gRPC at Lyft

gRPC Meetup - SF

Chris Roche
Lyft, Software Engineer - Core Libraries

Howdy!

- Core Libraries @ Lyft
- Previously Core Platform & DevOps @ VSCO

Background

Infrastructure @ Lyft

- PHP Monolith (ongoing decomposition)
- Go "Tier-Zero" core services
- Python (Micro)services
- Envoy network fabric

What problems are we trying to solve?

- More services = more communication
- Many errors in our Python services are type related
- Documenting APIs are hard to maintain
- No single source of truth for the shape of our data

Solution: gRPC and Protocol Buffers!

- Standardize the API definitions and I/O
- Enforce types at the service boundaries
- IDLs become our single source of truth

Tier-Zero Core Services

Tier-Zero Core Services

- Primary apps of the business
- Go: type-safety & performance
- proto3-based ODM for MongoDB & DynamoDB
- gRPC interface
- Interceptors used to augment RPC endpoints

gRPC Unary Interceptor

```go
func RequestID(
    ctx context.Context,
    req interface{},
    info *grpc.UnaryServerInfo,
    handler grpc.UnaryHandler) (interface{}, error) {

    if meta, ok := metadata.FromContext(ctx); ok {
        if hdrs, ok := meta[RequestIDHeader]; ok && len(hdrs) > 0 {
            ctx = context.WithValue(ctx, RequestIDHeader, hdrs[0])
        }
    }

    return handler(ctx, req)
}
```

- Logging, metrics, request-scoped info
- RPC/Service independent
- Only one per server, though, so...

Chaining Interceptors

```
func Chain(wrappers ...grpc.UnaryServerInterceptor) grpc.UnaryServerInterceptor {
    return func(
        ctx context.Context,
        req interface{},
        info *grpc.UnaryServerInfo,
        handler grpc.UnaryHandler) (interface{}, error) {

        for i := len(wrappers) - 1; i >= 0; i-- {
            handler = wrapHandler(wrappers[i], info, handler)
        }

        return handler(ctx, req)
    }
}

func wrapHandler(
    wrapper grpc.UnaryServerInterceptor,
    info *grpc.UnaryServerInfo,
    handler grpc.UnaryHandler) grpc.UnaryHandler {

    return func(ctx context.Context, req interface{}) (interface{}, error) {
        return wrapper(ctx, req, info, handler)
    }
}
```

Chaining Interceptors

```
grpc.NewServer(
    grpc.UnaryInterceptor(
        Chain(
            RequestID,
            Metrics,
            // etc...,
        ),
    ),
    // other server options...
)
```

- Provided with the Lyft "StdLib"
- Core Libraries solves this so that every service doesn't

Future: protoc-gen-go plugin

- Extra type-safety (Request/Response messages)
- Service/RPC level customizability
- Fork required, though 😕

Python Services

Python Services

- Fronted by Gunicorn + Gevent
- Flask-based HTTP servers
- gRPC clients of tier-zero services
- Want to also be gRPC servers
- Just one issue...

Gevent + gRPC =

Envoy

Envoy

- L7 edge & service proxy
- Modern C++11 codebase
- HTTP/2 & **gRPC fluent**
- Extensible (L3/4/7) filter architecture

Goal

"The network should be transparent to applications. When network and application problems do occur it should be easy to determine the source of the problem."

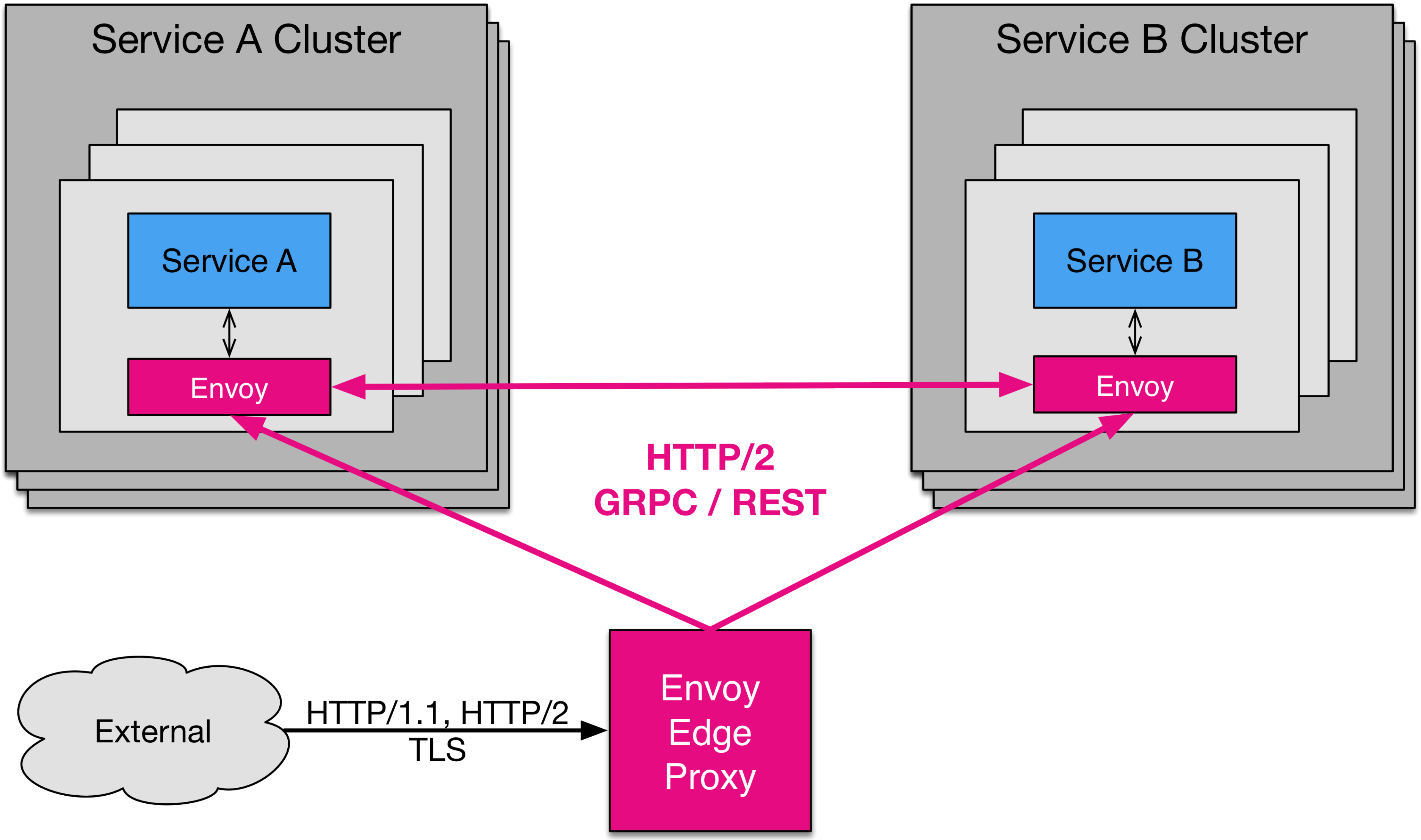
Design

- Out of process architecture
- Transparent to applications
- Hot restart

Features

- Service discovery
- Load balancing
- Health checks
- Mesh routing
- Protocol agnostic
- Robust stats
- Oh, ...and **Open Source!**

Topology


Service A Cluster
Service B Cluster
Service A
Service B
Envoy
Envoy
HTTP/2
GRPC / REST
External
HTTP/1.1, HTTP/2
TLS
Envoy
Edge
Proxy

So what about gRPC?

Code Generation via protoc

- proto2 extensions
- Custom message and field options
- Generated Python client/server (Flask)
- protoc plugin leveraging protoc-gen-go utilities

Proto Extensions + Options

```
service HelloWorld {
  option (http_server_options).isHttpServer = true;

  rpc GetHttpHello (SayHelloRequest) returns (SayHelloResponse) {
    option (http_options).path = "/api/gethello";
    option (http_options).method = "get";
    option (http_options).impl = "test_http.handle_hello_world_get";
  }
}
```

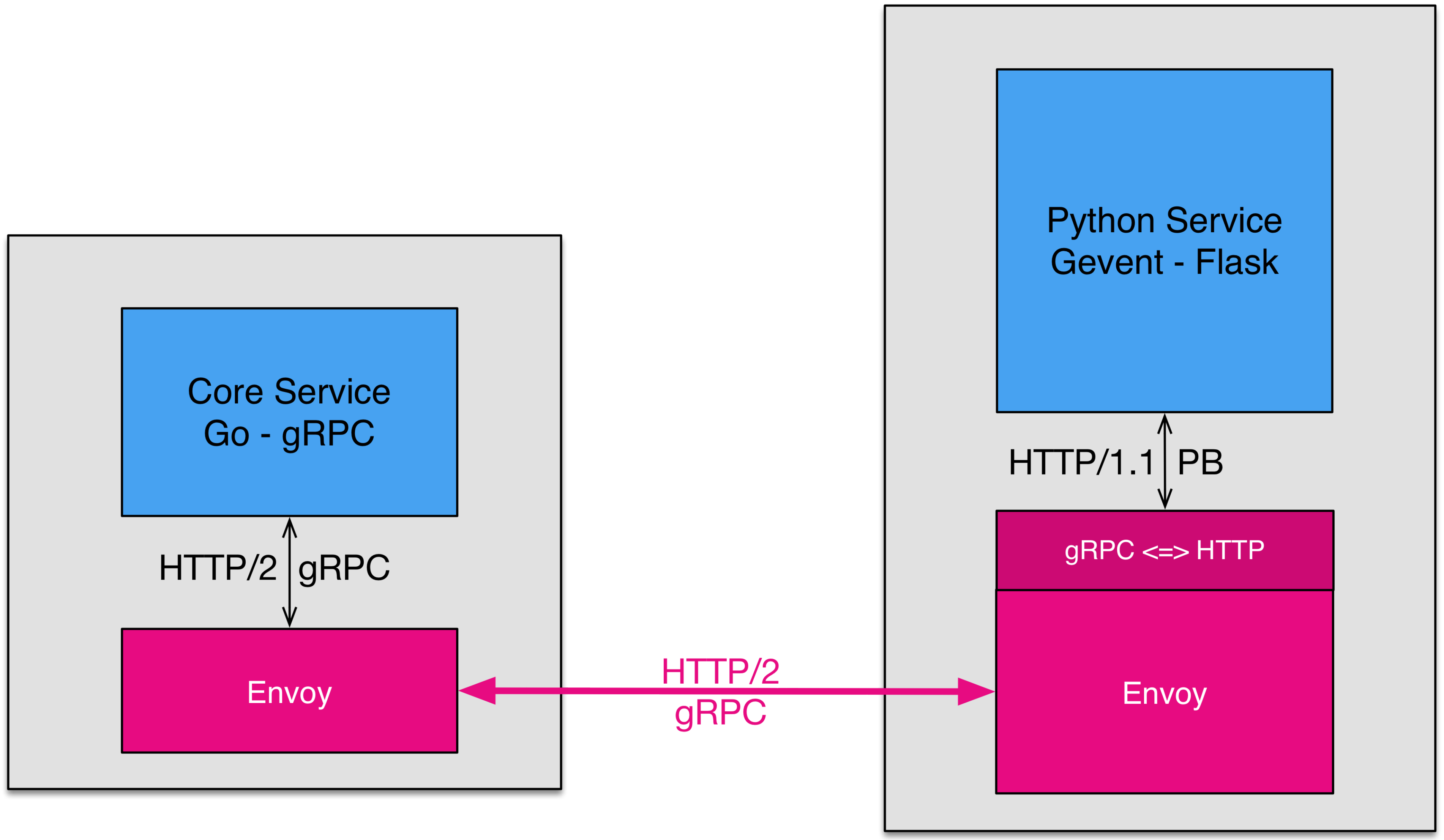
Generated Server

```
@blueprint.route('/api/gethello', methods=['GET'])
def get_hello():
    if request.headers.get('Content-Type') == 'application/proto':
        try:
            input = SayHelloRequest()
            input.ParseFromString(request.data)

            # Call the actual implementation method
            resp = handle_hello_world_get(input)
            return resp.SerializeToString()
        except Exception as e:
            logger.warning(
                'Exception calling handle_hello_world_get on get_hello: {}'.format(repr(e))
            )
            raise e
    else:
        # Non proto application code goes here
        return handle_hello_world_get(request)
```

Generated Client

```
def get_hello(self, input):
    try:
        assert isinstance(input, SayHelloRequest)
        headers = {
            'Content-Type': 'application/proto'
        }
        response = self.get(
            '/api/gethello',
            data=input.SerializeToString(),
            headers=headers,
            raw_request=True,
            raw_response=True)
        op = SayHelloResponse()
        op.ParseFromString(response.content)

        return op
    except Exception as e:
        logger.warning(
            'Exception calling get_hello : {}'.format(repr(e))
        )
        raise e
```

Envoy filter to upgrade/downgrade gRPC



gRPC + Envoy + gEvent =

Organization & Process

Difficulties

protoc and plugins:

- Gnarly installation and API
- Differences in behavior between language plugins

Previously...

- Each project rolled their own build
- Divergent versions of protoc and plugins

Dockerized protoc + plugins

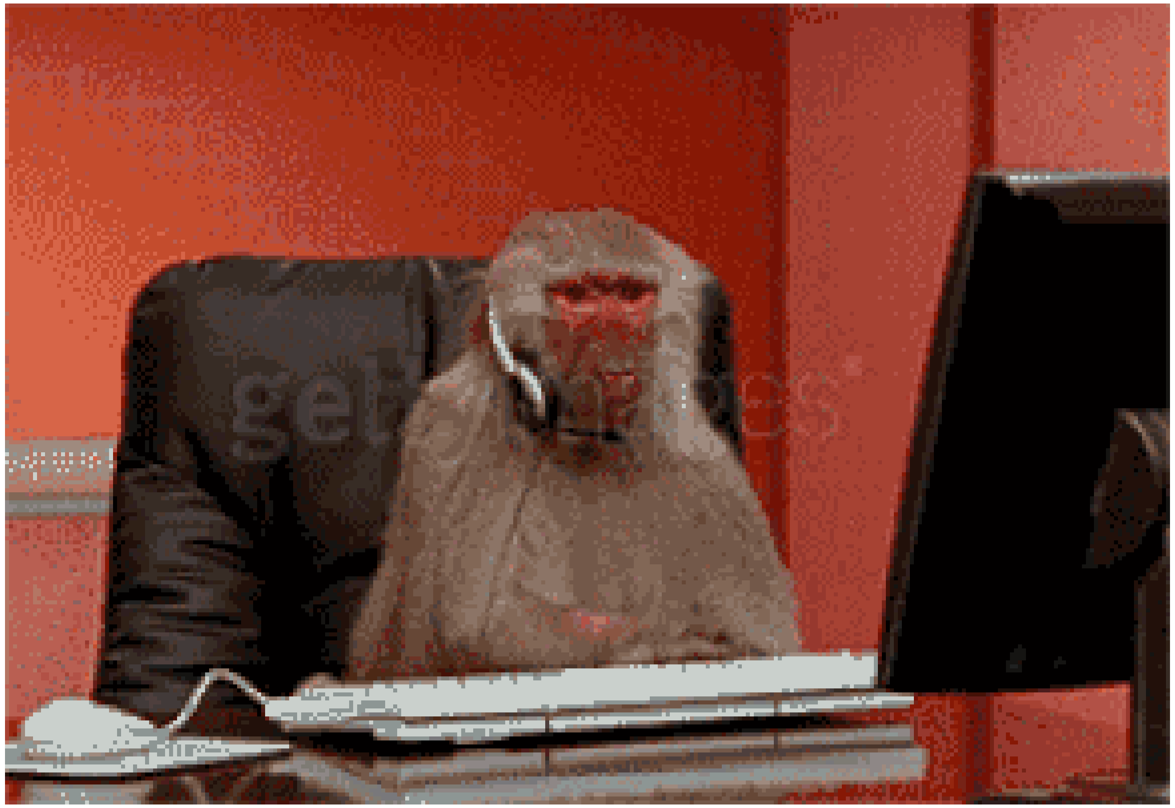
- Pre-baked image with **protoc** and **protoc-gen-***
- Build scripts for each language
- Volume in **./proto/** and **./generated/**
- Use a packagecloud 3.0 deb package

```
docker pull lyft_idl     # pull the builder image
git checkout -b updates  # create a new branch
vim proto/hello.proto    # modify IDL
make build               # builds generated code
git add * && git commit  # Commit IDL + generated
```

Lyft IDL Repository

- Single repository for all message and service IDLs
- All languages generated together
- Generated code committed (it's okay, we promise)
- Package manager based versioning (pip / glide)
- CI verifies builds, linting, generated tests

Live Coding: gRPC KV Store + Envoy + Flask Python Client



Links

- Envoy: lyft.github.io/envoy (https://lyft.github.io/envoy)
- Example Dockerized protoc: github.com/twoism/grpc-gen (https://github.com/twoism/grpc-gen)
- packagecloud proto 3.0 deb: packagecloud.io/capotej/protobuf3 (https://packagecloud.io/capotej/protobuf3)
- This Talk: talks.rodaine.com/grpc-lyft (http://talks.rodaine.com/grpc-lyft)
- Office monkeys appear c/o Getty Images & Hart Productions

Thank you

Chris Roche
Lyft, Software Engineer - Core Libraries
croche@lyft.com (mailto:croche@lyft.com)

http://rodaine.com (http://rodaine.com)
http://github.com/rodaine (http://github.com/rodaine)
@rodaine (http://twitter.com/rodaine)