LatticeProxyEventV2

August 18, 2026

LatticeProxyEventV2

Kind: Interface

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

Part of: Adapter

`LatticeProxyEventV2` represents an incoming request passed from AWS VPC Lattice to the Lambda adapter. It stores request metadata, multi-value headers and query parameters, an optional body, encoding state, and the associated Lattice request context.

Properties

Property	Type
`version`	`string`
`path`	`string`
`method`	`string`
`headers`	`Record<string, string[]
`queryStringParameters`	`Record<string, string[]
`body`	`string
`isBase64Encoded`	`boolean`
`requestContext`	`LatticeRequestContextV2`

Diagram

```
graph LR
  Request[AWS VPC Lattice request] --> Event[LatticeProxyEventV2]
  Event --> Route[path and method]
  Event --> Metadata[headers and query parameters]
```

```
  Event --> Payload[body and isBase64Encoded]
  Event --> Context[requestContext]
  Event --> Handler[Lambda handler]
```

Usage

```ts
import type { LatticeProxyEventV2 } from "./handler";

export function readRequest(event: LatticeProxyEventV2) {
  const contentType = event.headers["content-type"]?.find(Boolean);
  const requestId = event.requestContext.requestId;

  const body = event.body
    ? event.isBase64Encoded
      ? Buffer.from(event.body, "base64").toString("utf8")
      : event.body
    : null;

  return {
    requestId,
    method: event.method,
    path: event.path,
    contentType,
    query: event.queryStringParameters,
    body,
  };
}
```

AI Coding Instructions

- Preserve header and query parameter values as string arrays because a request can include repeated values.
- Check for a `null` body before reading or parsing it.
- Decode `body` as base64 only when `isBase64Encoded` is `true`.
- Pass `requestContext` through to logging, tracing, and authorization code rather than rebuilding request metadata.

How it works

`LatticeProxyEventV2` is an exported TypeScript interface for one AWS Lambda event variant. It is included in the `LambdaEvent` union used by the AWS Lambda adapter. [src/adapter/aws-lambda/handler.ts:23-38]

Its declared fields are:

- `version`, `path`, and `method` strings. [src/adapter/aws-lambda/handler.ts:29-32]
- `headers` and `queryStringParameters` maps whose values are `string[]` or `undefined`. [src/adapter/aws-lambda/handler.ts:33-34]
- A nullable string `body`, an `isBase64Encoded` flag, and a `requestContext` of type `LatticeRequestContextV2`. [src/adapter/aws-lambda/handler.ts:35-37]
- The referenced request context contains service-network, service, and target-group ARNs; region and epoch time strings; plus an `identity` object with optional source-VPC, principal, session, and X.509-related properties. [src/adapter/aws-lambda/types.ts:155-173]

When `getProcessor` receives an event whose `requestContext` has its own `serviceArn` property, it selects `LatticeV2Processor` for that event. ALB and API Gateway v2 detection run before this check. [src/adapter/aws-lambda/handler.ts:625-637] [src/adapter/aws-lambda/handler.ts:639-657]

Through that processor, the adapter converts the event into a Fetch `Request` as follows:

- The request path comes from `event.path`, and the HTTP method comes from `event.method`. [src/adapter/aws-lambda/handler.ts:584-591]
- The processor returns an empty query string; therefore, `queryStringParameters` is not read when constructing the request URL. [src/adapter/aws-lambda/handler.ts:593-595] [src/adapter/aws-lambda/handler.ts:318-324]
- Each defined header array is appended to a `Headers` object once per value. Header values containing non-ASCII characters are URI-encoded before being appended. [src/adapter/aws-lambda/handler.ts:597-609] [src/adapter/aws-lambda/handler.ts:13-21]
- The request domain is taken from `requestContext.domainName` when that property exists; otherwise the adapter reads the first `host` value from `headers` and, if needed, from `multiValueHeaders`. The declared Lattice context has no `domainName` field, and `LatticeProxyEventV2` has no `multiValueHeaders` field. [src/adapter/aws-lambda/handler.ts:301-316] [src/adapter/aws-lambda/handler.ts:29-38] [src/adapter/aws-lambda/types.ts:155-173]
- If `body` is truthy, it is base64-decoded when `isBase64Encoded` is true; otherwise it is UTF-8 encoded. The adapter sets `content-length` to the resulting byte length. [src/adapter/aws-lambda/handler.ts:333-339]
- A `null` or empty-string body is not assigned to the constructed request because the body branch tests `if (event.body)`. [src/adapter/aws-lambda/handler.ts:333-339]

`handle()` passes the constructed request, the original event, its request context, and the optional Lambda context to `app.fetch`. [src/adapter/aws-lambda/handler.ts:252-274] If request construction throws a `TypeError`, it logs the error and returns a 400 `"Invalid request"` response; other thrown errors produce a logged 500 `"Internal Server Error"` response. [src/adapter/aws-lambda/handler.ts:255-266]

For this event type, response conversion starts with a single-value `headers` object rather than `multiValueHeaders`, because `LatticeProxyEventV2` does not declare `multiValueHeaders`. [src/adapter/aws-lambda/handler.ts:361-372] Response bodies are base64-encoded when the configured or default content-type test marks the response binary, or when `content-encoding` is present and is not `identity`; otherwise the body is returned as text. [src/adapter/aws-lambda/handler.ts:349-360] [src/adapter/aws-lambda/handler.ts:666-674] Response headers are copied into that result header object. [src/adapter/aws-lambda/handler.ts:374-385] If the response has `set-cookie` headers, this processor writes the extracted cookie array to `result.headers['set-cookie']` and removes `set-cookie` from the response headers before the remaining headers are copied. [src/adapter/aws-lambda/handler.ts:388-400] [src/adapter/aws-lambda/handler.ts:615-620]

The interface itself contains no runtime validation; runtime classification only checks for `requestContext.serviceArn`, while later request construction reads the declared fields directly. [src/adapter/aws-lambda/handler.ts:29-38] [src/adapter/aws-lambda/handler.ts:653-657] [src/adapter/aws-lambda/handler.ts:584-609]