HTTP API Trigger Documentation

The **HTTP trigger** allows a Zenphi flow to be started by receiving a "Post" HTTP request. This means that a flow can be initiated from any system capable of sending an HTTP call. When a flow is published using this trigger, Zenphi generates an endpoint URL that can be used in the HTTP request.

An example of how to trigger the flow could be a webpage where when a request is made to receive product information, a POST call is made to the HTTP API trigger on its dedicated address, delivering the product ID number requested and a tokenised ID for the customers contact to serve the information to their inbox.

But even more interesting is that the flow could call back to the website over a returning API call and serve the information in real time to the requester.

Key aspects of the HTTP Trigger:

- **Functionality**: It receives "Post" HTTP requests to initiate flows.
- **Endpoint URL**: Zenphi provides a unique endpoint URL for each flow published with this trigger.
- **Variables**: The trigger can utilise variables from the received HTTP request to create a workflow.
- **Metadata**: Each trigger may include metadata that translates into start parameters for the flow.

To set up the trigger you have the following tabs available:

- The **Variables Tab** displays a list of variables included in the HTTP request:

Display Name	Name	Type
Http Method	httpMethod	String
Http Body	httpBody	String
Remote IP	remoteIp	String
Content Type	contentType	String
Form Data	formData	String
Query String Data	queryStringData	String
Headers	headers	Collection

- The **Invocation Tab** shows the endpoint URL and HTTP method after the workflow is published; these can be copied and used:
  - HTTP Method (POST)
  - URL Using Authentication Header:
    - URL
    - Auth Header
  - URL Including Authorization Token
    - URL
- The **Usage** tab allows viewing of flow actions that use the trigger's output.

    - The **Conditional Run** tab lets you set rules for when the flow should be run, to avoid unnecessary executions. For example to only run for a specific IP or Form Data.

Example Scenario:

1. A user completes a form on a website.
2. Clicking "Submit" sends an HTTP request to Zenphi, containing the user's data.
3. This HTTP request starts a workflow.
4. The workflow then uses the data from the HTTP body to create a record in a Zenphi table.

The **output** from the trigger offers the following information:

- Http Method - The method of this invocation E.g. GET, POST
- Http Body - The contents of the body
- Remote IP - The IP address of the caller
- Content Type - The content type of the request
- Form Data - The form data of the request converted to a JSON string
- Query String Data - The query string converted to a JSON string
- Headers - A collection of HTTP headers associated with the request
    - Header Name - The name of the HTTP header, e.g., Content-Type or Authorization
    - Header Value - The value assigned to the HTTP header, e.g., application/json or Bearer token

Code examples

Example JavaScript to call the HTTP API trigger from a webpage (with `corsProxy` temporary workaround):

JavaScript

```
<script>
// To handle CORS errors, create a CORS Proxy to call through. Use the below
service for testing purposes only
const corsProxy = 'http://cors-anywhere.herokuapp.com/';

//The URL and Auth Token to call, found in the section "URL Using
Authentication Header". Copy the full URL field as is for zenphiPostUrl, and
copy only the token in the Auth Header (the string after
"x-zenphi-httptriggertoken:") for the postToken.
const zenphiPostUrl = 'https://webhook.eu1.zenphi.io/http/[Your flow
ID]/[Your trigger ID]';
const postToken = '[token]'; // Replace with your actual token

// Construct headers and body
const requestOptions = {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'x-zenphi-httptriggertoken': postToken // Pass token in header
  },
  body: JSON.stringify({
    key1: 'value1',
    key2: 'value2' // Replace with actual payload
  })
};

fetch(corsProxy + zenphiPostUrl, requestOptions)
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    return response.json();
  })
  .then(data => {
    console.log('Flow triggered successfully:', data);
  })
  .catch(error => {
    console.error('Error triggering flow:', error);
  });
</script>
```

For a more permanent CORS solution, you will have to provide a proxy yourself where to direct your call. Example of a Node.js proxy using `http-proxy-middleware`:

```javascript
const express = require('express');
const { createProxyMiddleware } = require('http-proxy-middleware');

const app = express();
app.use('/api', createProxyMiddleware({
  target: 'https://webhook-zone1.zenphi.io',
  changeOrigin: true,
}));

app.listen(3001, () => {
  console.log('Proxy server running on port 3001');
});
```

Now, your frontend can make requests to `http://localhost:3001/api` instead of directly calling Zenphi's API, replacing the `corsProxy` variable in the JavaScript example earlier.

The same example as a PHP CURL call (with `corsProxy` temporary workaround):

```C/C++
// To handle CORS errors, create a CORS Proxy to call through. Use the below service for testing purposes only:
$corsProxy = 'http://cors-anywhere.herokuapp.com/';

/* The URL and Auth Token to call, found in the section "URL Using Authentication Header". Copy the full URL field as is for $zenphiPostUrl, and copy only the token in the Auth Header (the string after "x-zenphi-httptriggertoken:") for the $postToken. */
$zenphiPostUrl = 'https://webhook-zone1.zenphi.io/http/[Your-Flow-ID]/[Your-Trigger-ID]';
$postToken = '[Token]';

// Construct headers and body
$headers = array(
    'Content-Type: application/json',
    'x-zenphi-httptriggertoken: ' . $postToken // Pass token in header
);

$payload = array(
    'key1' => 'value1',
    'key2' => 'value2' // Replace with actual payload
);

$ch = curl_init($corsProxy . $zenphiPostUrl);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_HEADER, false);
curl_setopt($ch, CURLOPT_POST, true);
curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($payload));
curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);

$response = curl_exec($ch);
$statusCode = curl_getinfo($ch, CURLINFO_HTTP_CODE);

if ($statusCode == 200) {
    echo "Flow triggered successfully: $response\n";
} else {
    echo "Failed to trigger flow. Status code: $statusCode\n";
}

curl_close($ch);
```

Since you will need to handle CORS errors from the client side, you could set up a simple proxy in PHP on your server. Then remove the `corsProxy` references in the code above. Example for such a solution:

```C/C++
<?php
// Define the Zenphi API endpoint and token
$zenphiApiUrl =
'https://webhook-zone1.zenphi.io/http/[Your-Flow-ID]/[Your-Trigger-ID]';
$zenphiToken = '[Token]'; // Replace with your actual token

// Add CORS headers to allow cross-origin requests
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Methods: GET, POST, OPTIONS');
header('Access-Control-Allow-Headers: Content-Type,
x-zenphi-httptriggertoken');

// Handle preflight (OPTIONS) requests
if ($_SERVER['REQUEST_METHOD'] === 'OPTIONS') {
    http_response_code(200); // Return 200 OK for preflight requests
    exit();
}

// Initialize cURL
$ch = curl_init();

// Pass the token in the headers and forward the request body
$headers = [
  'Content-Type: application/json',
  'x-zenphi-httptriggertoken: ' . $zenphiToken,
];
curl_setopt($ch, CURLOPT_URL, $zenphiApiUrl);
curl_setopt($ch, CURLOPT_POST, true);
curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);

// Forward the incoming POST data as JSON
$postData = file_get_contents('php://input');
curl_setopt($ch, CURLOPT_POSTFIELDS, $postData);

// Set common cURL options
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_HEADER, false);

// Execute the cURL request
$response = curl_exec($ch);
$statusCode = curl_getinfo($ch, CURLINFO_HTTP_CODE);

// Close cURL session
curl_close($ch);
```

```
// Return the response from Zenphi API with proper CORS headers
http_response_code($statusCode);
header('Content-Type: application/json');
echo $response;
?>
```

So what is this CORS error, and why do you have to handle it?

A **CORS (Cross-Origin Resource Sharing) error** occurs when a web application running in a browser attempts to make a request to a resource on a different domain, protocol, or port, and the server hosting the resource does not explicitly allow such requests. This is due to the browser's enforcement of the **Same-Origin Policy (SOP)**, which restricts cross-origin requests for security reasons. For example, if JavaScript code from `https://yourdomain.com` tries to fetch data from `https://zenphi.io`, the browser will block the request unless `zenphi.io` includes specific CORS headers in its response, such as `Access-Control-Allow-Origin`, indicating that it permits access from `yourdomain.com`

CORS errors are typically caused by missing server-side headers. Common scenarios include the absence of the `Access-Control-Allow-Origin` header, unsupported HTTP methods in the `Access-Control-Allow-Methods` header, or mismatched headers between the client and server during preflight requests. These errors often manifest in browser consoles with messages like "*No 'Access-Control-Allow-Origin' header is present*" or "*CORS policy does not allow access from this origin.*" To resolve these errors, developers must configure the server to include appropriate CORS headers or use proxy servers to bypass restrictions when dealing with third-party APIs.

In summary, the HTTP Trigger provides a flexible way to initiate Zenphi flows from external systems by sending HTTP requests to a designated endpoint.

Mikael Klambro
Egoiste Zenphi Application Consultant