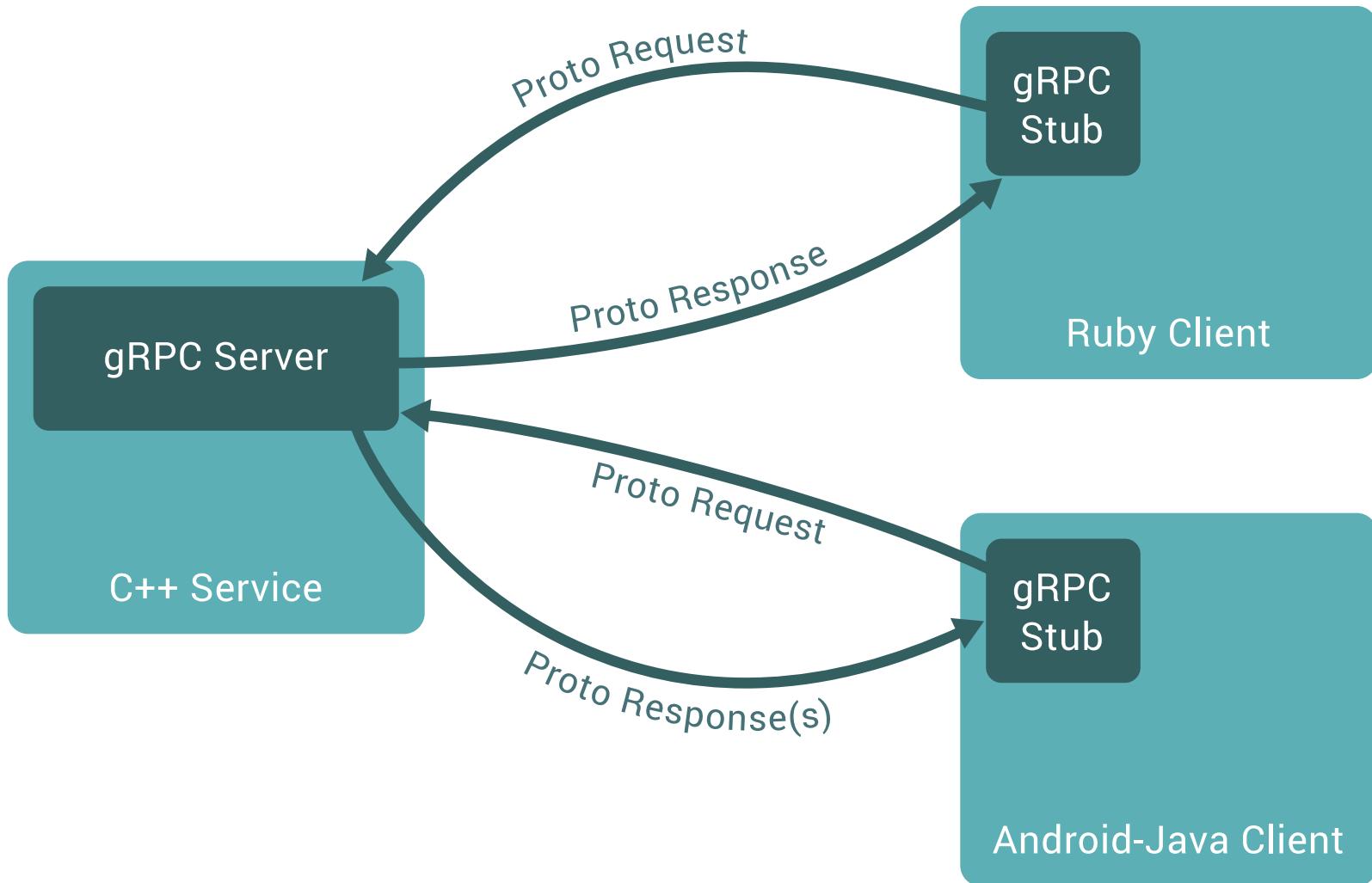
[9] gRPC speaks a lot of languages

- 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/) (<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

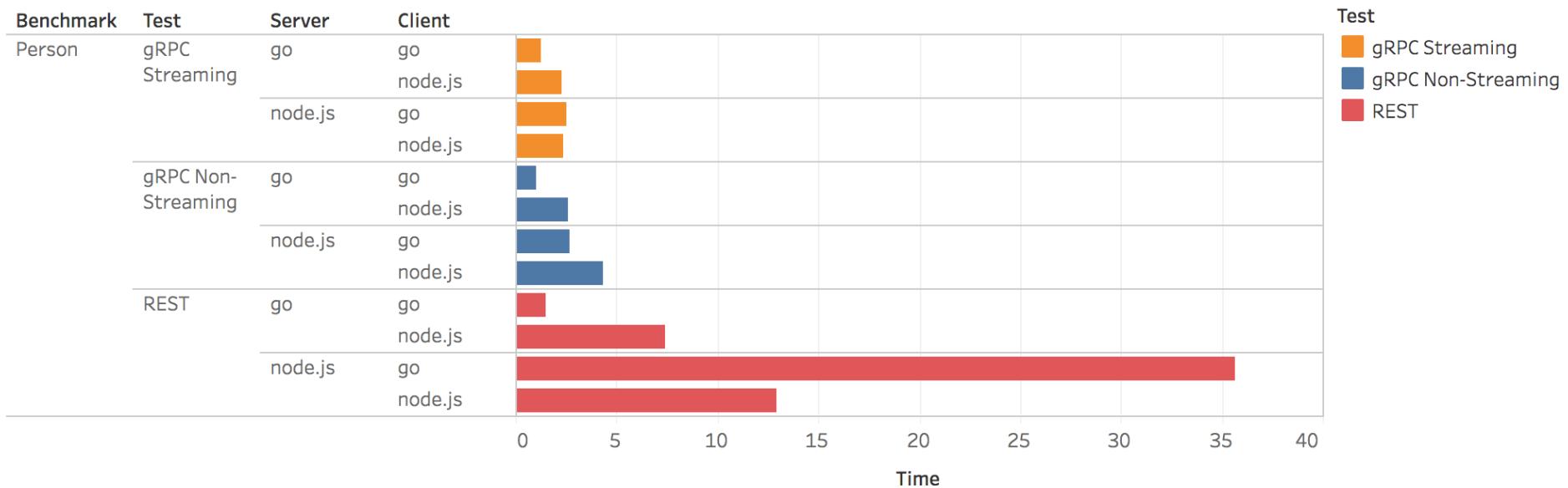
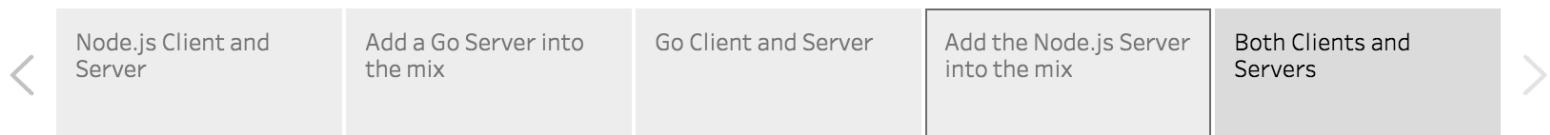
- Browser support vs no Browser support

- 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

gRPC vs REST - Remote Servers



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

[16] Whos is using gRPC ?

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

Square

" At Square, we have been collaborating with Google so that we can replace all uses of our custom RPC solution to use gRPC. We decided to move to gRPC because of its open support for multiple platforms, the demonstrated performance of the protocol, and the ability to customize and adapt it to our network. Developers at Square are looking forward to being able to take advantage of writing streaming APIs and in the future, push gRPC to the edges of the network for integration with mobile clients and third party APIs.



NETFLIX

" In our initial use of gRPC we've been able to extend it easily to live within our opinionated ecosystem. Further, we've had great success making improvements directly to gRPC through pull requests and interactions with Google's team that manages the project. We expect to see many improvements to developer productivity, and the ability to allow development in non-JVM languages as a result of adopting gRPC



CoreOS

" At CoreOS we are excited by the gRPC v1.0 release and the opportunities it opens up for people consuming and building what we like to call Google Infrastructure for Everyone Else. Today gRPC is in use in a number of our critical open source projects such as the etcd consensus database and the rkt container engine.



Cockroach LABS

" Our switch from a home-grown RPC system to gRPC was seamless. We quickly took advantage of the per-stream flow control to provide better scheduling of large RPCs over the same connection as small ones.



CISCO

" With support for high performance bi-directional streaming, TLS based security, and a wide variety of programming languages, gRPC is an ideal unified transport protocol for model driven configuration and telemetry.



carbon3D

" Carbon3D uses gRPC to implement distributed processes both within and outside our 3D printers. We actually switched from using Thrift early on for a number of reasons including but not limited to robust support for multiple languages like C++, Nodejs and Python. Features like bi-directional streaming are a huge win in keeping our systems implementations simpler and correct. Lastly the gRPC team/community is very active and responsive which is also a key factor for us in selecting an open source technology for mission critical projects.



WISCONSIN
UNIVERSITY OF WISCONSIN-MADISON

" We've been using gRPC for both classes and research at University of Wisconsin. Students in our distributed systems class (CS 739) utilized many of its powerful features when building their own distributed systems. In addition, gRPC is a key component of our OpenLambda research project (<https://www.open-lambda.org/>) which aims to provide an open-source, low-latency, serverless computational framework.



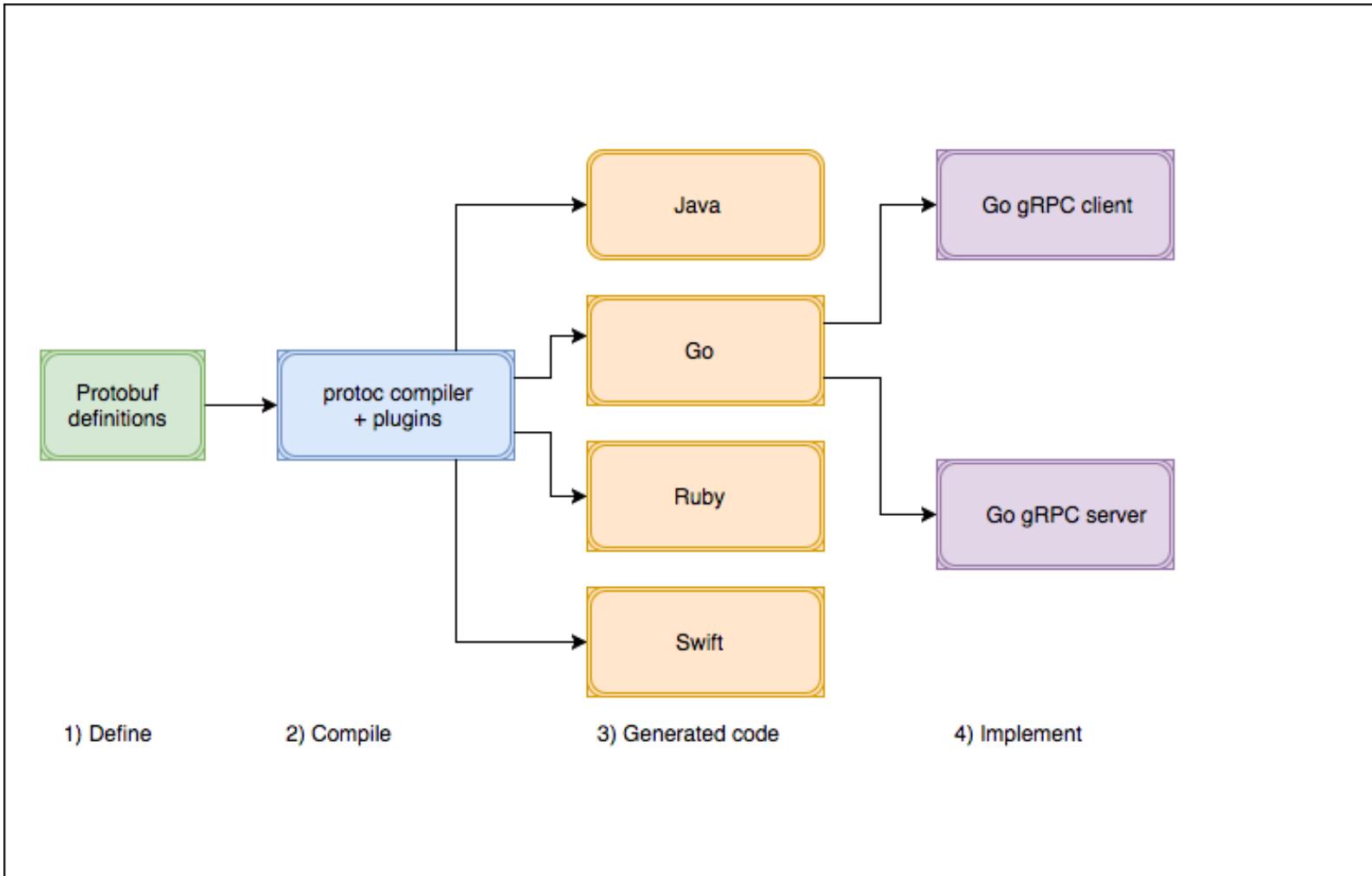
JUNIPER NETWORKS

" The fact that gRPC is built on HTTP/2 transport brings us native bi-directional streaming capabilities and flexible custom-metadata in request headers. The first point is important for large payload exchange and network telemetry scenarios while the latter enables us to expand and include capabilities including but not limited to various network element authentication mechanisms. In addition, the wide language binding support that gRPC/proto3 brings enables us to provide a flexible and rapid development environment for both internal and external consumers. Last but

[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) (<https://developers.google.com/protocol-buffers/docs/proto3>)

[20] Generate code with protoc

- Generate Go code with protoc

```
protoc -I ./proto/ ./proto/gohero.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(hero *superpower.SuperPower) bool { return hero.Cat == filter.Category })}, nil  
}
```

[23] Implement gRPC client

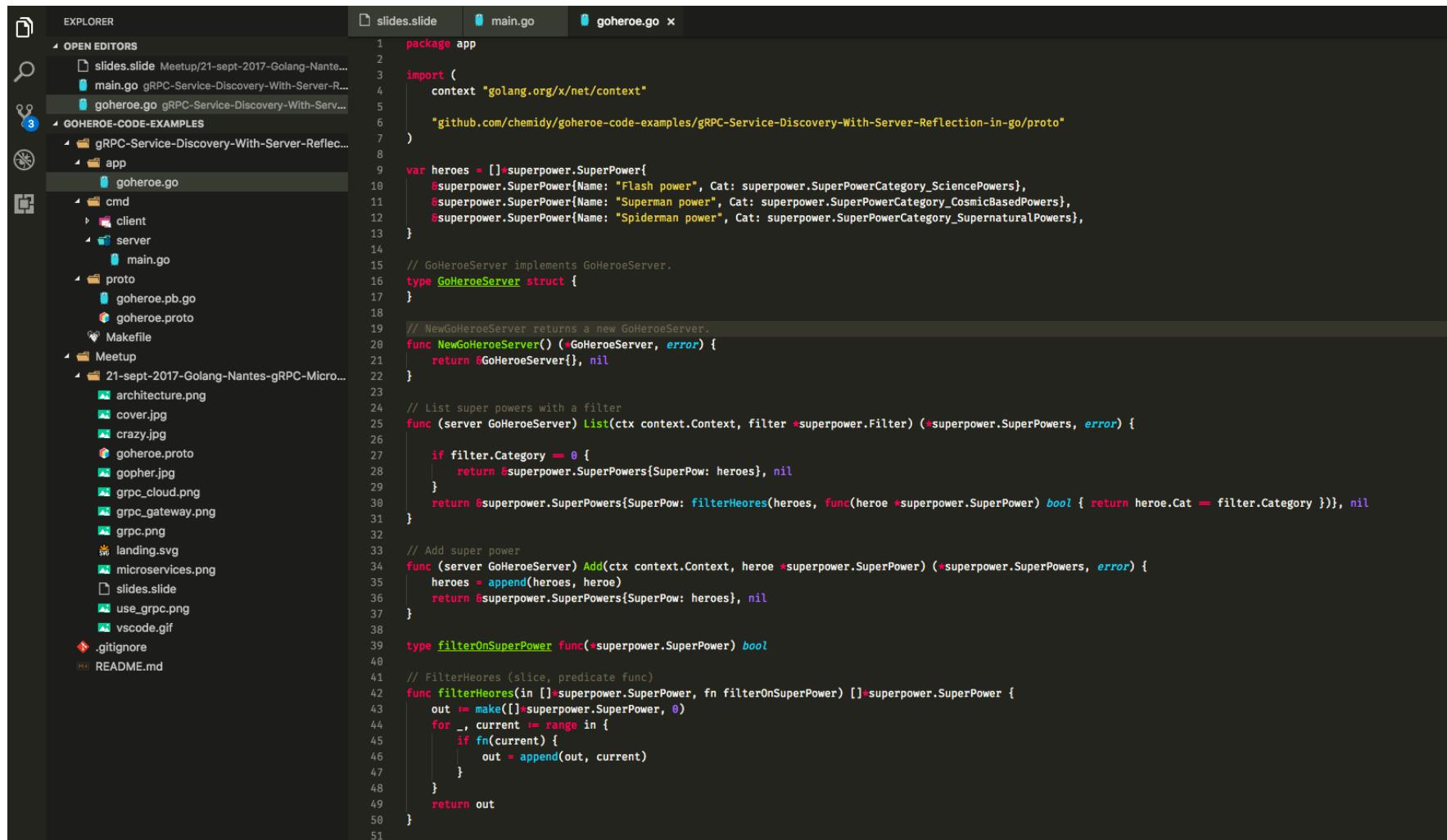
```
func main() {
    conn, err := grpc.Dial("localhost:44444", grpc.WithInsecure())
    if err != nil {
        log.Fatalf("did not connect: %v", err)
    }
    defer conn.Close()
    ctx := context.Background()
    client := superpower.NewGoHeroeClient(conn)
    filter := superpower.Filter{Category: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



```

EXPLORER
  OPEN EDITORS
    slides.slide Meetup/21-sept-2017-Golang-Nantes...
    main.go gRPC-Service-Discovery-With-Server-R...
    goheroe.go gRPC-Service-Discovery-With-Serv...
  GOHEROE-CODE-EXAMPLES
    gRPC-Service-Discovery-With-Server-Reflec...
      app
        goheroe.go
      cmd
        client
        server
          main.go
      proto
        goheroe.pb.go
        goheroe.proto
      Makefile
    Meetup
      21-sept-2017-Golang-Nantes-gRPC-Micro...
        architecture.png
        cover.jpg
        crazy.jpg
        goheroe.proto
        gopher.jpg
        grpc_cloud.png
        grpc_gateway.png
        grpc.png
        landing.svg
        microservices.png
        slides.slide
        use_grpc.png
        vscode.gif
      .gitignore
      README.md

slides.slide  main.go  goheroe.go x

1 package app
2
3 import (
4   context "golang.org/x/net/context"
5
6   "github.com/chemidy/goheroe-code-examples/gRPC-Service-Discovery-With-Server-Reflection-in-go/proto"
7 )
8
9 var heroes = []superpower.SuperPower{
10   &superpower.SuperPower{Name: "Flash power", Cat: superpower.SuperPowerCategory_SciencePowers},
11   &superpower.SuperPower{Name: "Superman power", Cat: superpower.SuperPowerCategory_CosmicBasedPowers},
12   &superpower.SuperPower{Name: "Spiderman power", Cat: superpower.SuperPowerCategory_SupernaturalPowers},
13 }
14
15 // GoHeroeServer implements GoHeroeServer.
16 type GoHeroeServer struct {
17 }
18
19 // NewGoHeroeServer returns a new GoHeroeServer.
20 func NewGoHeroeServer() (*GoHeroeServer, error) {
21   return &GoHeroeServer{}, nil
22 }
23
24 // List super powers with a filter
25 func (server GoHeroeServer) List(ctx context.Context, filter *superpower.Filter) (*superpower.SuperPowers, error) {
26
27   if filter.Category == 0 {
28     return &superpower.SuperPowers{SuperPow: heroes}, nil
29   }
30   return &superpower.SuperPowers{SuperPow: filterHeores(heroes, func(hero *superpower.SuperPower) bool { return hero.Cat == filter.Category })}, nil
31 }
32
33 // Add super power
34 func (server GoHeroeServer) Add(ctx context.Context, hero *superpower.SuperPower) (*superpower.SuperPowers, error) {
35   heroes = append(heroes, hero)
36   return &superpower.SuperPowers{SuperPow: heroes}, nil
37 }
38
39 type filterOnSuperPower func(*superpower.SuperPower) bool
40
41 // FilterHeores (slice, predicate func)
42 func filterHeores(in []superpower.SuperPower, fn filterOnSuperPower) []superpower.SuperPower {
43   out := make([]superpower.SuperPower, 0)
44   for _, current := range in {
45     if fn(current) {
46       out = append(out, current)
47     }
48   }
49   return out
50 }
51

```

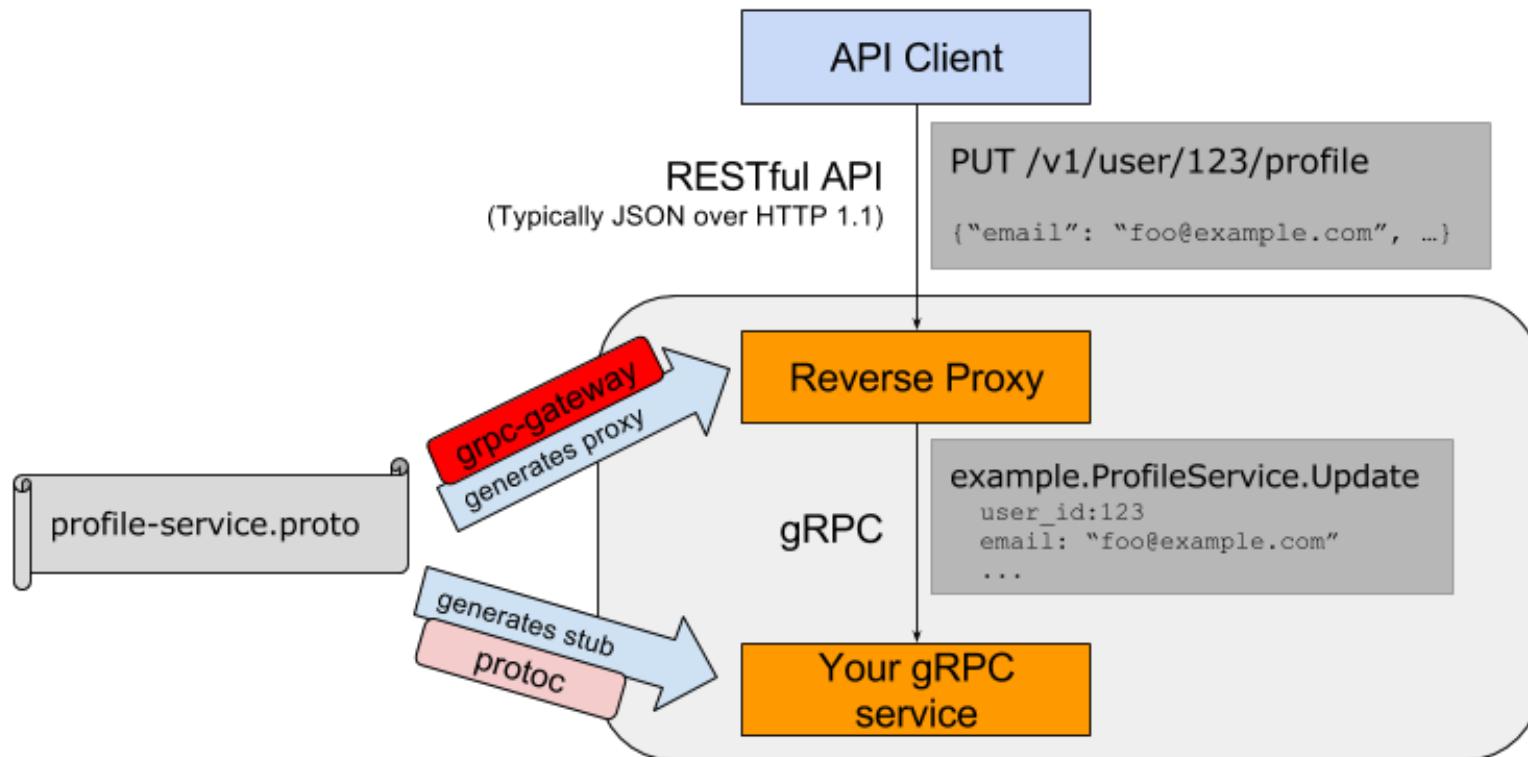
[25] Call gRPC services from Mobile (IOS,Android)

```
@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) (<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) (<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) (<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) (<https://github.com/yext/edward>)
- See [Edward docs](http://engblog.yext.com/edward/) (<http://engblog.yext.com/edward/>)

[28] Demo Edward

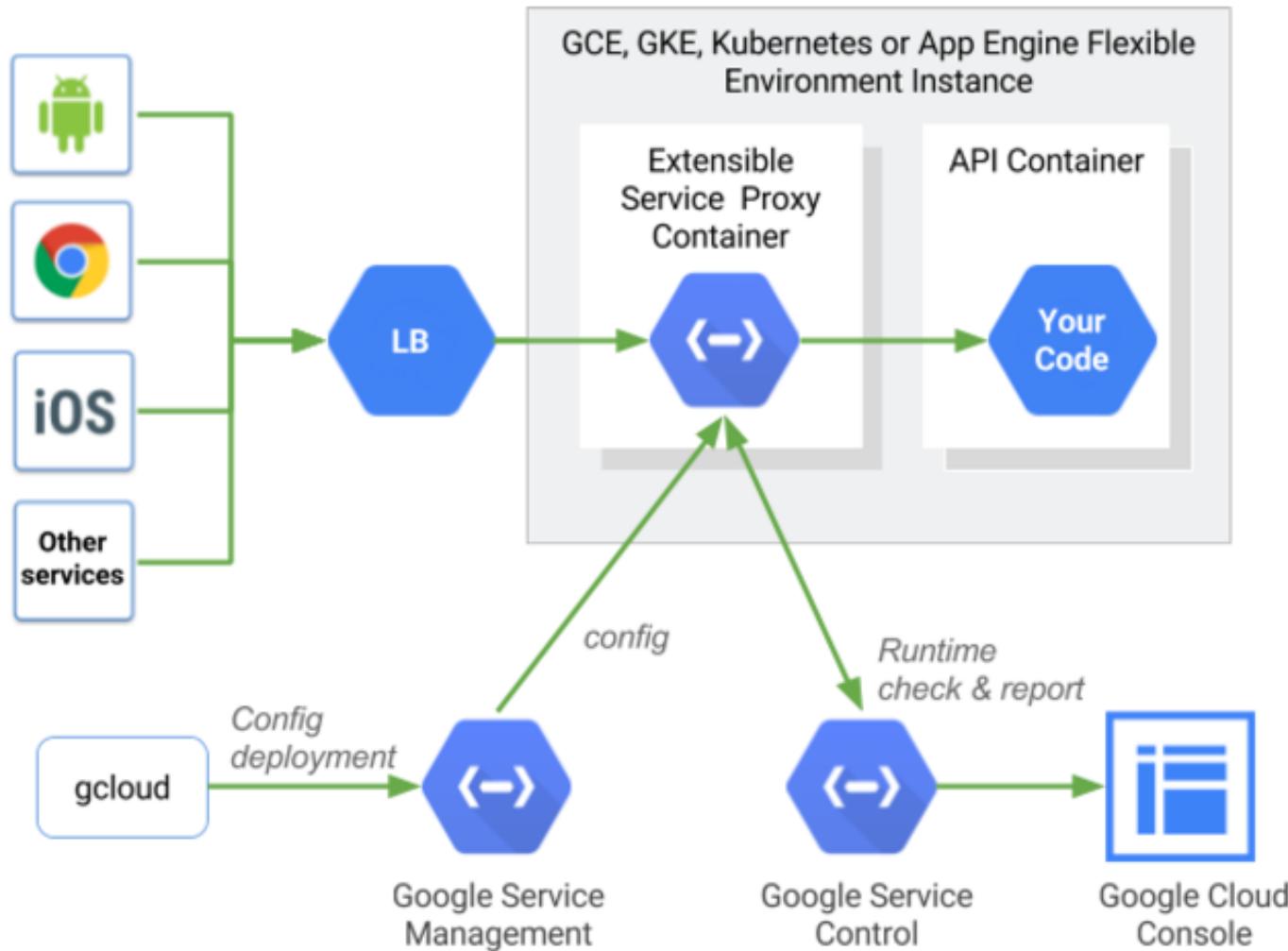
The screenshot shows the VS Code interface with the following details:

- Left Sidebar (Explorer):** Shows the project structure under "OPEN EDITORS". The "gohero" folder contains:
 - slides.slide
 - main.go
 - gohero.go
 - cmd
 - client
 - server
 - main.go
 - proto
 - gohero.pb.go
 - gohero.proto
 - Makefile
 - Meetup
 - 21-sept-2017-Golang-Nantes-gRPC-Micro...
 - architecture.png
 - cover.jpg
 - crazy.jpg
 - gohero.proto
 - gopher.jpg
 - grpc_cloud.png
 - grpc_gateway.png
 - grpc.png
 - landing.svg
 - microservices.png
 - slides.slide
 - use_grpc.png
 - vscode.gif
 - .gitignore
 - README.md
- Top Bar:** Shows tabs for "slides.slide", "main.go", and "gohero.go".
- Code Editor:** Displays the content of the "gohero.go" file. The code implements a gRPC service for managing heroes and their powers.

```
1 package app
2
3 import (
4     context "golang.org/x/net/context"
5
6     "github.com/chemidy/gohero-code-examples/gRPC-Service-Discovery-With-Server-Reflection-in-go/proto"
7 )
8
9 var heroes = []*superpower.SuperPower{
10     &superpower.SuperPower{Name: "Flash power", Cat: superpower.SuperPowerCategory_SciencePowers},
11     &superpower.SuperPower{Name: "Superman power", Cat: superpower.SuperPowerCategory_CosmicBasedPowers},
12     &superpower.SuperPower{Name: "Spiderman power", Cat: superpower.SuperPowerCategory_SupernaturalPowers},
13 }
14
15 // GoHeroeServer implements GoHeroeServer.
16 type GoHeroeServer struct {
17 }
18
19 // NewGoHeroeServer returns a new GoHeroeServer.
20 func NewGoHeroeServer() (*GoHeroeServer, error) {
21     return &GoHeroeServer{}, nil
22 }
23
24 // List super powers with a filter
25 func (server GoHeroeServer) List(ctx context.Context, filter *superpower.Filter) (*superpower.SuperPowers, error) {
26
27     if filter.Category == 0 {
28         return &superpower.SuperPowers{SuperPow: heroes}, nil
29     }
30     return &superpower.SuperPowers{SuperPow: filterHeores(heroes, func(hero *superpower.SuperPower) bool { return hero.Cat == filter.Category })}, nil
31 }
32
33 // Add super power
34 func (server GoHeroeServer) Add(ctx context.Context, heroie *superpower.SuperPower) (*superpower.SuperPowers, error) {
35     heroes = append(heroes, heroie)
36     return &superpower.SuperPowers{SuperPow: heroes}, nil
37 }
38
39 type filterOnSuperPower func(*superpower.SuperPower) bool
40
41 // FilterHeores (slice, predicate func)
42 func filterHeores(in []*superpower.SuperPower, fn filterOnSuperPower) []*superpower.SuperPower {
43     out := make([]*superpower.SuperPower, 0)
44     for _, current := range in {
45         if fn(current) {
46             out = append(out, current)
47         }
48     }
49     return out
50 }
```

[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

```

1 package app
2
3 import (
4     context "golang.org/x/net/context"
5
6     "github.com/chemidy/goheroe-code-examples/gRPC-Service-Discovery-With-Server-Reflection-in-go/proto"
7 )
8
9 var heroes = []*superpower.SuperPower{
10     &superpower.SuperPower{Name: "Flash power", Cat: superpower.SuperPowerCategory_SciencePowers},
11     &superpower.SuperPower{Name: "Superman power", Cat: superpower.SuperPowerCategory_CosmicBasedPowers},
12     &superpower.SuperPower{Name: "Spiderman power", Cat: superpower.SuperPowerCategory_SupernaturalPowers},
13 }
14
15 // GoHeroeServer implements GoHeroeServer.
16 type GoHeroeServer struct {
17 }
18
19 // NewGoHeroeServer returns a new GoHeroeServer.
20 func NewGoHeroeServer() (*GoHeroeServer, error) {
21     return &GoHeroeServer{}, nil
22 }
23
24 // List super powers with a filter
25 func (server GoHeroeServer) List(ctx context.Context, filter *superpower.Filter) (*superpower.SuperPowers, error) {
26
27     if filter.Category == 0 {
28         return &superpower.SuperPowers{SuperPowers: heroes}, nil
29     }
30     return &superpower.SuperPowers{SuperPowers: filterHeores(heroes, func(hero *superpower.SuperPower) bool { return hero.Cat == filter.Category })}, nil
31 }
32
33 // Add super power
34 func (server GoHeroeServer) Add(ctx context.Context, hero *superpower.SuperPower) (*superpower.SuperPowers, error) {
35     heroes = append(heroes, hero)
36     return &superpower.SuperPowers{SuperPowers: heroes}, nil
37 }
38
39 type filterOnSuperPower func(*superpower.SuperPower) bool
40
41 // FilterHeores (slice, predicate func)
42 func filterHeores(in []*superpower.SuperPower, fn filterOnSuperPower) []*superpower.SuperPower {
43     out := make([]*superpower.SuperPower, 0)
44     for _, current := range in {
45         if fn(current) {
46             out = append(out, current)
47         }
48     }
49     return out
50 }
51

```

- 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) (<http://LivingPackets.com>)

[@chemidy](http://twitter.com/chemidy) (<http://twitter.com/chemidy>)
[+GoHeroe](http://www.goheroe.org) (<http://www.goheroe.org>)

