APIGatewayProxyEventV2

August 17, 2026

APIGatewayProxyEventV2

Kind: Interface

Source: `src/adapter/aws-lambda/handler.ts`

Part of: Adapter

`APIGatewayProxyEventV2` represents an API Gateway HTTP API request received by the AWS Lambda adapter. It carries route details, headers, cookies, request body data, encoding state, and API Gateway request context for request conversion and handling.

Properties

Property	Type
`version`	`string`
`routeKey`	`string`
`headers`	`Record<string, string`
`multiValueHeaders`	`undefined`
`cookies`	`string[]`
`rawPath`	`string`
`rawQueryString`	`string`
`body`	`string
`isBase64Encoded`	`boolean`
`requestContext`	`ApiGatewayRequestContextV2`
`queryStringParameters`	`{ [name: string]: string
`pathParameters`	`{ [name: string]: string
`stageVariables`	`{ [name: string]: string

Diagram

```
graph LR
  Gateway[API Gateway Request] --> Event[Proxy Event]
  Event --> Route[Route and Path]
  Event --> Headers[Headers and Cookies]
  Event --> Payload[Body and Encoding]
  Event --> Context[Request Context]
  Event --> Adapter[AWS Lambda Adapter]
```

Usage

```
import type { APIGatewayProxyEventV2 } from "./handler";

function readRequest(event: APIGatewayProxyEventV2) {
  const body = event.isBase64Encoded
    ? Buffer.from(event.body ?? "", "base64")
    : event.body;

  return {
    path: event.rawPath,
    query: event.rawQueryString,
    headers: event.headers,
    cookies: event.cookies,
    body,
    context: event.requestContext,
  };
}
```

AI Coding Instructions

- Treat `body` as nullable before parsing or decoding it.
- Decode `body` only when `isBase64Encoded` is true.
- Read header values defensively because each header value may be undefined.
- Keep `multiValueHeaders` undefined when creating events for this adapter.
- Pass `requestContext` through when request metadata is needed downstream.

How it works

`APIGatewayProxyEventV2` is an exported TypeScript interface for the v2 Lambda event variant handled by this AWS Lambda adapter. It is included in the `LambdaEvent` union, alongside API Gateway v1, ALB, and Lattice event shapes. `src/adapter/aws-lambda/handler.ts:23-27`

Its declared fields are:

- Required event metadata: `version`, `routeKey`, `rawPath`, `rawQueryString`, `body`, and `isBase64Encoded`. `src/adapter/aws-lambda/handler.ts:41-50`
- `headers` as a map whose values can be strings or `undefined`; it explicitly declares `multiValueHeaders?: undefined`. `src/adapter/aws-lambda/handler.ts:44-45`
- Optional `cookies`, query parameters, path parameters, and stage variables. `src/adapter/aws-lambda/handler.ts:46-60`
- A required `requestContext` of type `ApiGatewayRequestContextV2`. `src/adapter/aws-lambda/handler.ts:51-51` That context includes a required `domainName` and an `http` object containing the request method, path, protocol, source IP, and user agent. `src/adapter/aws-lambda/types.ts:128-147`

At runtime, the adapter classifies an event as this type only when it has both a top-level `rawPath` property and an `http` property in `requestContext`; the code notes that `rawPath` alone can also occur on v1 events behind a custom-domain base-path mapping. `src/adapter/aws-lambda/handler.ts:646-651`

For an event selected this way, `EventV2Processor` converts it to a Fetch `Request`:

- It takes the path from `rawPath`, the query string from `rawQueryString`, and the HTTP method from `requestContext.http.method`. `src/adapter/aws-lambda/handler.ts:404-415`
- It constructs the request URL with the event's `requestContext.domainName`; if the raw query string is nonempty, it appends it after `?`. `src/adapter/aws-lambda/handler.ts:318-324`
- It joins a `cookies` array with `"; "` into a `Cookie` request header when the field is an array. `src/adapter/aws-lambda/handler.ts:417-421`
- It copies only truthy entries from `headers` into a `Headers` object. `src/adapter/aws-lambda/handler.ts:427-438`
- When `body` is truthy, it decodes it as base64 if `isBase64Encoded` is true; otherwise it UTF-8 encodes the string. It assigns that byte sequence as the request body and sets `content-length` to its byte length. `src/adapter/aws-lambda/handler.ts:333-339`

`handle()` obtains the matching processor, builds this request, passes it to `app.fetch()` with the original event, request context, and optional Lambda context, then converts the Fetch response back to a Lambda result. `src/adapter/aws-lambda/handler.ts:252-274` If request construction throws a `TypeError`, it logs the error and returns a 400 `"Invalid request"` result; other construction errors produce a logged 500 `"Internal Server Error"` result. `src/adapter/aws-lambda/handler.ts:255-266`

For this event variant, response `Set-Cookie` values are emitted in the result's `cookies` array rather than response headers. `src/adapter/aws-lambda/handler.ts:388-400` `src/adapter/aws-lambda/handler.ts:423-425`