

Edge Computing and Functions as a Service (FaaS)

Implementing header-based API Gateway versioning with Content Delivery Network (CDN)

In this experiment we use Lambda@Edge feature of Amazon CloudFront to implement a header-based API versioning solution for Amazon API Gateway.

Amazon API Gateway is a fully managed service that makes it easier for developers to create, publish, maintain, monitor, and secure APIs at any scale. Amazon CloudFront is a global content delivery network (CDN) service built for high-speed, low-latency performance, security, and developer ease-of-use. Lambda@Edge is a feature of Amazon CloudFront, a compute service that lets you run functions that customize the content that CloudFront delivers.

This experiment uses the AWS SAM CLI to build, deploy, and test the solution on AWS. The AWS Serverless Application Model (AWS SAM) is an open-source framework that you can use to build serverless applications on AWS. The AWS SAM CLI lets you locally build, test, and debug your applications defined by AWS SAM templates. You can also use the AWS SAM CLI to deploy your applications to AWS, or create secure continuous integration and deployment (CI/CD) pipelines.

After an API becomes publicly available, it is used by customers. As a service evolves, its contract also evolves to reflect new changes and capabilities. It's safe to evolve a public API by adding new features but it's not safe to change or remove existing features.

Any breaking changes may impact consumer's applications and break them at runtime. API versioning is important to avoid breaking backward compatibility and breaking a contract. You need a clear strategy for API versioning to help consumers adopt them.

Versioning APIs

Two of the most used API versioning strategies are URI versioning and header-based versioning.

URI versioning

This strategy is the most straightforward and the most commonly used approach. In this type of versioning, versions are explicitly defined as part of API URIs. These example URLs show how domain name, path, or query string parameters can be used to specify a version:

`https://api.example.com/v1/myservice`

```
https://apiv1.example.com/myservice
https://api.example.com/myservice?v=1
```

To deploy an API in API Gateway, the deployment is associated with a stage. A stage is a logical reference to a lifecycle state of your API (for example, dev, prod, beta, v2). As your API evolves, you can continue to deploy it to different stages as different versions of the API.

Header-based versioning

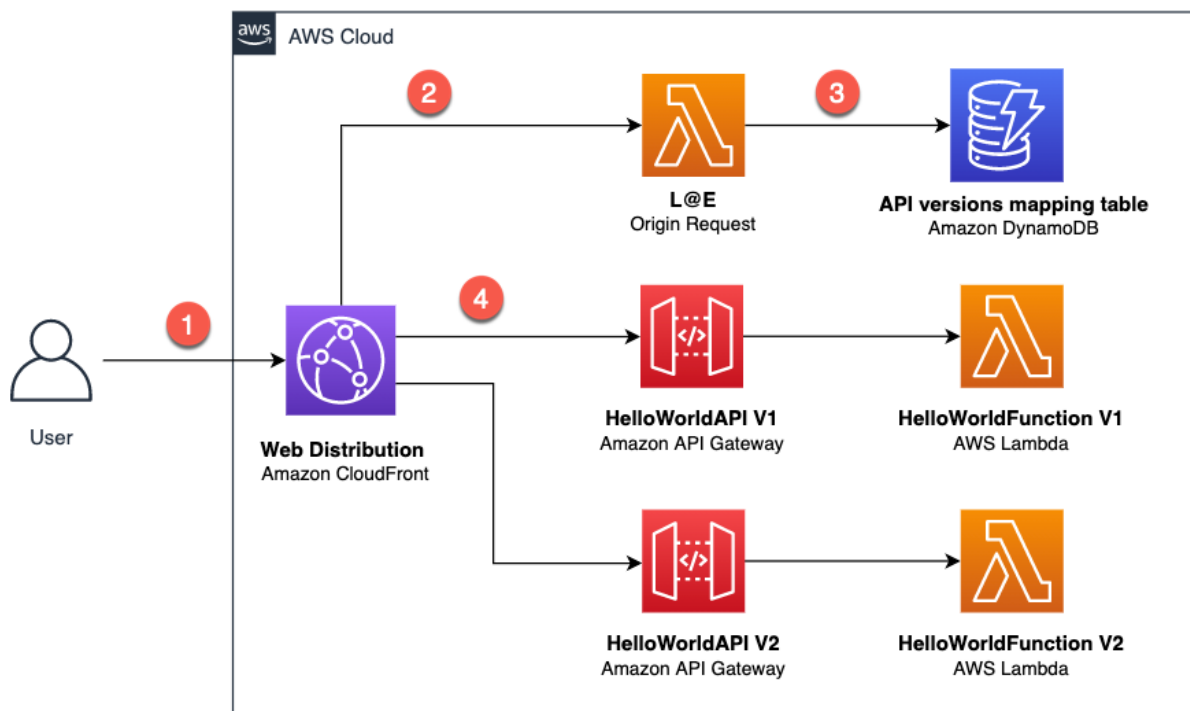
This strategy is another commonly used versioning approach. It uses HTTP headers to specify the desired version. It uses the “**Accept**” header for content negotiation or uses a custom header (for example, “**APIVER**” to indicate a version):

```
Accept:application/vnd.example.v1+json
APIVER:v1
```

This approach allows you to preserve URIs between versions. As a result, you have a cleaner and more understandable set of URIs. It is also easier to add versioning after design. However, you may need to deal with complexity of returning different versions of your resources.

Overview of solution

The target architecture for the solution uses Lambda@Edge. It dynamically routes a request to the relevant API version, based on the provided header:



In this experiment Edge Computing architecture:

1. The user sends a request with a relevant header, which can be either “**Accept**” or another custom header.
2. This request reaches the CloudFront distribution and triggers the **Lambda@Edge Origin Request**.
3. The **Lambda@Edge** function uses the provided header value and fetches data from an Amazon DynamoDB table. This table contains mappings for API versions. The function then modifies the **Origin** and the **Host** header of the request and returns it back to CloudFront.
4. CloudFront sends the request to the relevant Amazon API Gateway URL.

In the next sections, I walk you through setting up the development environment and deploying and testing this solution.

Setting up the development environment

To deploy this solution on AWS, you use the AWS Cloud9 development environment.

1. Go to the AWS Cloud9 web console. In the Region dropdown, make sure you’re using **N. Virginia (us-east-1)** Region.
2. Select **Create environment**.
3. On Step 1 - **Name environment**, enter a name **edge-computing-studentXX** for the environment, and choose **Next step**.
4. On Step 2 - **Configure settings**, keep the existing environment settings.

Step 1
Name environment

Step 2
Configure settings

Step 3
Review

Configure settings

Environment settings

Environment type [Info](#)

Run your environment in a new EC2 instance or an existing server. With EC2 instances, you can connect directly through Secure Shell (SSH) or connect via AWS Systems Manager (without opening inbound ports).

☒ Create a new EC2 instance for environment (direct access)

Launch a new instance in this region that your environment can access directly via SSH.

☐ Create a new no-ingress EC2 instance for environment (access via Systems Manager)

Launch a new instance in this region that your environment can access through Systems Manager.

☐ Create and run in remote server (SSH connection)

Configure the secure connection to the remote server for your environment.

Instance type

☒ t2.micro (1 GiB RAM + 1 vCPU)

Free-tier eligible. Ideal for educational users and exploration.

☐ t3.small (2 GiB RAM + 2 vCPU)

Recommended for small-sized web projects.

☐ m5.large (8 GiB RAM + 2 vCPU)

Recommended for production and general-purpose development.

☐ Other instance type

Select an instance type.

t3.nano

Platform

☒ Amazon Linux 2 (recommended)

☐ Amazon Linux AMI

☐ Ubuntu Server 18.04 LTS

5. Choose **Next step**. Choose **Create environment**.

Deploying the solution

Now that the development environment is ready, you can proceed with the solution deployment. In this section, you download, build, and deploy a sample serverless application for the solution using AWS SAM.

Download our serverless application

The experiment code is available on GitHub. Clone the repository and download the sample source code to your Cloud9 IDE environment by running the following command in the Cloud9 terminal window:

```
$ git clone https://github.com/GeorgeNiece/edge-computing-api-gateway-faas.git ./api-gateway-header-based-versioning
```

This repository includes:

- **template.yaml**: Contains the AWS SAM template that defines your application's AWS resources.

- **hello-world/**: Contains the Lambda handler logic behind the API Gateway endpoints to return the hello world message.
- **edge-origin-request/**: Contains the Lambda@Edge handler logic to query the API version mapping and modify the **Origin** and the **Host** header of the request.
- **init-db/**: Contains the Lambda handler logic for a custom resource to populate sample DynamoDB table

Build your application

Run the following commands in order to first, change into the project directory, where the template.yaml file for the sample application is located then build your application:

```
$> cd ~/environment/api-gateway-header-based-versioning/
$> sam build
```

Output:

Build Succeeded

Built Artifacts : .aws-sam/build
Built Template : .aws-sam/build/template.yaml

Commands you can use next

=====

[*] Invoke Function: sam local invoke
[*] Deploy: sam deploy --guided

admin:~/environment/api-gateway-header-based-versioning (main) \$

Deploy your application

Run the following command to deploy the application in guided mode for the first time:

```
$> sam deploy --guided
```

Follow the on-screen prompts. Respond with **Enter** to accept the default options provided in the interactive experience. Respond to the interactive questions as follows:

- **Stack Name [sam-app]**: choose a stack name i.e. **"api-gateway-header-based-versioning"**
- **AWS Region [us-east-1]**: press **Enter** to accept the default value **"us-east-1"** Region
- **Parameter ApiVersionHeaderName [Accept]**: press **Enter** to accept the default value **"Accept"** header

- **Parameter SampleApiVersionMappingV1**
[application/vnd.example.v1+json]: press **Enter** to accept the default value
- **Parameter SampleApiVersionMappingV2**
[application/vnd.example.v2+json]: Press **Enter** to accept the default value
- **Confirm changes before deploy [y/N]:** N
- **Allow SAM CLI IAM role creation [Y/n]:** Y
- **HelloWorldFunctionV1 may not have authorization defined, Is this okay? [y/N]:** y
- **HelloWorldFunctionV2 may not have authorization defined, Is this okay? [y/N]:** y
- **Save arguments to configuration file [Y/n]:** Y
- **SAM configuration file [samconfig.toml]:** press **Enter** to accept the default value
- **SAM configuration environment [default]:** press **Enter** to accept the default value

```
CloudFormation outputs from deployed stack

-----
Outputs
-----
Key               CFDistribution
Description       Cloudfront Distribution Domain Name
Value             [redacted].cloudfront.net

Key               HelloWorldApiV2
Description       API Gateway endpoint URL for Prod stage for Hello World function version 2
Value             https://[redacted].execute-api.us-east-1.amazonaws.com/Prod/v2/hello/

Key               DynamoDBTableName
Description       Api Versions Mapping DynamoDB Table Name
Value             api-gateway-header-based-versioning-ApiVersionsMappingDynamoDBTable-[redacted]

Key               HelloWorldApiV1
Description       API Gateway endpoint URL for Prod stage for Hello World function version 1
Value             https://[redacted].execute-api.us-east-1.amazonaws.com/Prod/v1/hello/

Key               LambdaEdgeFunctionRouteApiVersion
Description       Lambda@Edge Route API Function ARN with Version
Value             arn:aws:lambda:us-east-1:[redacted]:function:api-gateway-header-based--LambdaEdge

-----

Successfully created/updated stack - api-gateway-header-based-versioning in us-east-1

admin:~/environment/api-gateway-header-based-versioning (main) $
```

The output shows the deployment of the AWS CloudFormation stack.

Testing the solution

This application implements all required components for the solution. It consists of two Amazon API Gateway endpoints backed by AWS Lambda functions. The deployment process also initializes the API Version Mapping DynamoDB table with the values provided earlier in the deployment process.

Run the following commands from the repository folder to see the created mappings:

```
$> STACK_NAME=$(grep stack_name ~/environment/api-gateway-header-based-versioning/samconfig.toml | awk -F\= '{gsub(/"/, "", $2); gsub(/ /, "", $2); print $2}')
```

```
$> DDB_TBL_NAME=$(aws cloudformation describe-stacks --region us-east-1 --stack-name $STACK_NAME --query 'Stacks[0].Outputs[?OutputKey==`DynamoDBTableName`].OutputValue' --output text) && echo $DDB_TBL_NAME
```

```
$> aws dynamodb scan --table-name $DDB_TBL_NAME
```

Output:

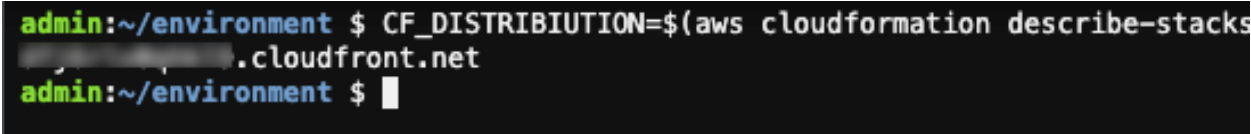
```
admin:~/environment/api-gateway-header-based-versioning (main) $ aws dynamodb scan --tab
{
  "Count": 2,
  "Items": [
    {
      "dn": {
        "S": "██████████.execute-api.us-east-1.amazonaws.com"
      },
      "hk": {
        "S": "application/vnd.example.v1+json"
      },
      "dp": {
        "S": "/Prod/v1"
      }
    },
    {
      "dn": {
        "S": "██████████.execute-api.us-east-1.amazonaws.com"
      },
      "hk": {
        "S": "application/vnd.example.v2+json"
      },
      "dp": {
        "S": "/Prod/v2"
      }
    }
  ],
  "ScannedCount": 2,
  "ConsumedCapacity": null
}
admin:~/environment/api-gateway-header-based-versioning (main) $ █
```

When a user sends a GET request to CloudFront, it routes the request to the relevant API Gateway endpoint version according to the provided header value. The Lambda function behind that API Gateway endpoint is invoked and returns a “hello world” message.

To send a request to the CloudFront distribution, which is created as part of the deployment process, first get its domain name from the deployed AWS CloudFormation stack:

```
$> CF_DISTRIBIUTION=$(aws cloudformation describe-stacks --region  
us-east-1 --stack-name $STACK_NAME --query  
'Stacks[0].Outputs[?OutputKey=='CFDistribution'].OutputValue' --  
output text) && echo $CF_DISTRIBIUTION
```

Output:



```
admin:~/environment $ CF_DISTRIBIUTION=$(aws cloudformation describe-stacks  
--region us-east-1 --stack-name $STACK_NAME --query  
'Stacks[0].Outputs[?OutputKey=='CFDistribution'].OutputValue' --  
output text) && echo $CF_DISTRIBIUTION  
admin:~/environment $
```

You can now send a GET request along with the relevant header you specified during the deployment process to the CloudFront to test the application.

Run the following command to test the application for API version one. Note that if you entered a different value other than the default value provided during the deployment process, change the --header parameter to match your inputs:

```
$> curl -i -o - --silent -X GET  
"https://{CF_DISTRIBIUTION}/hello" --header  
"Accept:application/vnd.example.v1+json" && echo
```

Output:


```

admin:~/environment/api-gateway-header-based-versioning (main) $ curl -i -o -
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 35
Connection: keep-alive
Date: [REDACTED]
x-amzn-RequestId: a871c0a6-3b25-486d-aec9-7fde655b7ca1
x-amz-apigw-id: DkCRhEP0IAMF4Tg=
X-Amzn-Trace-Id: Root=1-610b173c-161a7c861ebdb03d6c6b39bb;Sampled=0
Via: 1.1 [REDACTED].cloudfront.net (CloudFront), 1.1 1b5
X-Amz-Cf-Pop: IAD79-C3
X-Cache: Miss from cloudfront
X-Amz-Cf-Pop: IAD50-C2
X-Amz-Cf-Id: [REDACTED]

{"message":"API v.1: Hello world!"}
admin:~/environment/api-gateway-header-based-versioning (main) $ █

```

The response shows that CloudFront successfully routed the request to the API Gateway v1 endpoint as defined in the mapping Amazon DynamoDB table. API Gateway v1 endpoint received the request. The Lambda function behind the API Gateway v1 was invoked and returned a “hello world” message.

Now you can change the header value to v2 and run the command again this time to test the API version two:

```

$> curl -i -o - --silent -X GET
"https://{CF_DISTRIBUTION}/hello" --header
"Accept:application/vnd.example.v2+json" && echo

```

Output:

```

admin:~/environment/api-gateway-header-based-versioning (main) $ curl -i -o -
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 35
Connection: keep-alive
Date: [REDACTED]
x-amzn-RequestId: 9e33dc0b-4fa8-4689-a345-369103470f4c
x-amz-apigw-id: DkCkxGnEIAMFk_g=
X-Amzn-Trace-Id: Root=1-610b17b7-4b576f943a1ee5c93f165be4;Sampled=0
Via: 1.1 [REDACTED].cloudfront.net (CloudFront), 1.1 65
X-Amz-Cf-Pop: IAD79-C3
X-Cache: Miss from cloudfront
X-Amz-Cf-Pop: IAD50-C2
X-Amz-Cf-Id: [REDACTED]

{"message":"API v.2: Hello world!"}
admin:~/environment/api-gateway-header-based-versioning (main) $ █

```

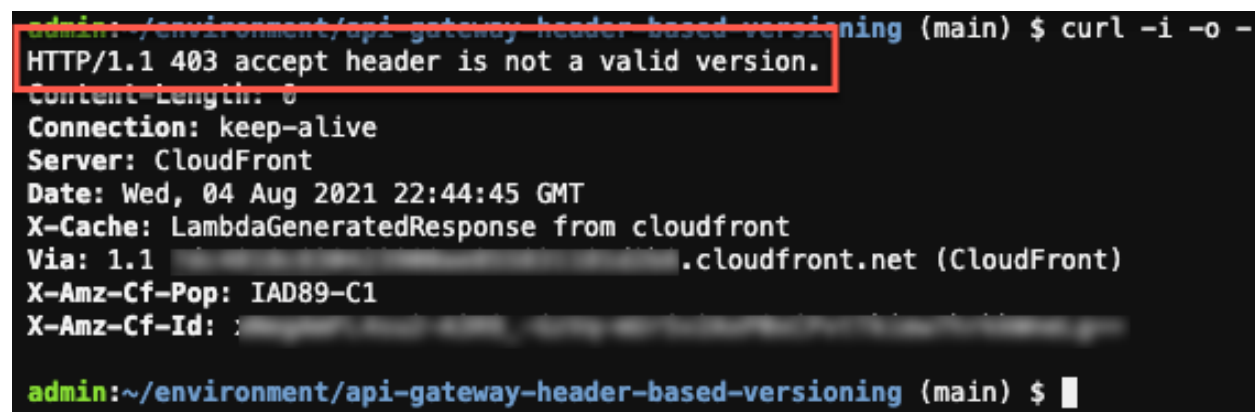
The response shows that CloudFront routed the request to the API Gateway v2 endpoint as defined in the mapping DynamoDB table. API Gateway v2 endpoint received the request. The Lambda function behind the API Gateway v2 was invoked and returned a “hello world” message.

This solution requires valid a header value on each individual request, so the application checks and raises an error if the header is missing or the header value is not valid.

You can remove the header parameter and run the command to test this scenario:

```
$> curl -i -o - --silent -X GET  
"https://${CF_DISTRIBUTION}/hello" && echo
```

Output:

A terminal window screenshot showing a curl command execution. The first line shows the prompt 'admin: ~/environment/api-gateway-header-based-versioning (main) \$' followed by the command 'curl -i -o -'. The second line shows the response 'HTTP/1.1 403 accept header is not a valid version.' which is highlighted with a red box. Subsequent lines show headers: 'Content-Length: 0', 'Connection: keep-alive', 'Server: CloudFront', 'Date: Wed, 04 Aug 2021 22:44:45 GMT', 'X-Cache: LambdaGeneratedResponse from cloudfront', 'Via: 1.1 [REDACTED].cloudfront.net (CloudFront)', 'X-Amz-Cf-Pop: IAD89-C1', and 'X-Amz-Cf-Id: [REDACTED]'. The final line shows the prompt 'admin:~/environment/api-gateway-header-based-versioning (main) \$'.

The response shows that Lambda@Edge validated the request and raised an error to inform us that the request did not have a valid header.

Mitigating latency

In this solution, Lambda@Edge reads the API version mappings data from the DynamoDB table. Accessing external data at the edge can cause additional latency to the request. In order to mitigate the latency, solution uses following methods:

1. **Cache data in Lambda@Edge memory:** As data is unlikely to change across many Lambda@Edge invocations, Lambda@Edge caches API version mappings data in the memory for a certain period of time. It reduces latency by avoiding an external network call for each individual request.
2. **Use Amazon DynamoDB global table:** It brings data closer to the CloudFront distribution and reduces external network call latency.

Cleaning up

To clean up the resources provisioned as part of the solution:

1. Go to the AWS CloudFormation console. Choose the “**api-gateway-header-based-versioning**” stack then choose **Delete**.

2. Go to the [AWS Cloud9 web console](#). Select the environment you created then choose **Delete**.

Conclusion

Header-based API versioning is a commonly used versioning strategy. This post shows how to use CloudFront to implement a header-based API versioning solution for API Gateway. It uses the AWS SAM CLI to build and deploy a sample serverless application to test the solution in the AWS Cloud.

To learn more about API Gateway, visit the [API Gateway developer guide documentation](#), and for CloudFront, refer to [Amazon CloudFront developer guide documentation](#).

For more serverless learning resources, visit [Serverless Land](#).