

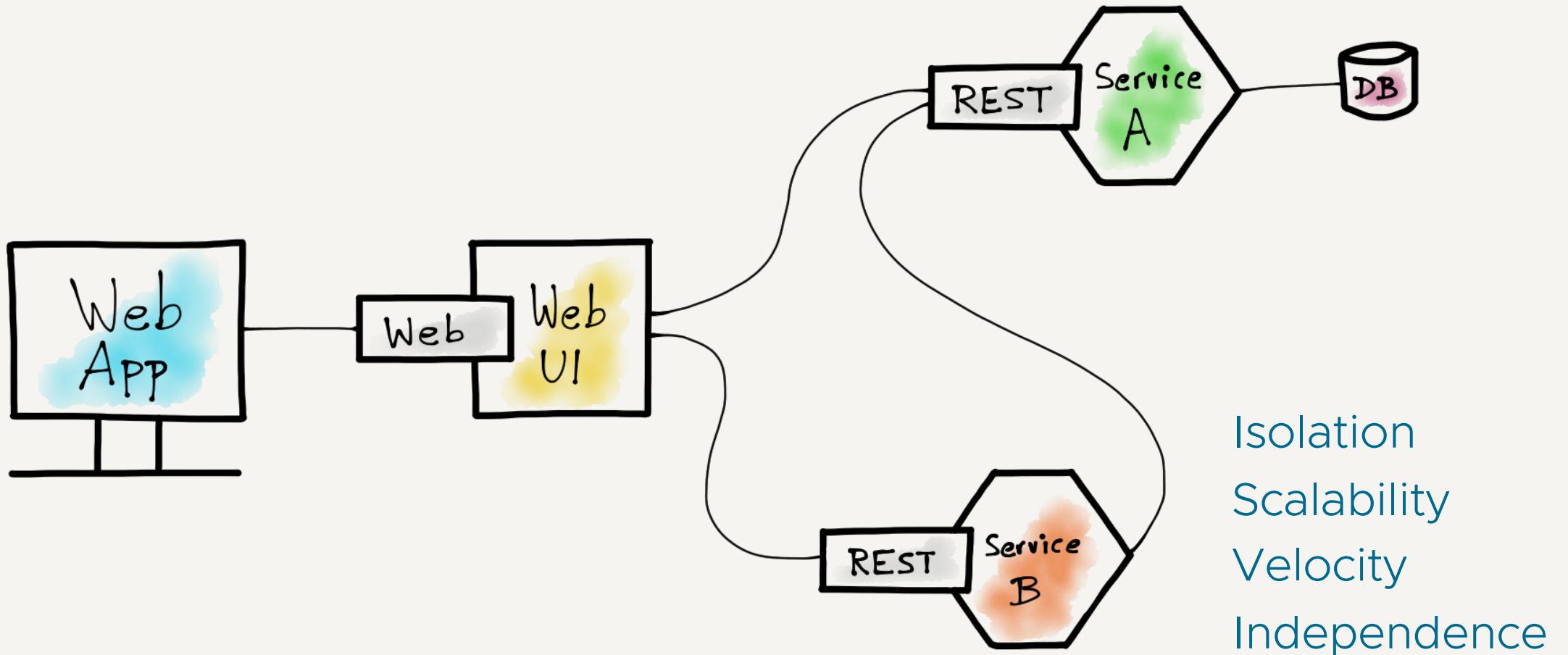
Seamless Cloud-Native Apps with gRPC-Web and Istio

All Things Open 2019

Venil Noronha

Open Source Technology Center, VMware

The REST APIs Approach



Drawbacks of REST APIs

```
1 GET /api/users/123
2
3 {
4     "id": 123,
5     "name": "John Doe",
6     "weight": 180.56
7 }
```

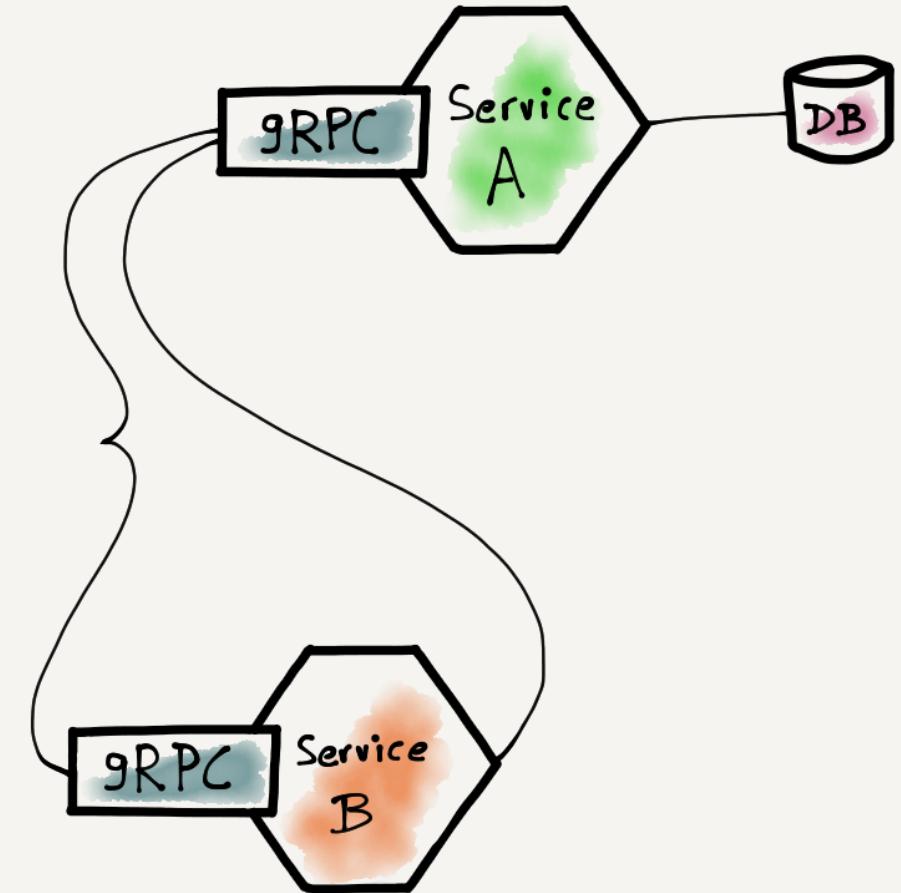
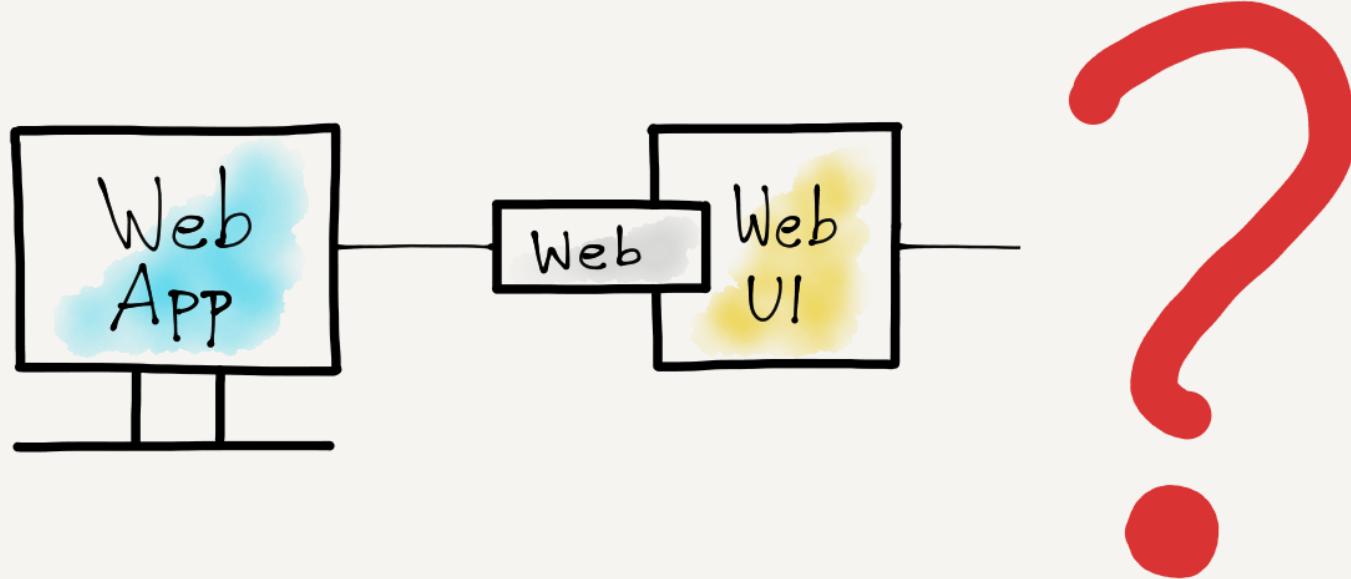
Type Safety
Compatibility
Performance
Contract Definition

Protobufs and gRPC

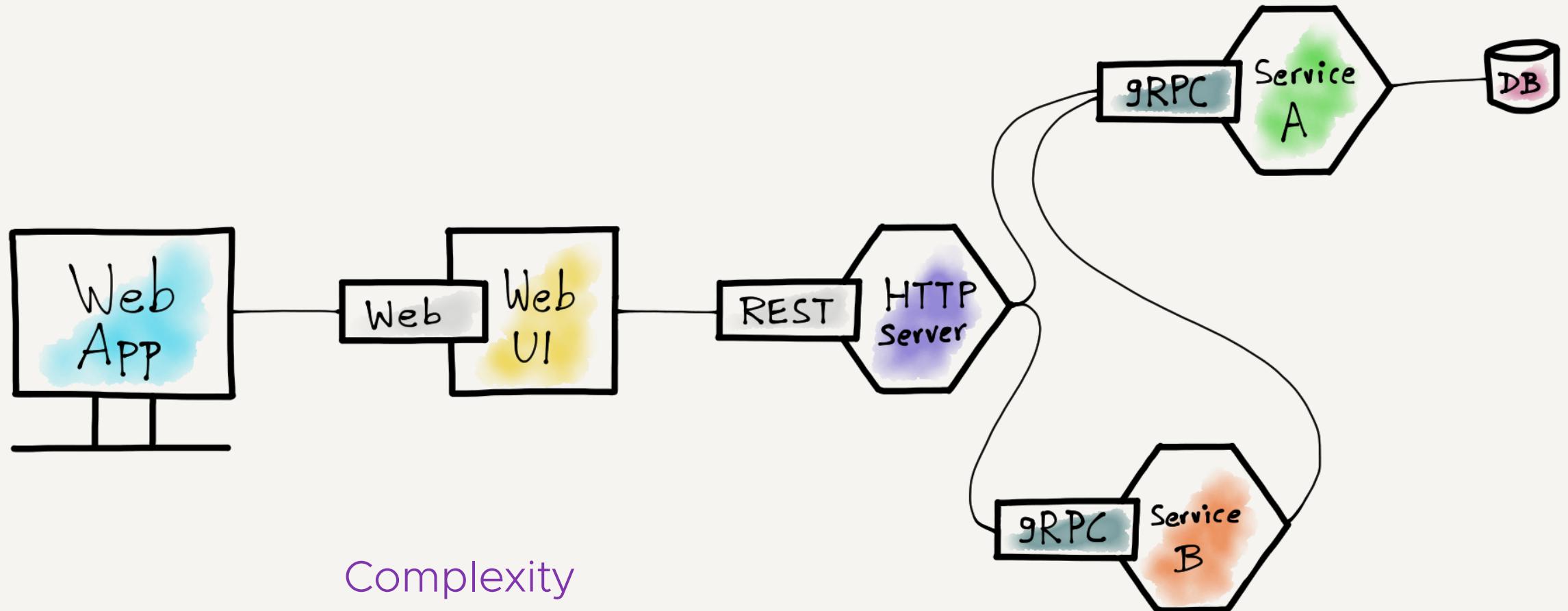
```
1 service UserService {  
2   rpc FindUser (FindUserRequest) returns (FindUserResponse);  
3 }  
4  
5 message FindUserRequest {  
6   uint64 id = 1;  
7 }  
8  
9 message FindUserResponse {  
10  uint64 id = 1;  
11  string name = 2;  
12  double weight = 3;  
13 }
```

Type Safety
Compatibility
Performance
Contract Definition

Challenges with gRPC and the Web

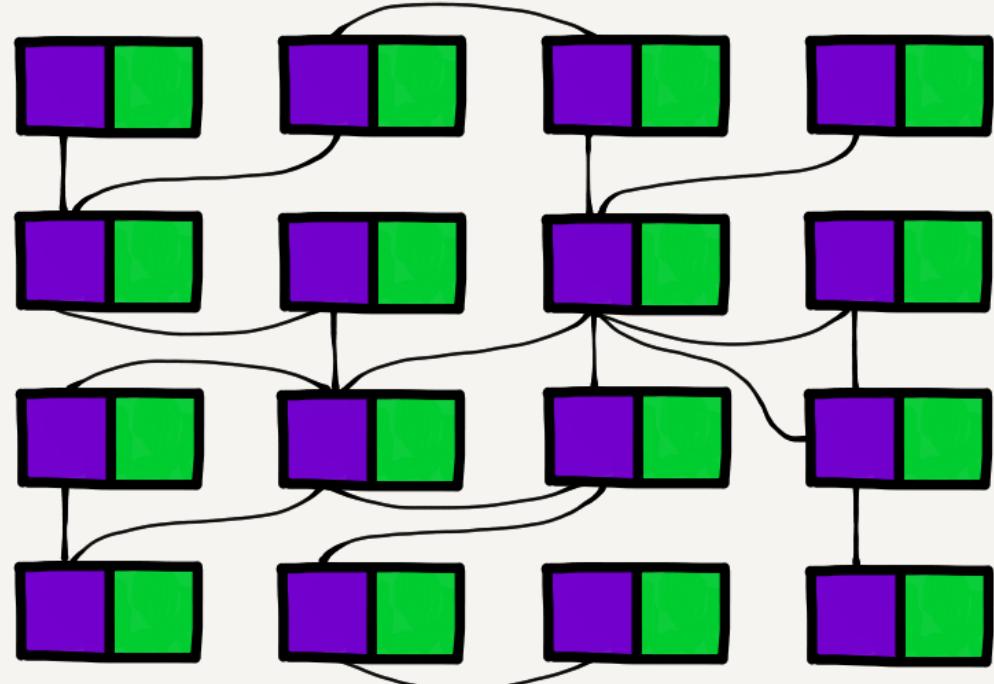


The gRPC + HTTP Server Pattern

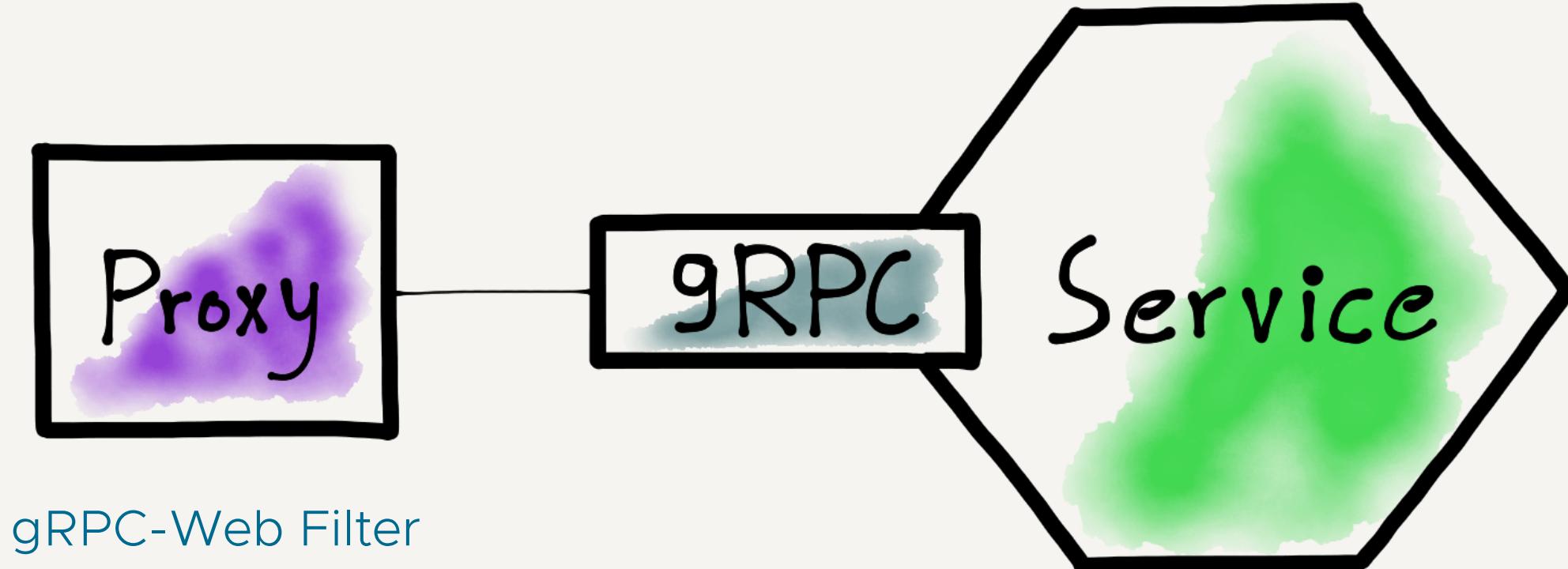


Service Mesh Overview

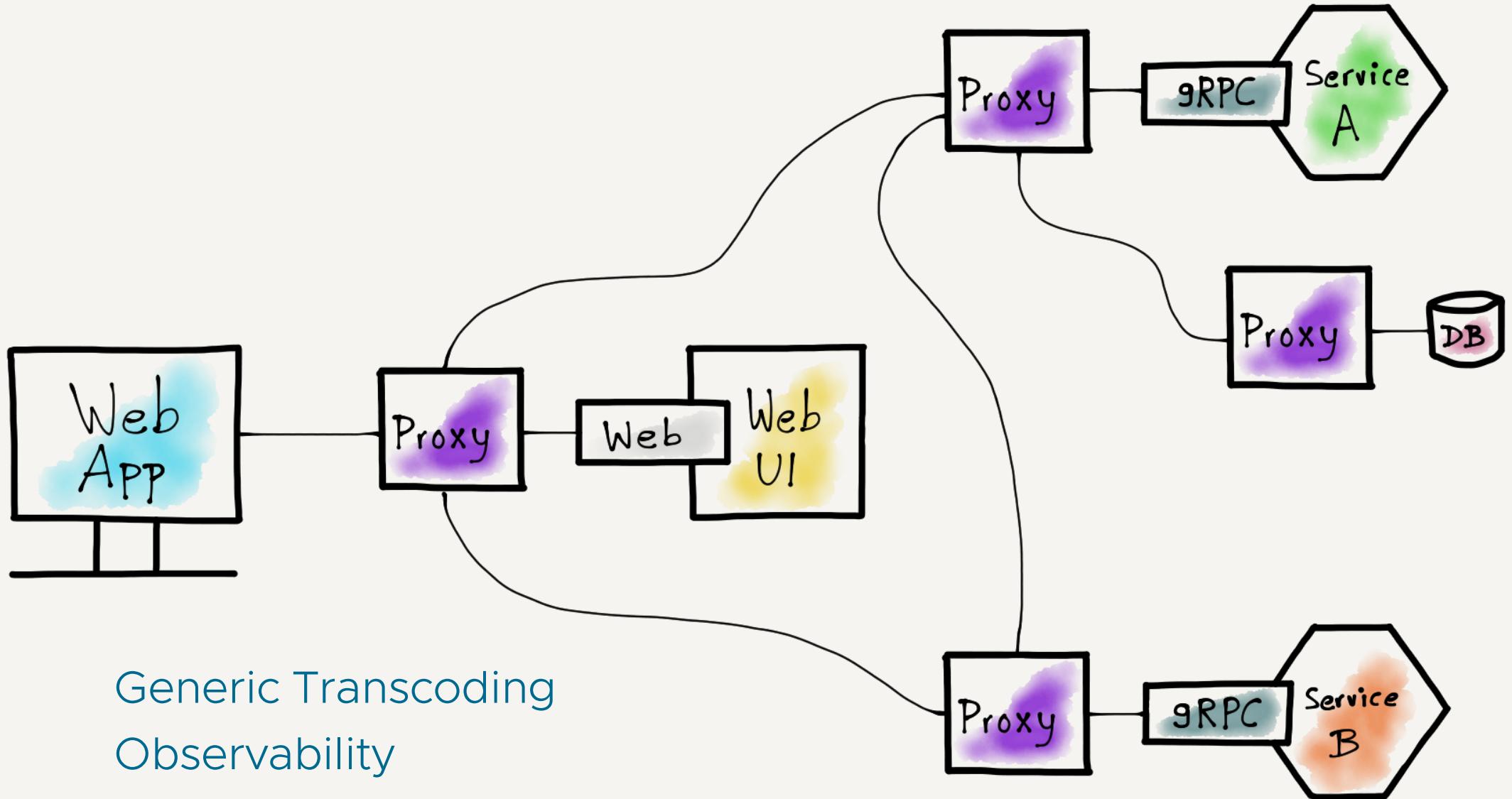
- Dedicated infrastructure layer
- Handles service-to-service communication
- Manages complex topology of services
- Array of lightweight network proxies
- Deployed alongside application code
- Doesn't need the application to be aware



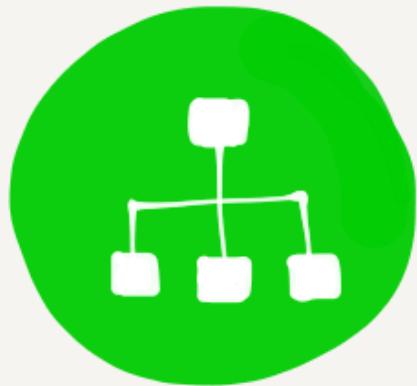
The Proxy Pattern



gRPC-Web and Istio

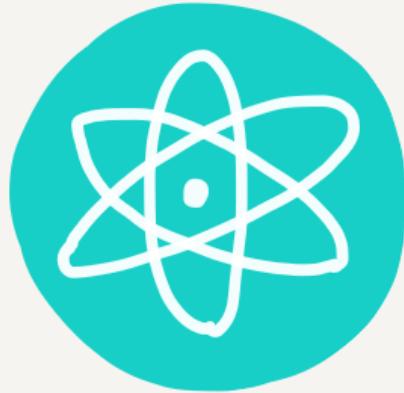


Benefits of a Service Mesh



Discoverable

Services can find each other



Resilient

Built-in robustness frameworks, load balancing & test infrastructure



Configurable

Configure services dynamically at runtime



Observable

Standardized metrics, monitoring & distributed tracing



Secure

Encrypt and protect service communication

gRPC-Web and Istio Demo

An Emoji Web App

<https://github.com/venilnoronha/grpc-web-istio-demo>

The API Definition

```
1 package proto;  
2  
3 service EmojiService {  
4     rpc InsertEmojis (EmojiRequest) returns (EmojiResponse);  
5 }  
6  
7 message EmojiRequest {  
8     string input_text = 1;          I like :pizza: and :sushi!:  
9 }  
10  
11 message EmojiResponse {  
12     string output_text = 1;        I like 🍕 and 🍣!  
13 }
```

Generate Definitions for Go and JavaScript

Parse and generate the Go file

```
1 protoc -I proto/ proto/emoji.proto \
2     --go_out=plugins=grpc:proto
```

Parse and generate the JavaScript files

```
1 protoc -I proto/ proto/emoji.proto \
2     --js_out=import_style=commonjs:proto \
3     --grpc-web_out=import_style=commonjs, \
4                                     mode=grpcwebtext:proto
```

The Generated Go File

```
1 type EmojiServiceServer interface {
2     InsertEmojis(context.Context, *EmojiRequest) (*EmojiResponse, error)
3 }
4
5 type EmojiRequest struct {
6     InputText string `protobuf:"bytes,1,..."`
7 }
8
9 type EmojiResponse struct {
10    OutputText string `protobuf:"bytes,1,..."`
11 }
```

The Generated JavaScript Files

```
1 proto.EmojiRequest.prototype.setInputText = function(value) {  
2   jspb.Message.setProto3StringField(this, 1, value);  
3 };  
4  
5 proto.EmojiResponse.prototype.getOutputText = function() {  
6   return jspb.Message.getFieldWithDefault(this, 1, "");  
7 };  
8  
9 proto.EmojiServiceClient.prototype.insertEmojis =  
10  function(request, metadata, callback) {  
11    return this.client_.rpcCall('/proto.EmojiService/InsertEmojis',  
12      request, metadata, callback);  
13  };
```

The EmojiService Server

```
1 func (s *server) InsertEmojis(ctx context.Context,
2     req *proto.EmojiRequest) (*proto.EmojiResponse, error) {
3     outputText := emoji.Sprint(req.InputText)
4     return &proto.EmojiResponse{OutputText: outputText}, nil
5 }
6
7 func main() {
8     grpcServer := grpc.NewServer()
9     proto.RegisterEmojiServiceServer(grpcServer, &server{})
10
11    listener, err := net.Listen("tcp", ":9000")
12    grpcServer.Serve(listener)
13 }
```

```
$ go run cmd/server.go  
2019/03/09 14:16:18 Listening on [::]:9000
```

█

The EmojiService Client

```
1 var server = flag.String("server", "localhost:9000", "Server address")
2 var text = flag.String("text", "Hello world!", "Input text")
3
4 func main() {
5     conn, err := grpc.Dial(*server, grpc.WithInsecure())
6     client := proto.NewEmojiServiceClient(conn)
7
8     req := &proto.EmojiRequest{InputText: *text}
9     res, err := client.InsertEmojis(context.Background(), req)
10    log.Printf("Server says: %s", res.OutputText)
11 }
```

```
$ go run cmd/client.go \
>     --text 'I like :pizza: and :sushi:!' \
>     --server 'localhost:9000'
2019/03/09 14:18:53 Request: I like :pizza: and :sushi:!
2019/03/09 14:18:53 Server says: I like 🍕 and 🍣 !
$ █
```

The Server Container

```
1 FROM golang:1.12 as builder
2 WORKDIR /root/go/src/.../grpc-web-istio-demo/
3 COPY ./ .
4 RUN CGO_ENABLED=0 GOOS=linux \
5     go build -a -installsuffix cgo -v \
6     -o bin/server ./cmd/server.go
7
8 FROM scratch
9 WORKDIR /bin/
10 COPY --from=builder /root/.../grpc-web-istio-demo/bin/server .
11 ENTRYPOINT [ "/bin/server" ]
12 EXPOSE 9000
```

Build and Push the Server Image

Build the Docker image

```
1 docker build -f server.Dockerfile \
2 | | | | | -t vnoronha/grpc-web-istio-demo:server .
```

Push the Docker image

```
1 docker push vnoronha/grpc-web-istio-demo:server
```

The Web UI HTML

```
1 <!DOCTYPE html>
2 <html>
3   <body>
4     <div id="editor" contentEditable="true"
5       onkeyup="insertEmojis()">
6     </div>
7     <script src="dist/main.js"></script>
8   </body>
9 </html>
```

The Web UI JavaScript

```
1 const {EmojiRequest, EmojiResponse} = require('./emoji_pb.js');
2 const {EmojiServiceClient} = require('./emoji_grpc_web_pb.js');
3
4 var client = new EmojiServiceClient('http://' + window.location.host);
5 var editor = document.getElementById('editor');
6
7 function insertEmojis() {
8     var req = new EmojiRequest();
9     req.setInputText(editor.innerText);
10    client.insertEmojis(req, {}, (err, res) => {
11        editor.innerText = res.getOutputText();
12    });
13}
```

The Web UI Container

```
1 FROM node:8.15 as builder
2 WORKDIR /web-ui/
3 COPY ./ .
4 RUN npm install
5 RUN npx webpack app.js
6
7 FROM python:2.7
8 WORKDIR /web-ui/
9 COPY --from=builder /web-ui/ .
10 ENTRYPOINT [ "python" ]
11 CMD [ "-m", "SimpleHTTPServer", "9001" ]
12 EXPOSE 9001
```

Build and Push the Web UI Image

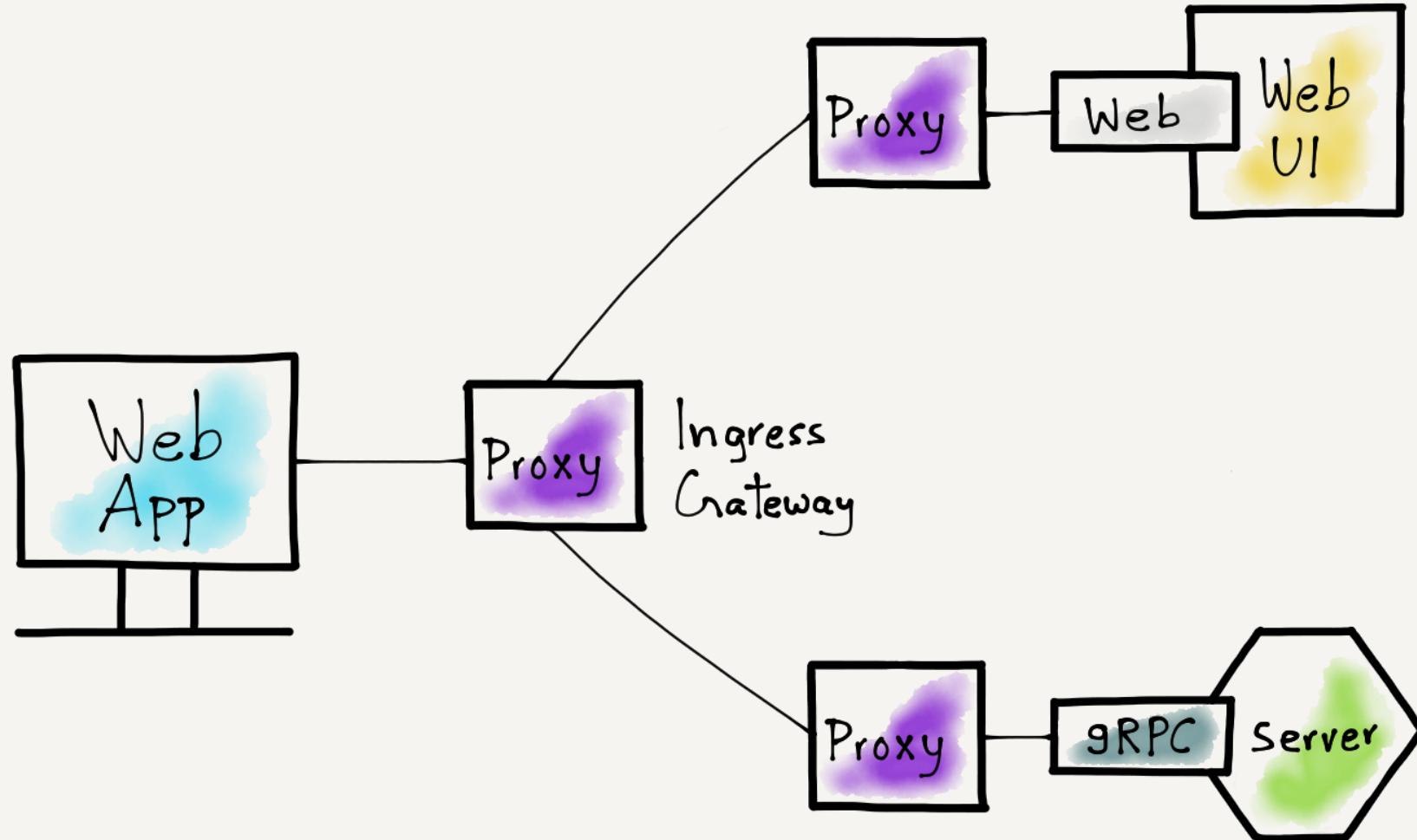
Build the Docker image

```
1 docker build -f docker/web-ui.Dockerfile  
2 | | | | | -t vnoronha/grpc-web-istio-demo:web-ui .
```

Push the Docker image

```
1 docker push vnoronha/grpc-web-istio-demo:web-ui
```

The Emoji Web App Deployment



The Kubernetes Configuration for the Server

```
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: server
5   labels:
6     app: server
7 spec:
8   ports:
9     - name: grpc-web
10    port: 9000
11 selector:
12   app: server
```

```
1 apiVersion: extensions/v1beta1
2 kind: Deployment
3 metadata:
4   name: server
5 spec:
6   replicas: 1
7   template:
8     spec:
9       containers:
10      - name: server
11        image: grpc-web-istio-demo:server
12        ports:
13          - containerPort: 9000
```

The Kubernetes Configuration for the Web UI

```
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: web-ui
5   labels:
6     app: web-ui
7 spec:
8   ports:
9     - name: http
10    port: 9001
11 selector:
12   app: web-ui
```

```
1 apiVersion: extensions/v1beta1
2 kind: Deployment
3 metadata:
4   name: web-ui
5 spec:
6   replicas: 1
7   template:
8     spec:
9       containers:
10      - name: web-ui
11        image: grpc-web-istio-demo:web-ui
12        ports:
13          - containerPort: 9001
```

The Istio Gateway Configuration

```
1 apiVersion: istio.io/v1alpha3
2 kind: Gateway
3 metadata:
4   name: gateway
5 spec:
6   selector:
7     istio: ingressgateway
8   servers:
9     - port:
10       number: 80
11       name: http
12       protocol: HTTP
```

```
1 apiVersion: istio.io/v1alpha3
2 kind: VirtualService
3 spec:
4   gateways:
5     - gateway
6   http:
7     - match:
8       - uri:
9         prefix: /proto.EmojiService
10    route:
11      - destination:
12        host: server
13    route:
14      - destination:
15        host: web-ui
```

```
$ kubectl apply -f <(istioctl kube-inject -f server.yaml)
service/server created
deployment.extensions/server created
$
$ kubectl apply -f <(istioctl kube-inject -f web-ui.yaml)
service/web-ui created
deployment.extensions/web-ui created
$
$ kubectl apply -f gateway.yaml
gateway.networking.istio.io/gateway created
virtualservice.networking.istio.io/virtual-service created
$ █
```

```
$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
server-6985ccb646-x82pv	2/2	Running	0	26s
web-ui-dd6ddcbcb-7jwq4	2/2	Running	0	13s
\$ █				

192.168.99.100:31380 Not Secure | 192.168.99.100:31380

I like 🍕 and 🍣!

Elements Console Sources Network Performance Memory Application Security Audits

View: Group by frame Preserve log Disable cache Offline Online

Filter Hide data URLs All XHR JS CSS Img Media Font Doc WS Manifest Other

5000 ms 10000 ms 15000 ms 20000 ms 25000 ms 30000 ms 35000 ms 40000 ms 45000 ms

Name Headers Preview Response Timing

InsertEmojis

General

Request URL: http://192.168.99.100:31380/proto.EmojiService/InsertEmojis

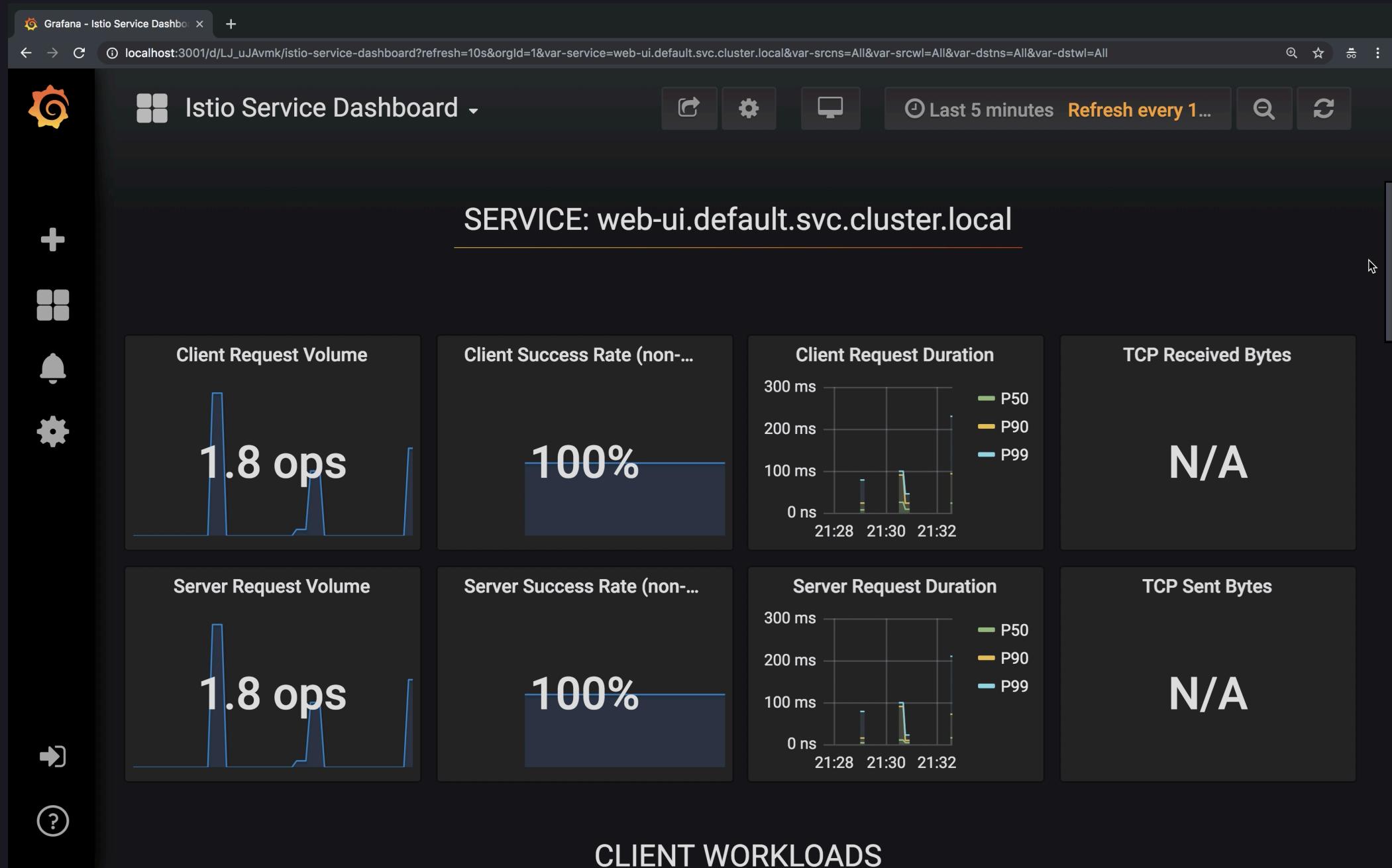
Request Method: POST

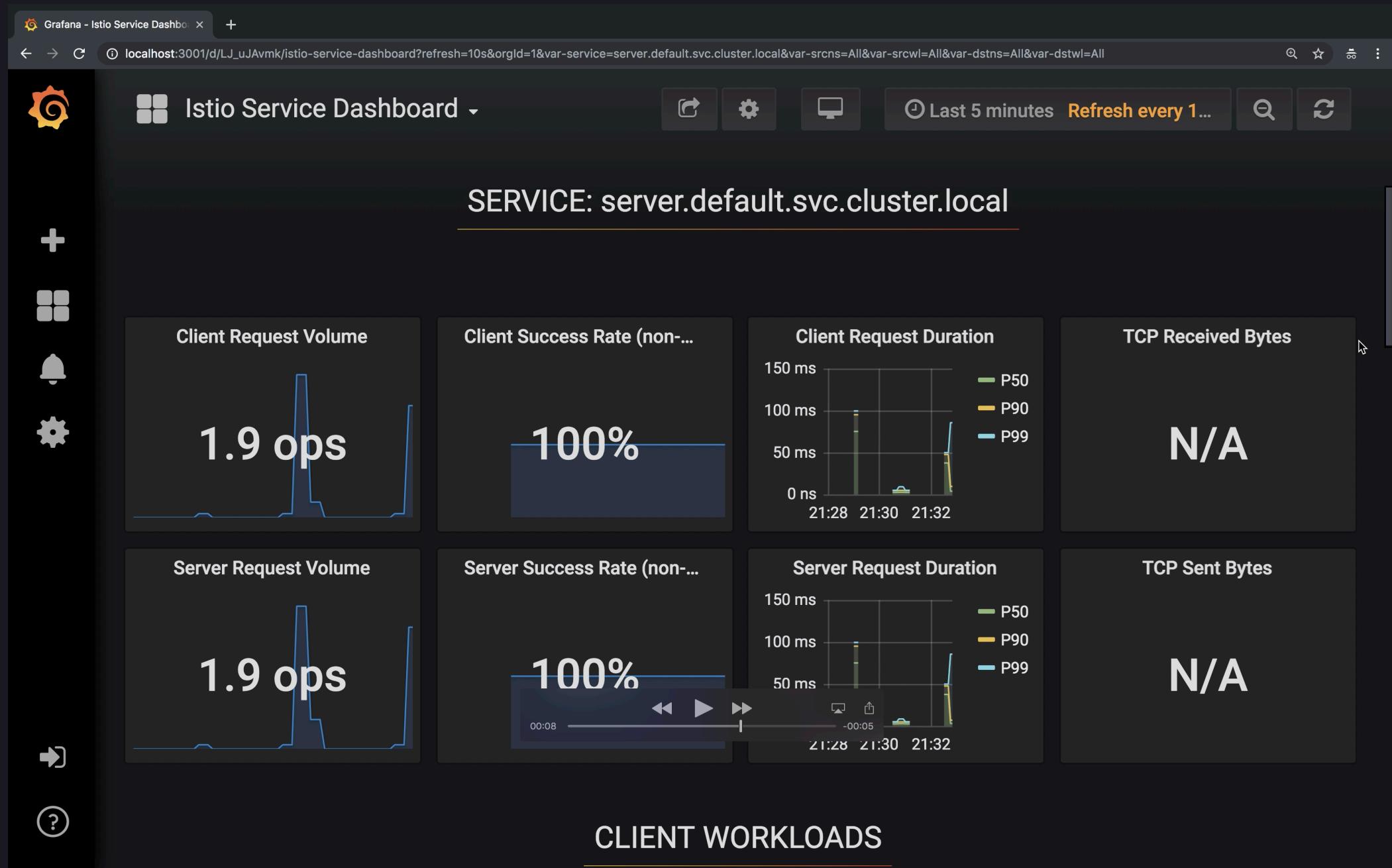
Status Code: 200 OK

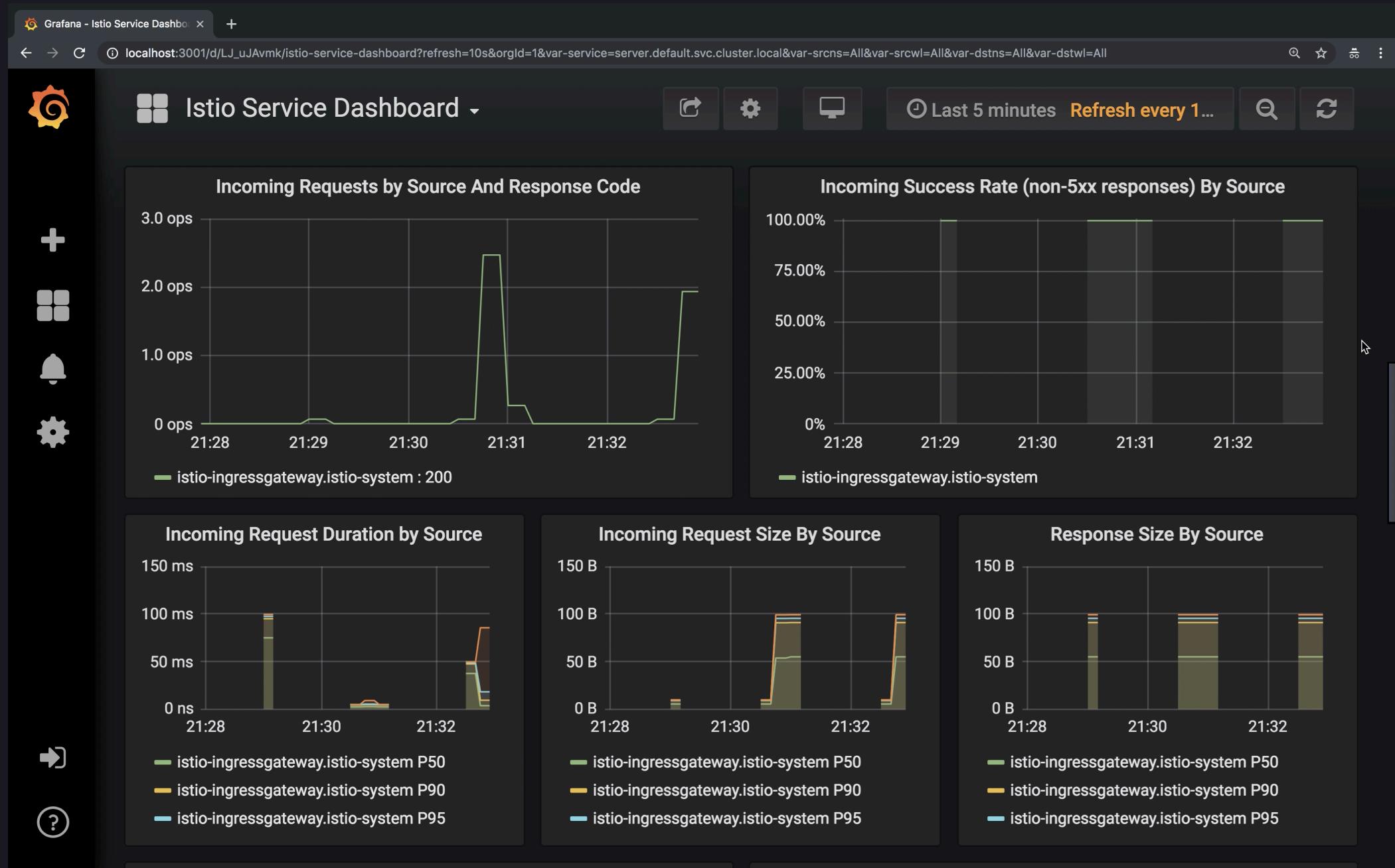
Remote Address: 192.168.99.100:31380

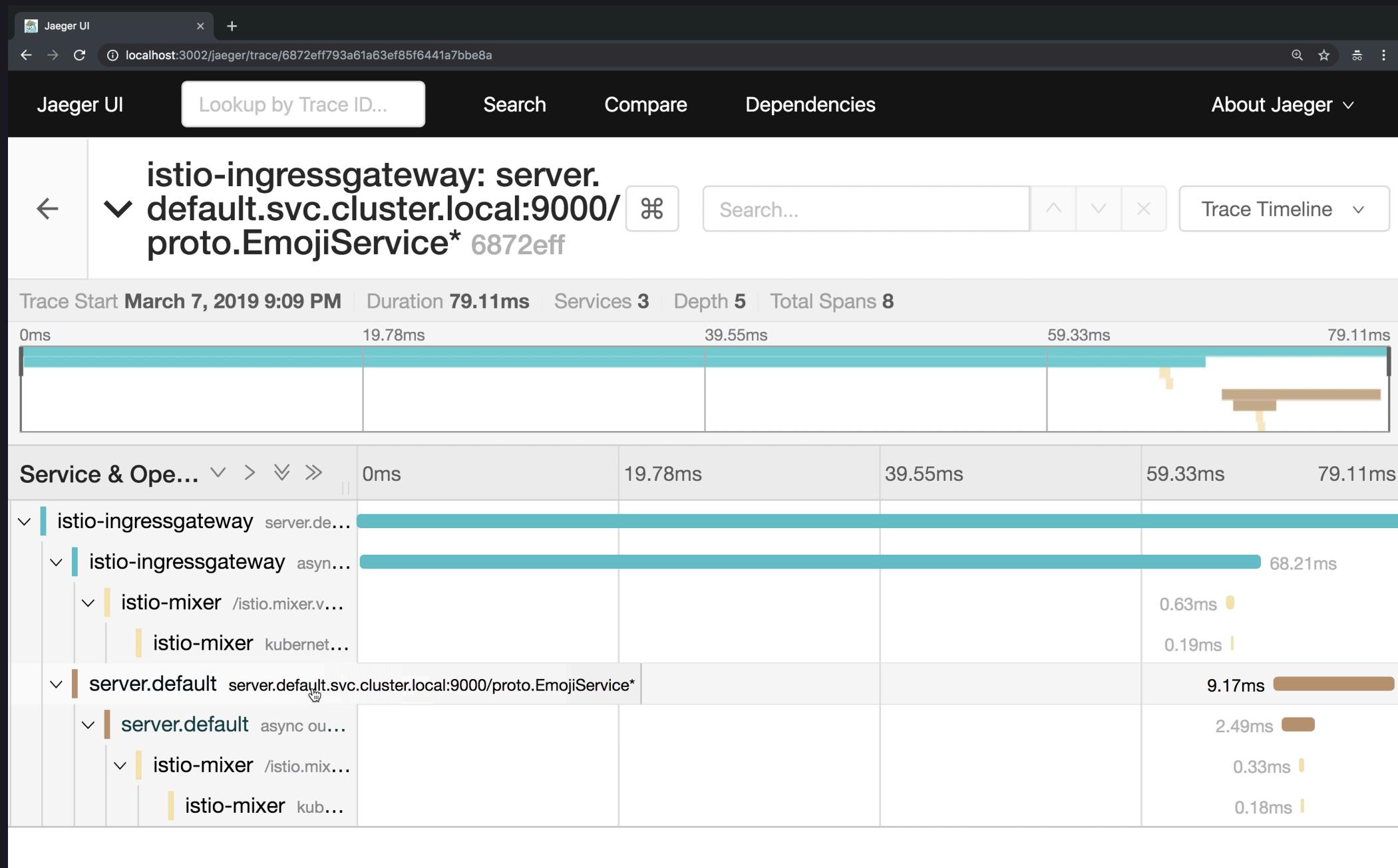
Referrer Policy: no-referrer-when-downgrade

34 requests | 415 KB tra...









Jaeger UI +

localhost:3002/jaeger/trace/6872eff793a61a63ef85f6441a7bbe8a

Jaeger UI Lookup by Trace ID... Search Compare Dependencies About Jaeger

istio-ingressgateway: server.
default.svc.cluster.local:9000/
proto.EmojiService* 6872eff

Search... Trace Timeline

Trace Start March 7, 2019 9:09 PM | Duration 79.11ms | Services 3 | Depth 5 | Total Spans 8

0ms 19.78ms 39.55ms 59.33ms 79.11ms

Service & Ope... 0ms 19.78ms 39.55ms 59.33ms 79.11ms

	guid:x-request-id	"e87d206f-1560-98f5-8752-67fdc1c8b749"
	http.url	"http://192.168.99.100:31380/proto.EmojiService/InsertEmojis"
	http.method	"POST"
	downstream_cluster	"_"
	user_agent	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.106 Safari/537.36"
	http.protocol	"HTTP/2"
	request_size	12
	upstream_cluster	"inbound 9000 grpc-web server.default.svc.cluster.local"
	http.status_code	200

Kiali Console

localhost:3003/console/graph/namespaces?layout=dagre&duration=60&edges=hide&graphType=app&injectServiceNodes=true&namespaces=default

kiali

Overview

Graph

Applications

Workloads

Services

Istio Config

Graph Type: App

Namespace: default

Edge Labels: App

Fetching: Last min (Every 15 sec)

HTTP Traffic (requests per second):

Total	%Success	%Error
1.60	100.00	0.00

HTTP - Total Request Traffic min / max:

The graph visualization shows the following connections:

- An arrow from the **istio-ingressgateway** node (labeled "istio-system") to a **server** node.
- An arrow from the **istio-ingressgateway** node to a **web-ui** node.
- An arrow from the **server** node to another **server** node.
- An arrow from the **web-ui** node to another **web-ui** node.

Legend:

- +
-
- ↔
- 1
- 2
- Legend

Kiali Console

localhost:3003/console/graph/namespaces?layout=dagre&duration=60&edges=hide&graphType=app&injectServiceNodes=true&namespaces=default

kiali

Overview

Graph

Applications

Workloads

Services

Istio Config

Graph

Namespace: default

Display

Edge Labels

Graph Type: App

Fetching: Last min

Every 15 sec

Services: server

HTTP Traffic (requests per second):

	Total	%Success	%Error
In	0.58	100.00	0.00
Out	0.00	100.00	0.00

In

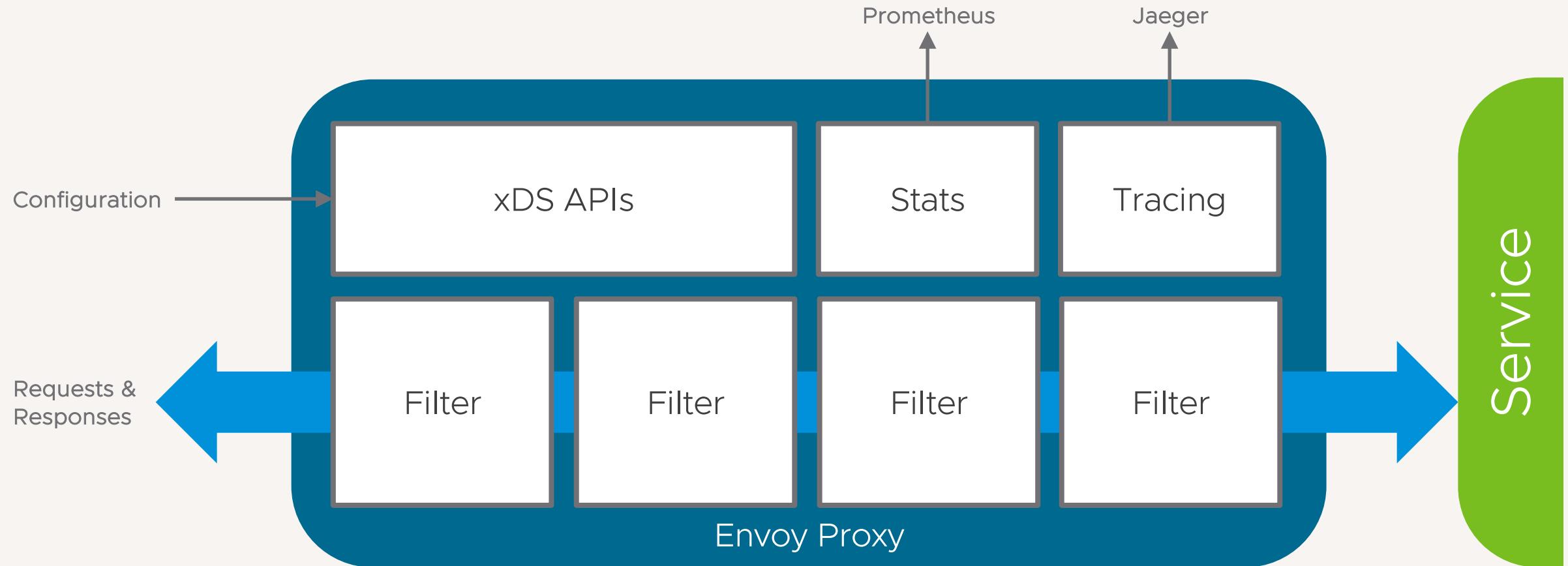
Out

0 25 50 75 100 %

OK 3xx 4xx 5xx

The Kiali interface displays a graph of service interactions. The 'Graph' tab is selected. The graph shows three main nodes: 'istio-ingressgateway' (labeled 'istio-system'), 'server', and 'web-ui'. Arrows indicate traffic flow: from 'istio-ingressgateway' to 'server' (green arrow), from 'istio-ingressgateway' to 'web-ui' (green arrow), and from 'server' to 'web-ui' (green arrow). The 'server' node has two green square icons. Below the graph is a timeline bar with markers at 00:13 and -00:07. A legend at the bottom includes icons for zooming, selecting nodes 1 and 2, and a 'Legend' button.

The Envoy Proxy



gRPC – JSON Transcoder

Rate Limiting

Conclusion

- Protobufs – API Contracts, Data Models, Compatibility
- gRPC – Based on HTTP/2, Client Stubs, Performance
- gRPC-Web – Protobufs + gRPC
- Envoy – Built-in HTTP-gRPC Transcoder
- Istio – Envoy, Metrics, Tracing, Service Graph

Hack with gRPC-Web and Istio!



Thank You

 venilnoronha.io

 [venilnoronha](https://github.com/venilnoronha)

 [venilnoronha](https://twitter.com/venilnoronha)

