KubeCon | CloudNativeCon

Europe 2018

# A Brokerless REST Interface for gRPC services

Roman Zimine

Turbonomic

# Agenda


- Who are we?
- Monolith to µServices
- Communication 1.0
- Problems with 1.0
- gRPC to the rescue!
- REST -> gRPC Challenges
- Our Solution
- Demo
- How It Works

# Spoiler Alert

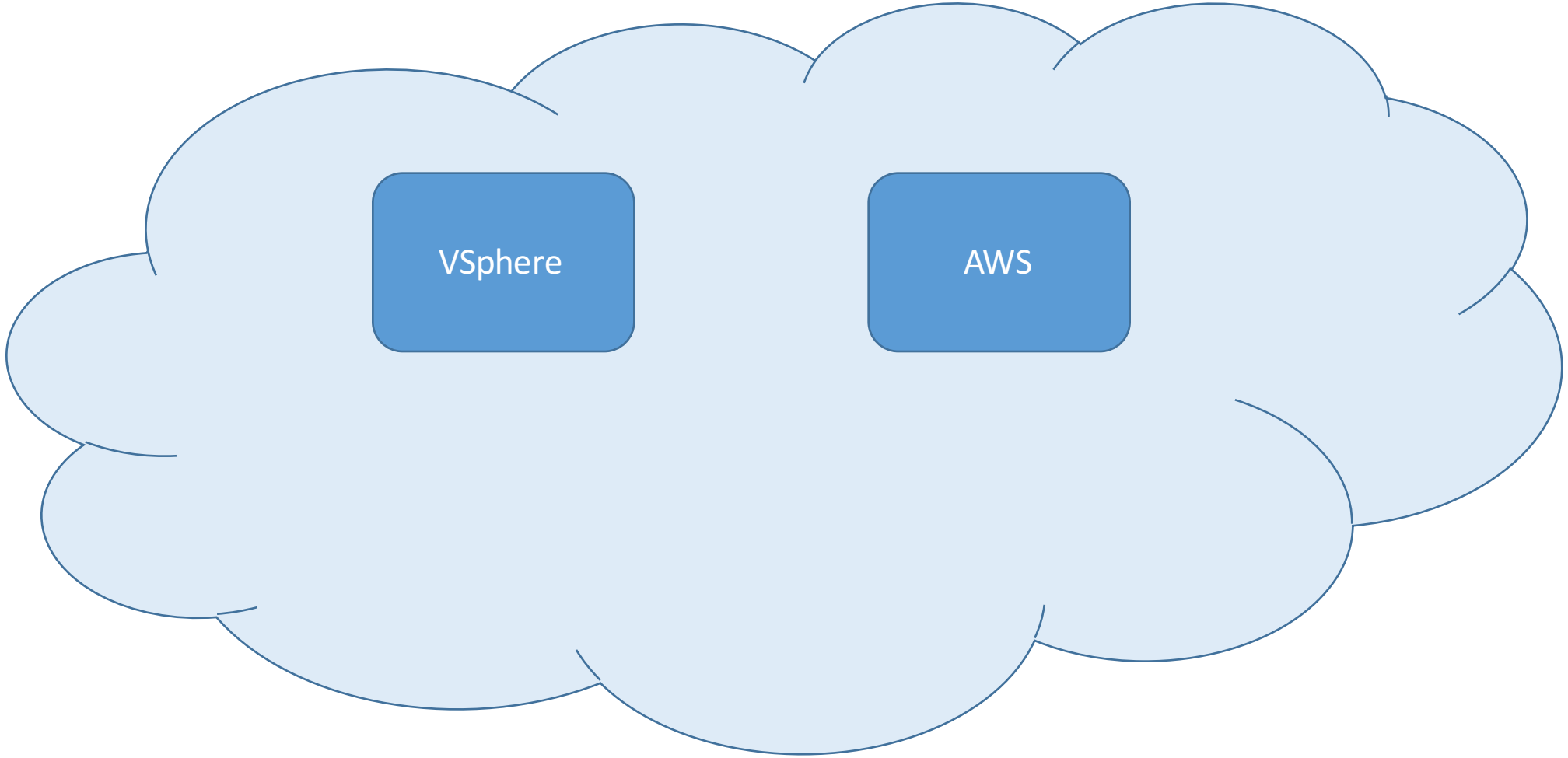

- We wrote a protobuf compiler plugin (framework) to generate Spring Web REST controllers that we inject into the application.

# Who Are We?


- **Workload automation for hybrid cloud** assures performance, while minimizing cost and maintaining compliance
- **Software drives continuous state of health** by matching workload demand to infrastructure supply
- **Technology agnostic:** Container Platforms, Virtualization, Cloud, etc.
- Launched in 2010

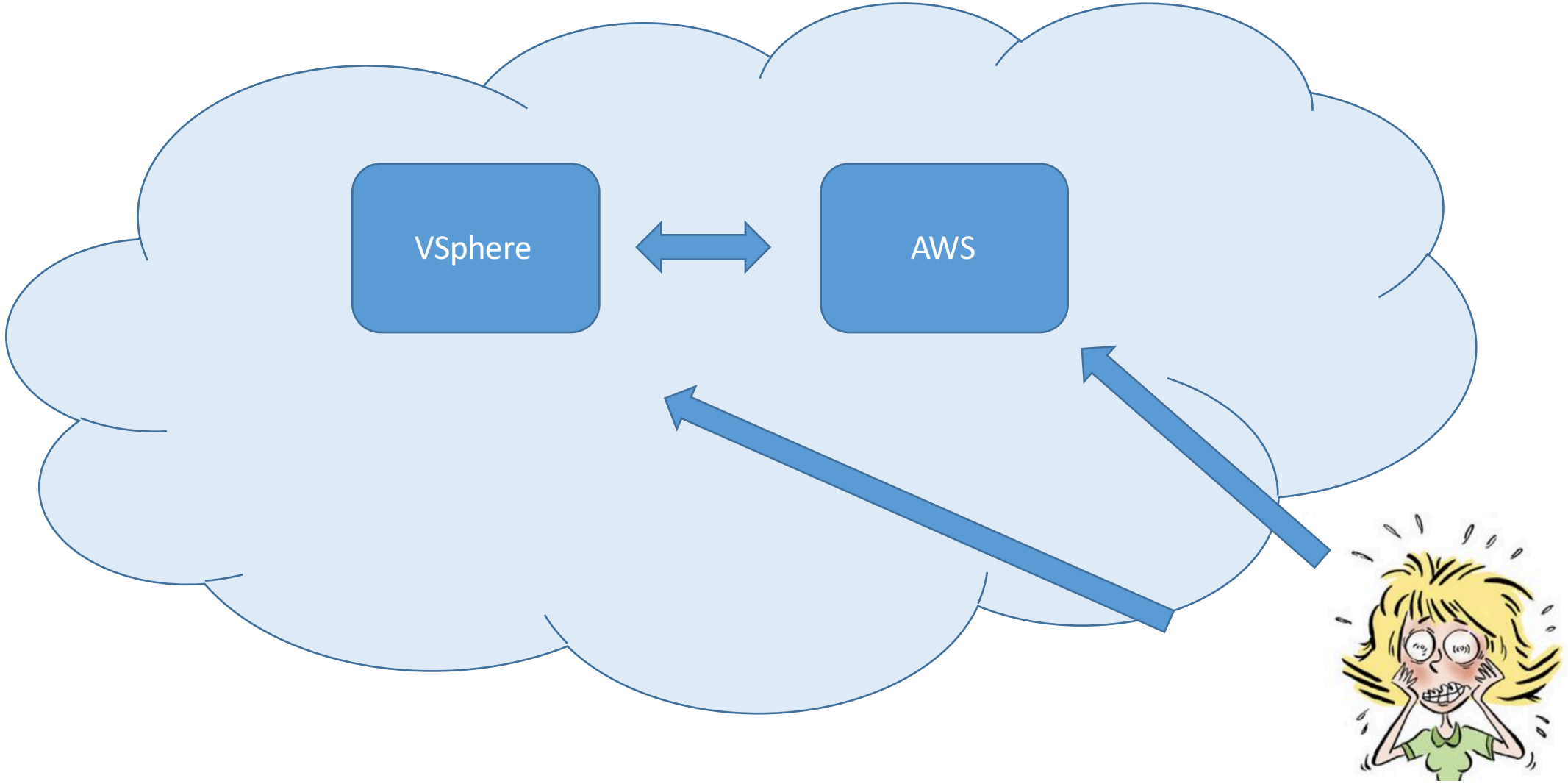
# Who Are We?





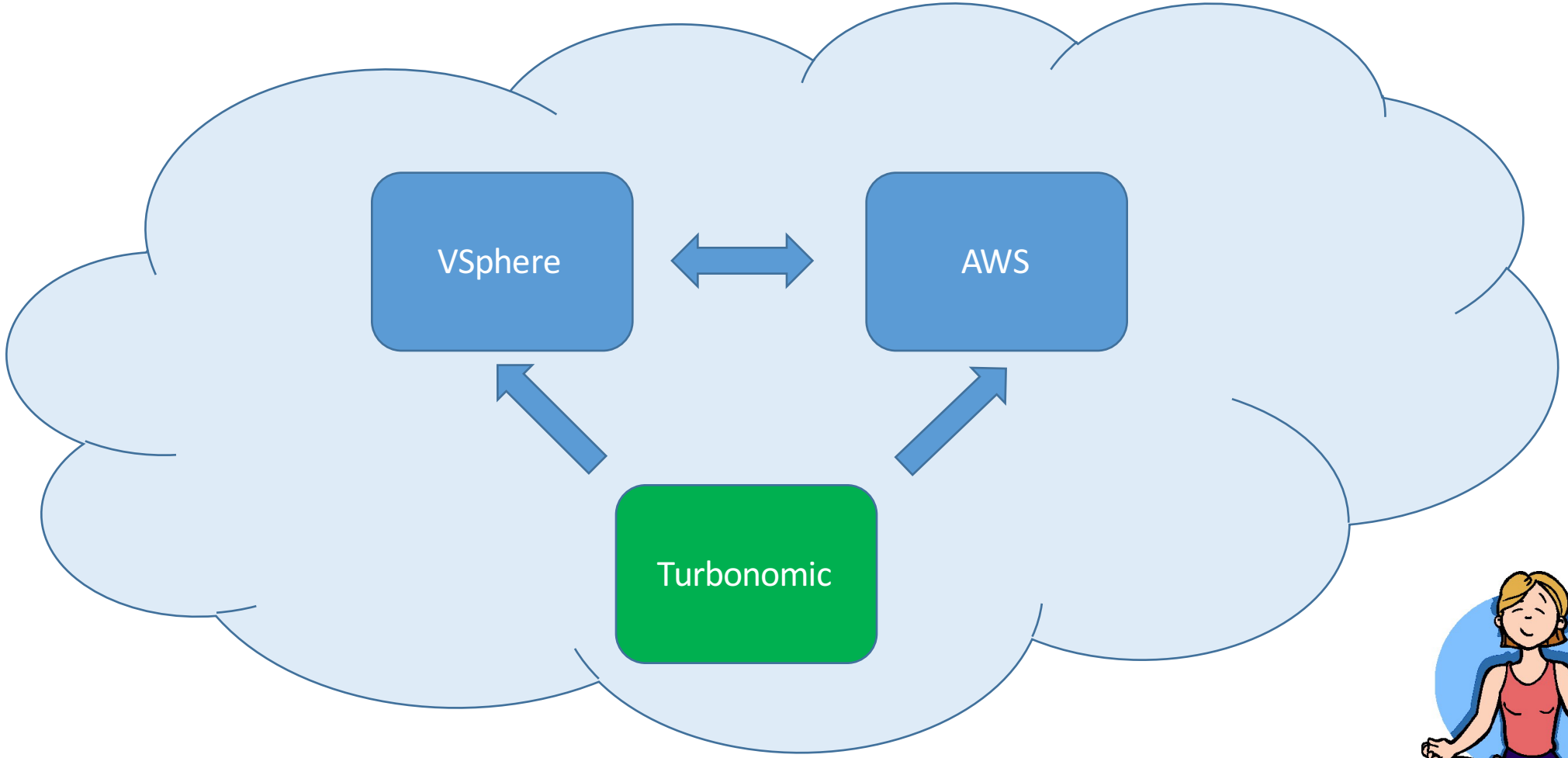
# Who Are We?





# Who Are We?






# Monolith to µServices


- Bigger customers have bigger environments.
- Q: How do we go from managing 10,000 entities to 1,000,000?
- A: Independently scalable containerized components!

# Monolith to µServices

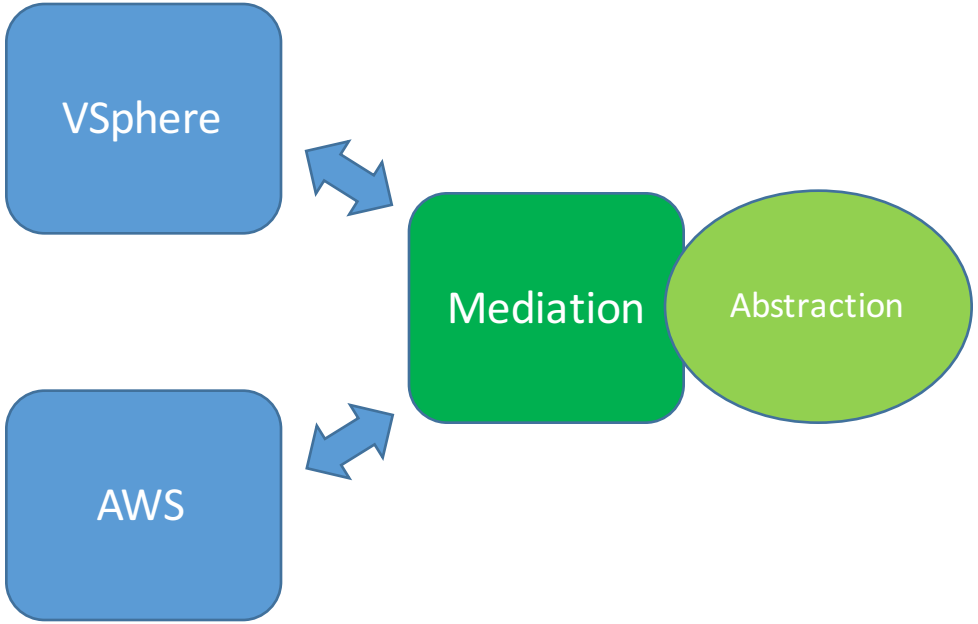


VSphere

AWS

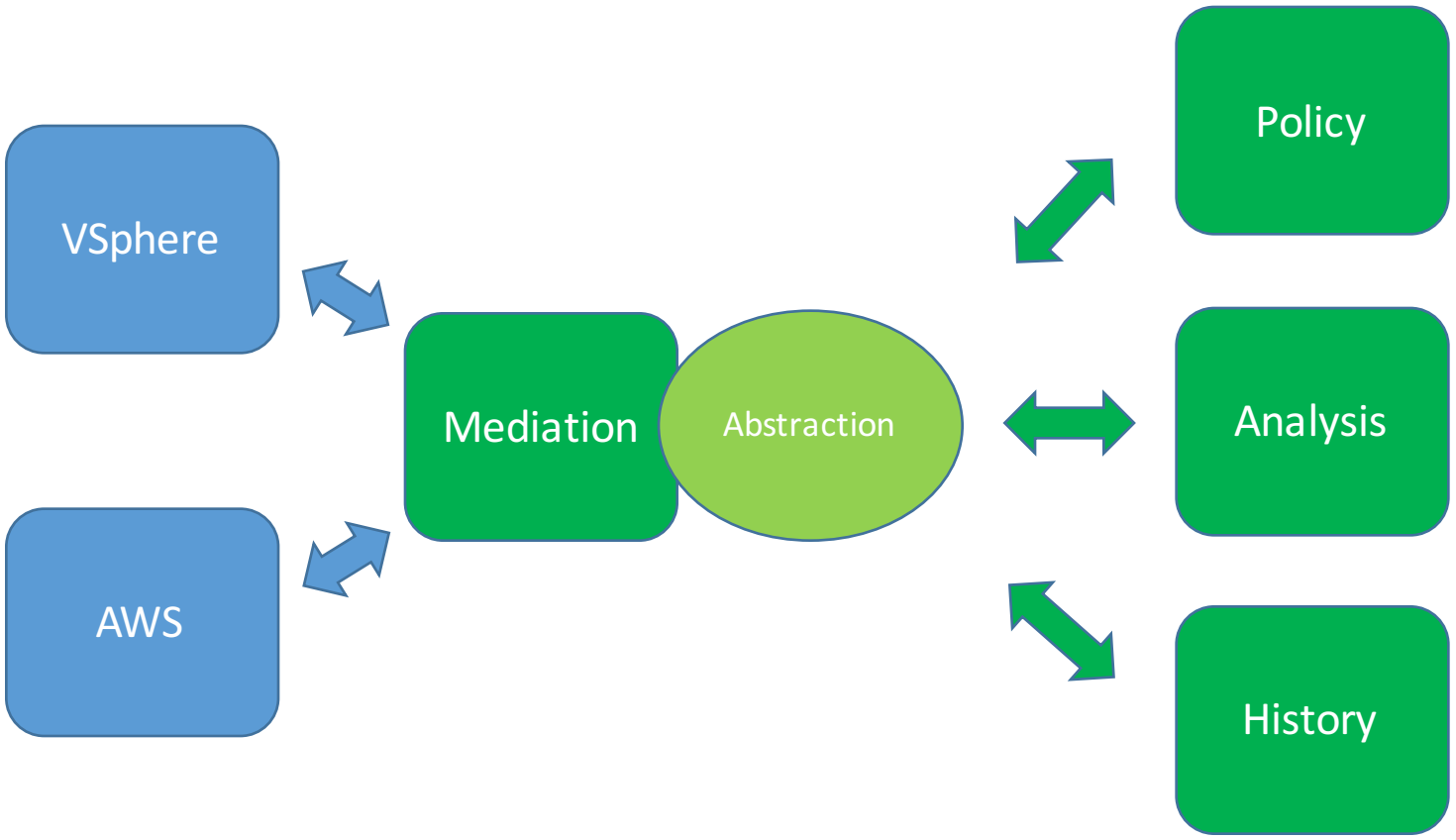
# Monolith to µServices





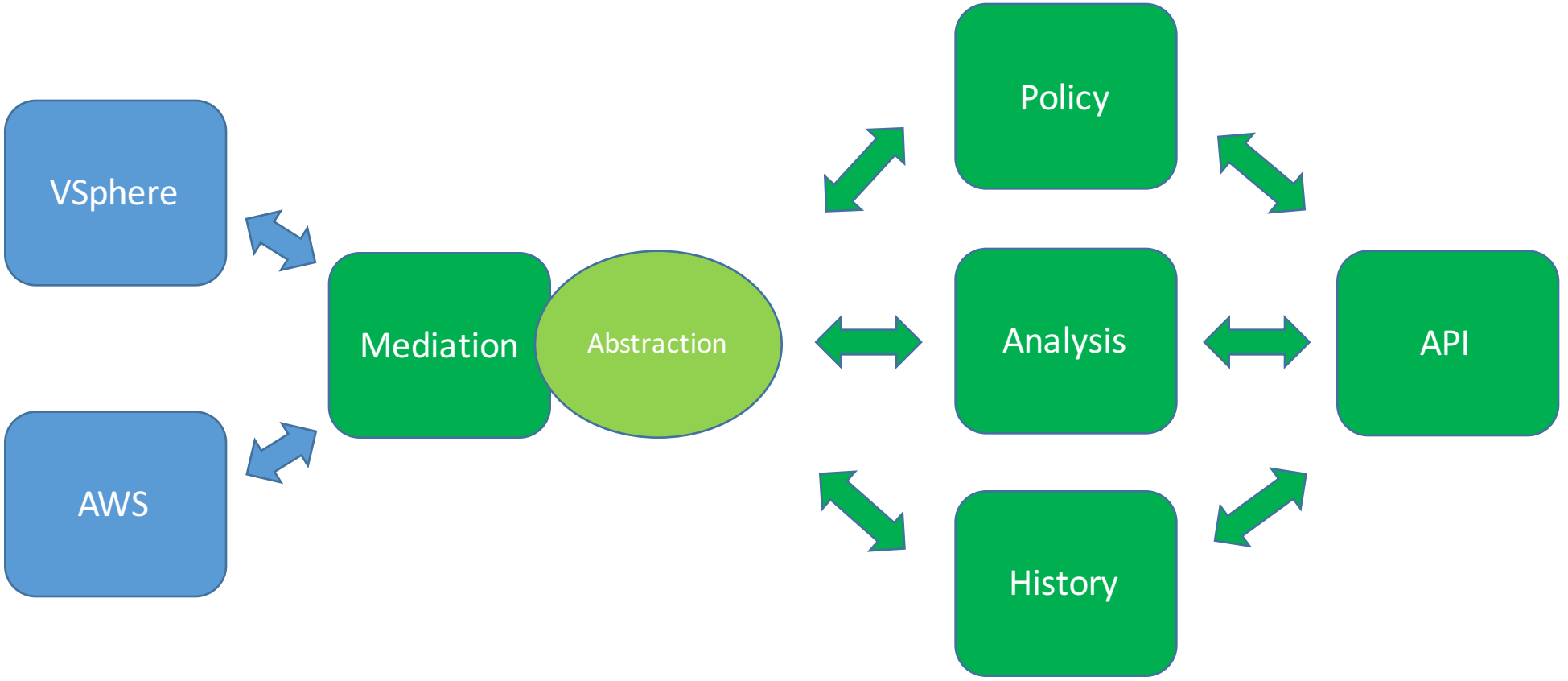
# Monolith to µServices





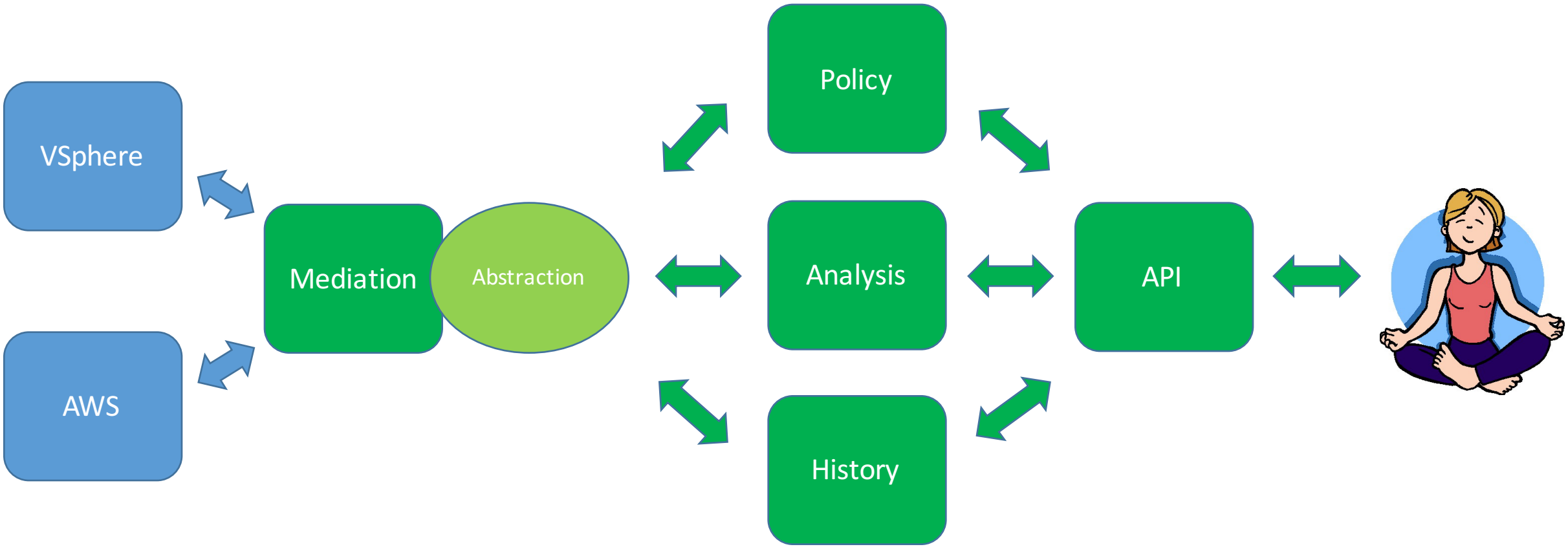
# Monolith to µServices





# Monolith to µServices





# Communication 1.0


- Pub-sub with protobuf.
- Synchronous calls via Java interfaces hiding Spring REST.

# Communication 1.0



- Pub-sub with protobuf.
- Synchronous calls via Java interfaces hiding Spring REST.

```java
public interface EchoComponent {

    @Nonnull
    String echo(@Nonnull final String echoString);

}
```

Java interface used by the clients.

# Communication 1.0



- Pub-sub with protobuf.
- Synchronous calls via Java interfaces hiding Spring REST.

```java
public interface EchoComponent {

    @Nonnull
    String echo(@Nonnull final String echoString);

}
```

Java interface used by the clients.

```java
@RestController
public class EchoController {

    @RequestMapping(method = RequestMethod.POST)
    @ApiOperation(value = "Echo a string back to the caller.")
    public ResponseEntity<String> doEcho(
            @ApiParam(value = "The string to echo.")
            @RequestBody final String echoStr) {
        return new ResponseEntity<>(echoStr, HttpStatus.OK);
    }
}
```

REST controller in the component.

# Problems with 1.0



- Developer productivity – Lots of boilerplate code!
  - Creating and documenting Java objects to match protobuf objects used in pub-sub.
  - Manually implementing an RPC-like layer using Java interfaces.
- Performance (speed - make calls faster)

# gRPC to the rescue!


Service definition

```
message EchoRequest {
    optional string echo = 1;
}

message EchoResponse {
    optional string echo = 1;
}

service EchoService {
    // The Echo call makes a single request and
    // gets a single response.
    rpc Echo(EchoRequest) returns (EchoResponse)
}
```

# gRPC to the rescue!


Service definition

```
message EchoRequest {
    optional string echo = 1;
}

message EchoResponse {
    optional string echo = 1;
}

service EchoService {
    // The Echo call makes a single request and
    // gets a single response.
    rpc Echo(EchoRequest) returns (EchoResponse)
}
```

Service implementation (server)

```
public class EchoRpcService extends EchoServiceImplBase {

    public void echo(EchoRequest echoRequest,
                     StreamObserver<EchoResponse> responseObserver) {
        responseObserver.onNext(EchoResponse.newBuilder()
            .setEcho(echoRequest.getEcho())
            .build());
        responseObserver.onCompleted();
    }

}
```

# gRPC to the rescue!


Service definition

```
message EchoRequest {
    optional string echo = 1;
}

message EchoResponse {
    optional string echo = 1;
}

service EchoService {
    // The Echo call makes a single request and
    // gets a single response.
    rpc Echo(EchoRequest) returns (EchoResponse)
}
```

Service implementation (server)

```
public class EchoRpcService extends EchoServiceImplBase {

    public void echo(EchoRequest echoRequest,
                     StreamObserver<EchoResponse> responseObserver) {
        responseObserver.onNext(EchoResponse.newBuilder()
            .setEcho(echoRequest.getEcho())
            .build());
        responseObserver.onCompleted();
    }

}
```

Service usage (client)

```
final EchoServiceBlockingStub echoService = EchoServiceGrpc.newBlockingStub(channel);
...
final EchoResponse response = echoService.echo(EchoRequest.newBuilder()
    .setEcho("HELLO")
    .build());
```

# REST -> gRPC Challenges



- Developer Productivity (again!)
  - None of our favourite tools work (cURL, Swagger/OpenAPI).
  - No equivalent tools existed in the gRPC ecosystem.
    - grpc-gateway was the most promising, but requires a proxy.

# Our Solution



- A protobuf compiler plugin to generate Spring Web REST controllers that we inject into the application context.

# Our Solution



- A protobuf compiler plugin (framework) to generate Spring Web REST controllers that we inject into the application context.

Service definition

```
message EchoRequest {
    optional string echo = 1;
}

message EchoResponse {
    optional string echo = 1;
}

service EchoService {
    // The Echo call makes a single request and
    // gets a single response.
    rpc Echo(EchoRequest) returns (EchoResponse)
}
```

# Our Solution



- A protobuf compiler plugin (framework) to generate Spring Web REST controllers that we inject into the application context.

Service definition

```
message EchoRequest {
    optional string echo = 1;
}

message EchoResponse {
    optional string echo = 1;
}

service EchoService {
    // The Echo call makes a single request and
    // gets a single response.
    rpc Echo(EchoRequest) returns (EchoResponse)
}
```

Generated controller (excerpt)

```
@ApiOperation(
  value = "echo",
  notes =
      "The Echo call makes a single request and gets a single response."
)
@RequestMapping(
  path = "echo",
  method = RequestMethod.POST,
  consumes = {MediaType.APPLICATION_JSON_UTF8_VALUE},
  produces = {MediaType.APPLICATION_JSON_UTF8_VALUE}
)
public ResponseEntity<EchoServiceResponse<EchoREST.EchoResponse>>
    echo(@RequestBody EchoREST.EchoRequest input) {
  if (service == null) {
    return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
        .body(EchoServiceResponse.error(
                "The service  was not injected into the RestController. Check that
  }
  List<EchoREST.EchoResponse> responseList = new ArrayList<>();
```

# Demo

KubeCon

CloudNativeCon

Europe 2018

# How It Works



- Read CodeGeneratorRequest proto from stdin.

# How It Works



- Read CodeGeneratorRequest proto from stdin.

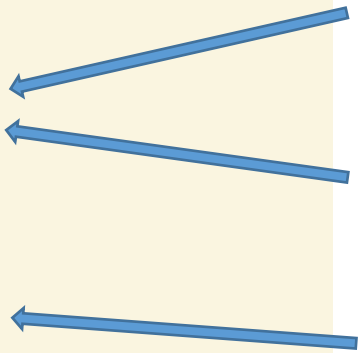
```
// An encoded CodeGeneratorRequest is written to the plugin's stdin.
message CodeGeneratorRequest {
    ...
    repeated FileDescriptorProto proto_file = 15;
    ...
}

message FileDescriptorProto {
    ...

    repeated DescriptorProto message_type = 4;

    ...

    repeated ServiceDescriptorProto service = 6;

    ...
}
```

# How It Works



- Read CodeGeneratorRequest proto from stdin.

```
// An encoded CodeGeneratorRequest is written to the plugin's stdin.
message CodeGeneratorRequest {
    ...
    repeated FileDescriptorProto proto_file = 15;
    ...
}

message FileDescriptorProto {
    ...

    repeated DescriptorProto message_type = 4;

    ...

    repeated ServiceDescriptorProto service = 6;

    ...
}
```



```
message EchoRequest {
    optional string echo = 1;
}

message EchoResponse {
    optional string echo = 1;
}

service EchoService {
    // The Echo call makes a single request and
    // gets a single response.
    rpc Echo(EchoRequest) returns (EchoResponse)
}
```

# How It Works


- Read CodeGeneratorRequest proto from stdin.
- For each file:

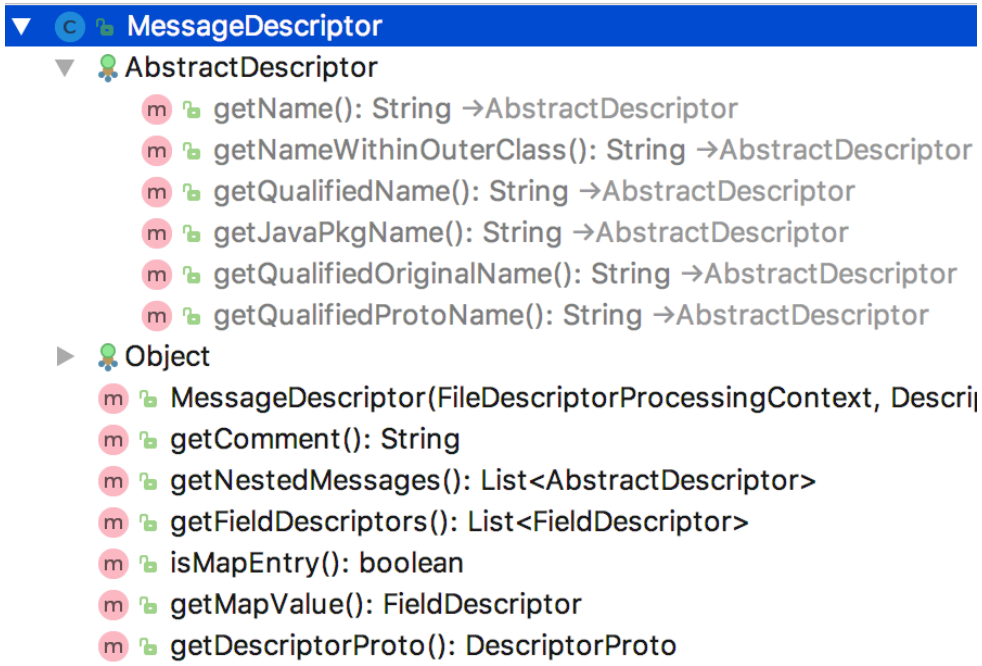
# How It Works


- Read CodeGeneratorRequest proto from stdin.
- For each file:
  - Parse protobuf descriptors into Java objects.

# How It Works



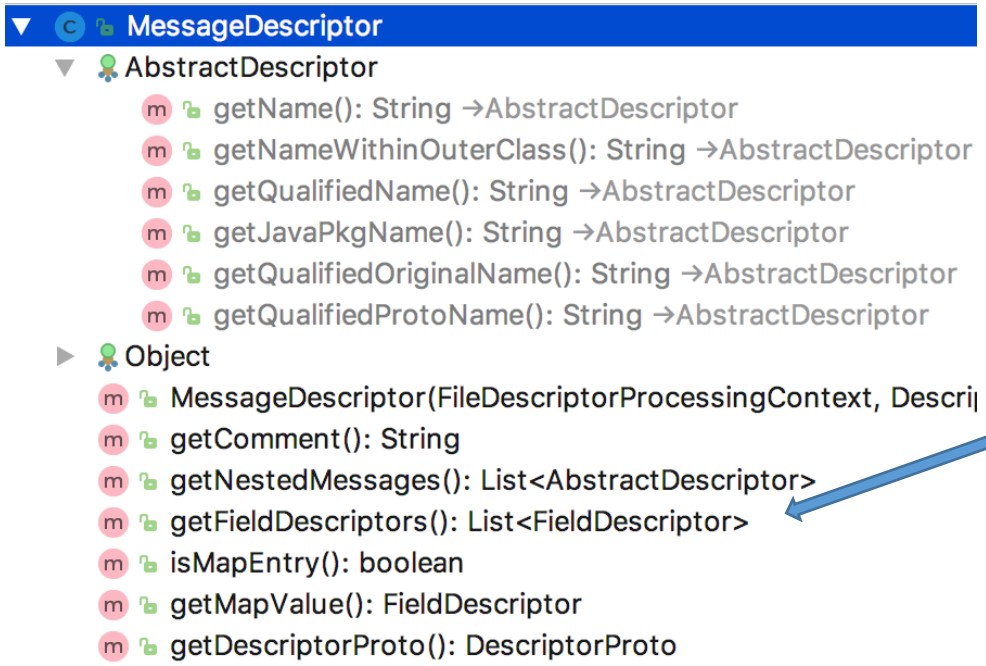
- Read CodeGeneratorRequest proto from stdin.
- For each file:
  - Parse protobuf descriptors into Java objects.



# How It Works



- Read CodeGeneratorRequest proto from stdin.
- For each file:
  - Parse protobuf descriptors into Java objects.




```
message EchoRequest {
    optional string echo = 1;
}
```

# How It Works


- Read CodeGeneratorRequest proto from stdin.
- For each file:
  - Parse protobuf descriptors into Java objects.
  - Make calls to the specific generator to process the descriptors.

# How It Works



- Read CodeGeneratorRequest proto from stdin.
- For each file:
  - Parse protobuf descriptors into Java objects.
  - Make calls to the specific generator to process the descriptors.

```
interface ProtocPluginCodeGenerator {

    String getPluginName();

    String generatePluginJavaClass(@Nonnull final String protoJavaClass);

    String generateImports();

    Optional<String> generateEnumCode(@Nonnull final EnumDescriptor enumDescriptor);

    Optional<String> generateMessageCode(@Nonnull final MessageDescriptor messageDescriptor);

    Optional<String> generateServiceCode(@Nonnull final ServiceDescriptor serviceDescriptor);

    boolean skipFile(@Nonnull final FileDescriptorProto fileDescriptorProto);
}
```

# How It Works



- Read CodeGeneratorRequest proto from stdin.
- For each file:
  - Parse protobuf descriptors into Java objects.
  - Make calls to the specific generator to process the descriptors.
  - Generate code from the descriptors and templates.

```
private static final String SERVICE_METHOD_TEMPLATE =
    "\n@ApiOperation(value=\"<methodName>\",notes=<comments>)" +
    "@RequestMapping(path=\"<methodName>\",method=RequestMethod.POST,"+
    "consumes = {MediaType.APPLICATION_JSON_UTF8_VALUE}," +
    "produces = {MediaType.APPLICATION_JSON_UTF8_VALUE})" +
    "public ResponseEntity\\<<responseBodyType>> <methodName> (@RequestBody <requestBodyType> input) {" +
        "if (service == null) {" +
            "return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)" +
                ".body(<responseWrapper>.error(\"The service <serviceName> was not injected " +
                    "into the RestController. Check that a bean implementing <serviceName>ImplBase" +
                    " exists in the Spring configuration.\"));" +
        "}" +
        "List\\<resultType>> responseList = new ArrayList\\<>();" +
        "LinkedBlockingQueue\\<Status> statusQueue = new LinkedBlockingQueue\\<>(1);" +
        "StreamObserver\\<<resultProto>> responseObserver = new StreamObserver\\<<resultProto>>() {" +
```

# How It Works


- Read CodeGeneratorRequest proto from stdin.
- For each file:
  - Parse protobuf descriptors into Java objects.
  - Make calls to the specific generator to process the descriptors.
  - Generate code from the descriptors and templates.
- Write CodeGeneratorResponse to stdout.

# How It Works


- Read CodeGeneratorRequest proto from stdin.
- For each file:
  - Parse protobuf descriptors into Java objects.
  - Make calls to the specific generator to process the descriptors.
  - Generate code from the descriptors and templates.
- Write CodeGeneratorResponse to stdout.

```
// The plugin writes an encoded CodeGeneratorResponse to stdout.
message CodeGeneratorResponse {
    message File {
        optional string name = 1;
        ...
        // The file contents - code as a string!
        optional string content = 15
    }

    repeated File file = 15;
}
```

# Check It Out!


- Common Plugin Framework (Java):
  - https://github.com/turbonomic/protoc-plugin-common
- Spring REST Plugin:
  - https://github.com/turbonomic/protoc-gen-spring-rest
- Turbonomic - ***Please stop by Booth S C-25***