

## Contents

1 Installation .....	2
1.1 Sample grpc hello-world service.....	2
1.2 Kreya App.....	2
1.3 Apisix Docker.....	2
1.4 Sample Angular Project .....	2
1.5 Call hello-world service via Kreya app.....	3
2 APISIX configurations.....	4
2.1 Upstream configuration.....	4
2.2 Route configuration.....	5
2.3 Test Route.....	5
3 Test Steps .....	5
3.1 Route with cors plugin.....	5
3.1.1 Add cors plugin to route.....	5
3.1.2 Test with Kreya App.....	6
3.2 Route with grpc-web plugin.....	6
3.2.1 Add grpc-web plugin to route.....	6
3.2.2 Test with Kreya App.....	7
3.3 Add cors & grpc-web plugins to route.....	7
3.3.1 Add cors & grpc-web plugins to route .....	7
3.3.2 Test with Kreya App.....	8
3.3.3 Test with Angular Project .....	8
3.4 Add grpc-web plugins to route & global cors plugin.....	10
3.4.1 Add grpc-web plugin to route.....	10
3.4.2 Add global cors plugin.....	10
3.4.3 Test with Kreya App.....	11
3.4.4 Test with Angular Project .....	11
4 Conclusion.....	12

# 1 Installation

## 1.1 Sample grpc hello-world service

<https://hub.docker.com/r/kutbar/grpc-hello-world> app runs on port 9000 default. This service doesn't need an authentication. It receives a name as an input, and responds with concat("Hello "+name) as an output. I run it on an external server.

## 1.2 Kreya App

<https://kreya.app/downloads/> for testing grpc services.

## 1.3 Apisix Docker

The following commands described in <https://apisix.apache.org/docs/apisix/getting-started/#step-1-install-apache-apisix> are used for installation.

```
# Download the docker-compose file of Apache APISIX
git clone https://github.com/apache/apisix-docker.git
# Switch the current directory to the apisix-docker/example
cd apisix-docker/example
```

Update docker-compose file in apisix\_conf directory



docker-compose.yml

Update apisix conf file



config.yaml

```
# Start Apache APISIX with docker-compose
docker-compose -p docker-apisix up -d
```

## 1.4 Sample Angular Project

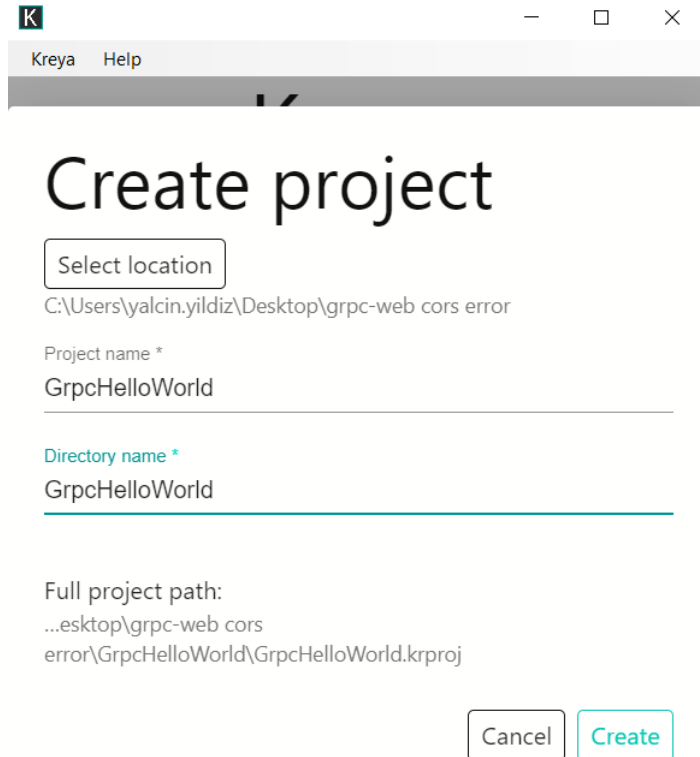
Unzip first, then run "npm install" and "npm start" commands in the terminal of an appropriate IDE(I'm using Microsoft Visual Studio).



grpc-web-angular.rar

## 1.5 Call hello-world service via Kreya app

- First, you need to create a new project with Kreya app. To do it, select “Create Project” on the main screen after starting the app. Then, specify project name and directory name after selecting suitable project location.



Kreya Help

### Create project

Select location  
C:\Users\yalcin.yildiz\Desktop\grpc-web cors error

Project name \*  
GrpcHelloWorld

Directory name \*  
GrpcHelloWorld

Full project path:  
...esktop\grpc-web cors error\GrpcHelloWorld\GrpcHelloWorld.krproj

Cancel Create

- Then, you need to select proto file. It is available at `grpc-web-angular\src\app\proto\hello.proto` in angular project, also you can use the following file with the same content.



hello.proto

## Initial project setup

How would you like to import your protos?

- gRPC proto files ?
- gRPC server reflection ?
- gRPC proto file descriptor set ?

Proto files Import paths

[Add proto directory](#) [Add proto files](#)

..\grpc-web-angular\src\app\proto\hello.proto

Skip Save

c. We can see that service is up and running.

The screenshot shows a gRPC client interface. At the top, it displays the service and method: `hello.HelloGrpc.SayHello`. A `Send` button is visible, and a status box shows `0 Ok`. Below this, there are tabs for `Request`, `Metadata`, `Auth`, `Settings`, `Response`, `Header`, `Trailer`, and `Trace`. The `Settings` tab is active, showing fields for `Endpoint` (set to `http://192.168.1.78:9000`), `Server certificate (TLS/SSL) validation` (set to `Enabled`), `Mode` (set to `gRPC`), and `Max message size (in bytes)`. The `Response` tab is also visible, showing a JSON response: `{\"message\": \"Hello Yalcin!\"}`.

## 2 APISIX configurations

### 2.1 Upstream configuration

```
curl "http://localhost:9080/apisix/admin/upstreams/1" -H "X-API-KEY: edd1c9f034335f136f87ad84b625c8f1" -X PUT -d '
```

```
{
  "nodes": [
    {
      "host": "192.168.1.78",
      "port": 9000,
      "weight": 1
    }
  ],
  "timeout": {
    "connect": 2,
    "send": 3,
    "read": 4
  },
  "type": "roundrobin",
  "scheme": "grpc",
  "pass_host": "node",
  "name": "Kutbar grpc upstream",
  "desc": "Kutbar grpc upstream",
  "keepalive_pool": {
    "idle_timeout": 60,
    "requests": 1000,
    "size": 320
  }
}
```

## 2.2 Route configuration

```
curl "http://localhost:9080/apisix/admin/routes/1" -H "X-API-KEY:
edd1c9f034335f136f87ad84b625c8f1" -X PUT -d '
{
  "uri": "/*",
  "name": "Kutbar grpc route",
  "methods": [
    "POST",
    "OPTIONS"
  ],
  "upstream_id": "1",
  "status": 1
}'
```

## 2.3 Test Route

New route is working as expected.

The screenshot displays the Apisix Admin Console interface. At the top, a gRPC test is initiated with the endpoint 'hello.HelloGrpc.SayHello'. A 'Send' button is visible, and a green box indicates '0 Ok' responses. Below the test area, there are tabs for 'Request', 'Metadata', 'Auth', 'Settings', 'Response', 'Header', 'Trailer', and 'Trace'. The 'Request' tab is active, showing the endpoint 'http://localhost:9079', 'Server certificate (TLS/SSL) validation' set to 'Enabled', and 'Mode' set to 'gRPC'. The 'Response' tab is also active, showing a JSON response: {\"message\": \"Hello Yalcin!\"}.

## 3 Test Steps

### 3.1 Route with cors plugin

#### 3.1.1 Add cors plugin to route

```
curl "http://localhost:9080/apisix/admin/routes/1" -H "X-API-KEY:
edd1c9f034335f136f87ad84b625c8f1" -X PUT -d '
{
  "uri": "/*",
  "name": "Kutbar grpc route",
  "methods": [
    "POST",
```

```

"OPTIONS"
],
"plugins": {
  "cors": {
    "allow_credential": true,
    "allow_headers": "authorization,content-type,x-grpc-web,x-user-agent,origin,referrer",
    "allow_methods": "POST,OPTIONS",
    "allow_origins": "http://localhost:4200",
    "disable": false,
    "expose_headers": "authorization,content-type,x-grpc-web,x-user-agent,origin,referrer",
    "max_age": 5
  }
},
"upstream_id": "1",
"status": 1
}'

```

### 3.1.2 Test with Kreya App

The cors plugin is working fine.

The screenshot shows a gRPC client interface. The request is to the endpoint `http://localhost:9079` with the message `hello.HelloGrpc.SayHello`. The response is successful (0 Ok) and took 14ms. The response headers are as follows:

Header	Value
date	Tue, 12 Apr 2022 10:52:14 GMT
server	APISIX/2.13.0
access-control-allow-origin	http://localhost:4200
vary	Origin
access-control-allow-methods	POST,OPTIONS
access-control-max-age	5
access-control-expose-headers	authorization,content-type,x-grpc-web
access-control-allow-headers	authorization,content-type,x-grpc-web
access-control-allow-credentials	true

## 3.2 Route with grpc-web plugin

### 3.2.1 Add grpc-web plugin to route

```

curl "http://localhost:9080/apisix/admin/routes/1" -H "X-API-KEY:
edd1c9f034335f136f87ad84b625c8f1" -X PUT -d '
{
  "uri": "/*",
  "name": "Kutbar grpc route",
  "methods": [
    "POST",
    "OPTIONS"
  ],

```

```
"plugins": {
  "grpc-web": {}
},
"upstream_id": "1",
"status": 1
}'
```

### 3.2.2 Test with Kreya App

Since grpc-web works with HTTP1.1, port is updated to 9080. Both “gRPC Web” and “gRPC Web (Text)” modes are working fine and we have allow-origin header with the value ‘\*’.

The screenshot shows the Kreya app interface. On the left, the 'Request' tab is active, showing the endpoint 'http://localhost:9080', 'Server certificate (TLS/SSL) validation' set to 'Disabled', 'Mode' set to 'gRPC Web', and 'Max message size (in bytes)' set to an empty field. The 'Send' button is highlighted in green. On the right, the 'Response' tab is active, showing the following headers:

Header	Value
date	Tue, 12 Apr 2022 08:50:55 GMT
transfer-encoding	chunked
connection	keep-alive
server	APISIX/2.13.0
access-control-allow-origin	*

## 3.3 Add cors & grpc-web plugins to route

### 3.3.1 Add cors & grpc-web plugins to route

```
curl "http://localhost:9080/apisix/admin/routes/1" -H "X-API-KEY:
edd1c9f034335f136f87ad84b625c8f1" -X PUT -d '
```

```
{
  "uri": "/*",
  "name": "Kutbar grpc route",
  "methods": [
    "POST",
    "OPTIONS"
  ],
  "plugins": {
    "cors": {
      "allow_credential": true,
      "allow_headers": "authorization,content-type,x-grpc-web,x-user-agent,origin,referrer",
      "allow_methods": "POST,OPTIONS",
      "allow_origins": "http://localhost:4200",
      "disable": false,
      "expose_headers": "authorization,content-type,x-grpc-web,x-user-agent,origin,referrer",
      "max_age": 5
    }
  }
}
```

```

    },
    "grpc-web": {}
  },
  "upstream_id": "1",
  "status": 1
}'

```

### 3.3.2 Test with Kreya App

Both “gRPC Web” and “gRPC Web (Text)” modes are working successful. Cors headers other than allow-origin are available as expected. Allow-origin header value should be “<http://localhost:4200>”

The screenshot shows a web client interface for a gRPC Web (Text) request. The request metadata is as follows:

Request	Metadata	Auth	Settings
Origin	http://localhost:4200		
Authorization	Bearer xxx		

The response headers are as follows:

Response	Header	Trailer	Trace
date	Tue, 12 Apr 2022 10:39:38 GMT		
transfer-encoding	chunked		
connection	keep-alive		
server	APISIX/2.13.0		
access-control-allow-origin	*		
vary	Origin		
access-control-allow-methods	POST,OPTIONS		
access-control-max-age	5		
access-control-expose-headers	authorization,content-type,x-grpc-web		
access-control-allow-headers	authorization,content-type,x-grpc-web		
access-control-allow-credentials	true		

### 3.3.3 Test with Angular Project

As in previous test, authorization and origin headers are sent by Angular app.

```

const helloService = new HelloGrpcClient('http://localhost:9080', null, {
withCredentials: true });

const helloReq = new HelloRequest();
helloReq.setName('Angular');
const helloHeaders = {
  authorization: 'Bearer xxx'
};

```

Cors headers other than allow-origin and allow-headers are available as expected. Allow-origin header value should be <http://localhost:4200>, and allow-headers value should be “authorization,content-type,x-grpc-web,x-user-agent,origin,referrer”.



**General**

Request URL: http://localhost:9080/hello.HelloGrpc/SayHello  
 Request Method: OPTIONS  
 Status Code: 200 OK  
 Remote Address: [::1]:9080  
 Referrer Policy: strict-origin-when-cross-origin

**Response Headers**

Access-Control-Allow-Credentials: true  
**Access-Control-Allow-Headers: content-type,x-grpc-web,x-user-agent**  
 Access-Control-Allow-Methods: POST  
**Access-Control-Allow-Origin: \***  
 Access-Control-Expose-Headers: authorization,content-type,x-grpc-web,x-user-agent,origin,referrer  
 Access-Control-Max-Age: 5  
 Connection: keep-alive  
 Date: Tue, 12 Apr 2022 10:39:47 GMT  
 Server: APISIX/2.13.0  
 Transfer-Encoding: chunked  
 Vary: Origin

**Request Headers**

Accept: \*/\*  
 Accept-Encoding: gzip, deflate, br  
**Accept-Language: en-US,en;q=0.9**  
 Access-Control-Request-Headers: authorization,content-type,x-grpc-web,x-user-agent  
 Access-Control-Request-Method: POST  
 Cache-Control: no-cache  
 Connection: keep-alive  
 Host: localhost:9080  
 Origin: http://localhost:4200  
 Pragma: no-cache  
 Referer: http://localhost:4200/  
 Sec-Fetch-Dest: empty  
 Sec-Fetch-Mode: cors  
 Sec-Fetch-Site: same-site  
 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.88 Safari/53

Thus, we are facing with cors error in Angular app.

**Request Headers**

⚠ Provisional headers are shown [Learn more](#)  
 Accept: application/grpc-web-text  
 authorization: Bearer xxx  
 Content-Type: application/grpc-web-text  
 Referer: http://localhost:4200/  
 sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="100", "Google Chrome";v="100"  
 sec-ch-ua-mobile: ?0  
 sec-ch-ua-platform: "Windows"  
 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.88 Safari/537.36  
 X-Grpc-Web: 1  
 X-User-Agent: grpc-web-javascript/0.1

11 requests | 4.9 MB transferred | 4.9 MB resources | Finish: 6

**Console**

Angular is running in the development mode. Call enableProdMode() to enable the production mode. core.js:38781

⚠ Access to XMLHttpRequest at 'http://localhost:9080/hello.HelloGrpc/SayHello' from origin 'http://localhost:4200' has been blocked by CORS policy: Response to preflight request doesn't pass access control check: The value of the 'Access-Control-Allow-Origin' header in the response must not be the wildcard '\*' when the request's credentials mode is 'include'. The credentials mode of requests initiated by the XMLHttpRequest is controlled by the withCredentials attribute. app.component.ts:30

2  
 Http response at 400 or 500 level app.component.ts:31

Access to XMLHttpRequest at 'http://localhost:9080/hello.HelloGrpc/SayHello' from origin 'http://localhost:4200' has been blocked by CORS policy: Response to preflight request doesn't pass access control check: The value of the 'Access-Control-Allow-Origin' header in the response must not be the wildcard '\*' when the request's credentials mode is 'include'. The credentials mode of requests initiated by the XMLHttpRequest is controlled by the withCredentials attribute.

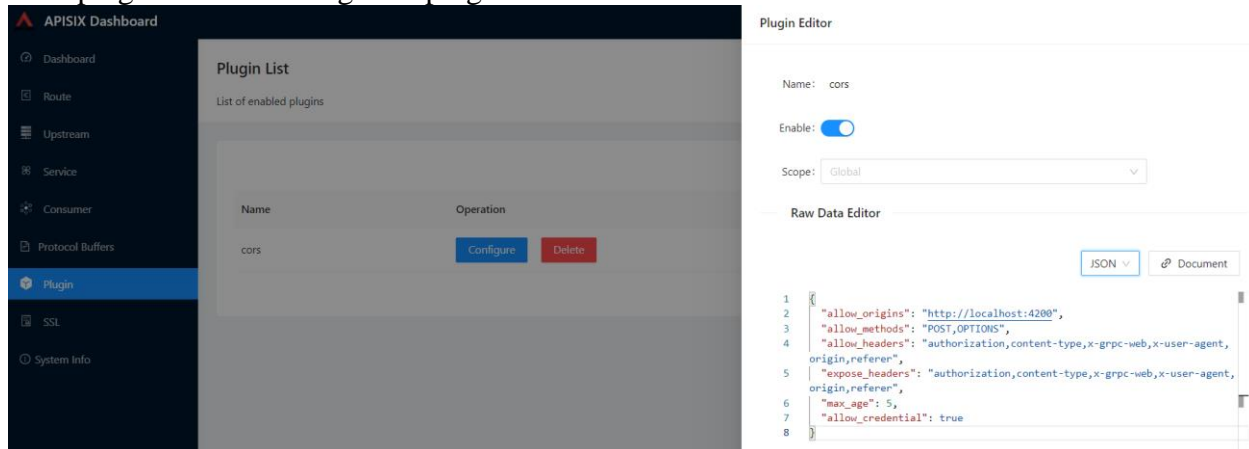
## 3.4 Add grpc-web plugins to route & global cors plugin

### 3.4.1 Add grpc-web plugin to route

```
curl "http://localhost:9080/apisix/admin/routes/1" -H "X-API-KEY:
edd1c9f034335f136f87ad84b625c8f1" -X PUT -d '
{
  "uri": "/*",
  "name": "Kutbar grpc route",
  "methods": [
    "POST",
    "OPTIONS"
  ],
  "plugins": {
    "grpc-web": {}
  },
  "upstream_id": "1",
  "status": 1
}'
```

### 3.4.2 Add global cors plugin

Cors plugin is added as a global plugin.



The screenshot displays the APISIX Dashboard interface. On the left is a sidebar with navigation options: Dashboard, Route, Upstream, Service, Consumer, Protocol Buffers, Plugin (highlighted), and SSL. The main content area is titled 'Plugin List' and shows a table of enabled plugins. The table has columns for 'Name' and 'Operation'. One plugin, 'cors', is listed with 'Configure' and 'Delete' buttons. To the right is the 'Plugin Editor' for the 'cors' plugin. It shows 'Name: cors', 'Enable: ', and 'Scope: Global'. Below is the 'Raw Data Editor' with a 'JSON' dropdown and a 'Document' icon. The raw data is a JSON object:

```
1 {
2   "allow_origins": "http://localhost:4200",
3   "allow_methods": "POST,OPTIONS",
4   "allow_headers": "authorization,content-type,x-grpc-web,x-user-agent,
5   origin,referer",
6   "expose_headers": "authorization,content-type,x-grpc-web,x-user-agent,
7   origin,referer",
8   "max_age": 5,
9   "allow_credential": true
10 }
```

```
{
  "allow_origins": "http://localhost:4200",
  "allow_methods": "POST,OPTIONS",
  "allow_headers": "authorization,content-type,x-grpc-web,x-user-
agent,origin,referer",
  "expose_headers": "authorization,content-type,x-grpc-web,x-user-
agent,origin,referer",
  "max_age": 5,
  "allow_credential": true
}
```

### 3.4.3 Test with Kreya App

Both “gRPC Web” and “gRPC Web (Text)” modes are working successful. Cors headers other than allow-origin are available as expected. Allow-origin header value should be “<http://localhost:4200>”

The screenshot shows the Kreya app interface. At the top, a request is shown: "gRPC Web (Text) hello.HelloGrpc.SayHello" with a "Send" button and a "0 Ok" status box. A "27ms" response time is also visible. Below this, there are tabs for "Request", "Metadata", "Auth", "Settings", "Response", "Header", "Trailer", and "Trace". The "Request" tab is active, showing "Origin" as "http://localhost:4200" and "Authorization" as "Bearer xxx". The "Response" tab is also active, showing a list of headers:

Header	Value
date	Wed, 13 Apr 2022 05:49:05 GMT
transfer-encoding	chunked
connection	keep-alive
server	APISIX/2.13.0
access-control-allow-origin	*
vary	Origin
access-control-allow-methods	POST,OPTIONS
access-control-max-age	5
access-control-expose-headers	authorization,content-type,x-grpc-web
access-control-allow-headers	authorization,content-type,x-grpc-web
access-control-allow-credentials	true

### 3.4.4 Test with Angular Project

Cors headers other than allow-origin are available as expected. Allow-origin header value should be <http://localhost:4200>.  
OPTIONS;

The screenshot shows a browser's developer console. Under the "General" section, the following details are visible:

- Request URL: <http://localhost:9080/hello.HelloGrpc/SayHello>
- Request Method: OPTIONS
- Status Code: 200 OK
- Remote Address: [::1]:9080
- Referrer Policy: strict-origin-when-cross-origin

Under the "Response Headers" section, the following headers are listed:

- Access-Control-Allow-Credentials: true
- Access-Control-Allow-Headers: authorization,content-type,x-grpc-web,x-user-agent,origin,referrer
- Access-Control-Allow-Methods: POST,OPTIONS
- Access-Control-Allow-Origin: <http://localhost:4200>
- Access-Control-Expose-Headers: authorization,content-type,x-grpc-web,x-user-agent,origin,referrer

POST;

```
▼ General
Request URL: http://localhost:9080/hello.HelloGrpc/SayHello
Request Method: POST
Status Code: ● 200
Referrer Policy: strict-origin-when-cross-origin

▼ Response Headers View source
Access-Control-Allow-Credentials: true
Access-Control-Allow-Headers: authorization,content-type,x-grpc-web,x-user-agent,origin,referrer
Access-Control-Allow-Methods: POST,OPTIONS
Access-Control-Allow-Origin: *
Access-Control-Expose-Headers: authorization,content-type,x-grpc-web,x-user-agent,origin,referrer
Access-Control-Max-Age: 5
```

## 4 Conclusion

We have grpc services that do and do not require authentication. Grpc services that do not require authentication can be perfectly consumed by web clients after using grpc-web and cors plugins. But we are experiencing cors issues on client side for the grpc services that do require authentication.

Grpc-web plugin manipulates allow-origin and allow-headers cors headers incorrect when it is used with cors plugin in the same route as you can see in section 3.3.

Grpc-web plugin manipulates allow-origin cors header incorrect when it is used with global cors plugin as you can see in section 3.4.