Design Approach: Lambda Layer vs. Custom-Built Adapter

This document explains our design decision to use the AWS Lambda Web Adapter with a sanitization wrapper instead of building a custom adapter from scratch.

Background

Our initial approach was to create a custom adapter that would handle HTTP/2 header sanitization. However, we encountered several challenges that led us to pivot to modifying the existing AWS Lambda Web Adapter instead.

The "Cannot Execute Binary File" Error

When attempting to deploy our initial custom-built adapter, we encountered the following error:

```
"Cannot execute binary file"
```

This error indicated that:

1. Our binary was not compatible with the Lambda execution environment
2. We likely had architecture mismatch issues (Lambda runs on Amazon Linux)

Approach Comparison

Aspect	Custom-Built Adapter	Modified AWS Lambda Web Adapter
Development Effort	High (full implementation)	Low (add sanitization only)
Reliability	Untested in production	Built on proven AWS code
Compatibility	Potential issues	Guaranteed Lambda compatibility
Maintenance	Ongoing responsibility	Core maintained by AWS
Security	Need full security review	Inherits AWS security posture
Performance	Unknown	Well-optimized for Lambda

Chosen Solution: Lambda Layer + Sanitization

We chose to implement a thin sanitization layer on top of the AWS Lambda Web Adapter for these reasons:

1. Reliability

The AWS Lambda Web Adapter is:

- Developed and maintained by AWS
- Well-tested in production environments
- Designed for the Lambda execution environment
- Optimized for performance

2. Implementation Simplicity

Our solution:

- Only adds header sanitization logic
- Doesn't need to reinvent request/response handling
- Integrates cleanly with the existing adapter
- Requires minimal code changes

3. Deployment Simplicity

Using the Lambda Layer approach:

- Works with any Lambda runtime
- Requires no application code changes
- Is easy to update or rollback
- Integrates with existing CI/CD pipelines

4. Maintainability

This approach is more maintainable because:

- We only maintain the sanitization logic
- Core adapter functionality is maintained by AWS
- Upgrades to the adapter can be incorporated easily
- Less code means fewer potential bugs

The Python Wrapper Alternative

For cases where adding a Lambda Layer isn't desirable, we also provide a Python wrapper solution:

```python
def sanitize_http2_headers(response):
    """Sanitize HTTP/2 disallowed headers"""
    disallowed_headers = [
        "connection", "keep-alive", "proxy-connection",
        "transfer-encoding", "upgrade"
    ]

    if "headers" in response and response["headers"]:
        sanitized_headers = {}
        for header_name, header_value in response["headers"].items():
            if header_name.lower() not in disallowed_headers:
                sanitized_headers[header_name] = header_value
        response["headers"] = sanitized_headers

    return response
```

This wrapper:

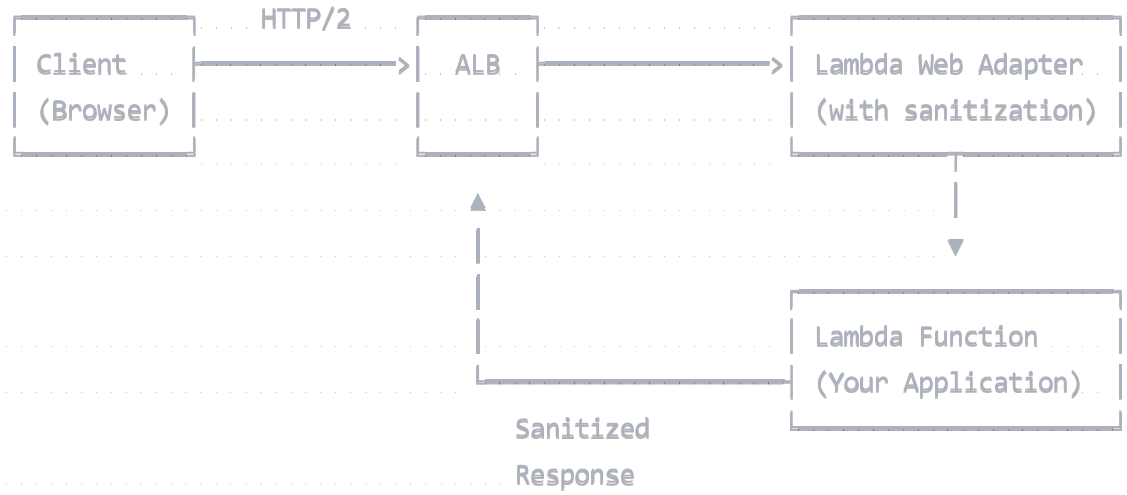
- Is even simpler to implement
- Has no dependencies
- Works with any Python Lambda function
- Adds negligible performance overhead

Technical Implementation Details

Lambda Layer Approach

The AWS Lambda Web Adapter acts as a proxy between the ALB and your Lambda code. Our modification:

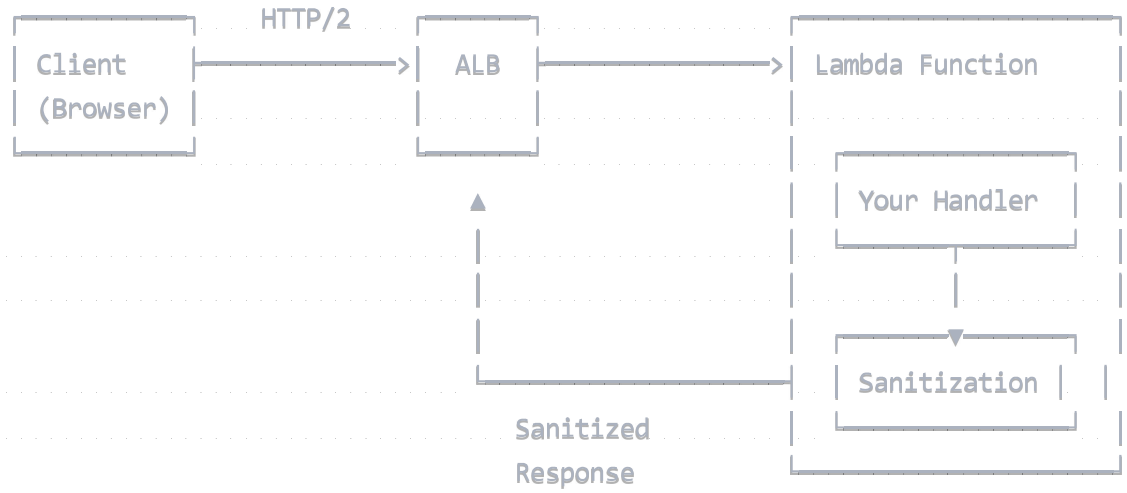
1. Intercepts responses before they're sent back to the ALB
2. Sanitizes HTTP/2-incompatible headers
3. Forwards the cleaned response to the ALB



Python Wrapper Approach

The Python wrapper is a simple function that:

1. Takes the response object from your Lambda handler
2. Filters out HTTP/2-incompatible headers
3. Returns the sanitized response



Conclusion

Building a custom adapter from scratch would have been a significant undertaking with many potential pitfalls. By leveraging the existing AWS Lambda Web Adapter and adding targeted sanitization, we've created a solution that:

1. Solves the HTTP/2 header problem effectively
2. Minimizes development and maintenance effort
3. Leverages AWS's proven code for reliability

4. Is simple to deploy and use
5. Works with any runtime and framework

This approach exemplifies the principle of "standing on the shoulders of giants" - building upon established work to create a targeted solution that addresses a specific need.