cache as much as you can

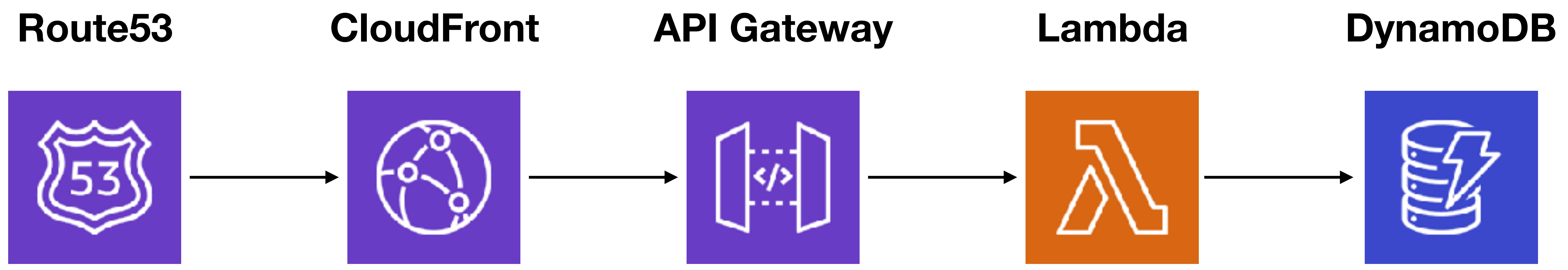
cache as much as you can

improve response time

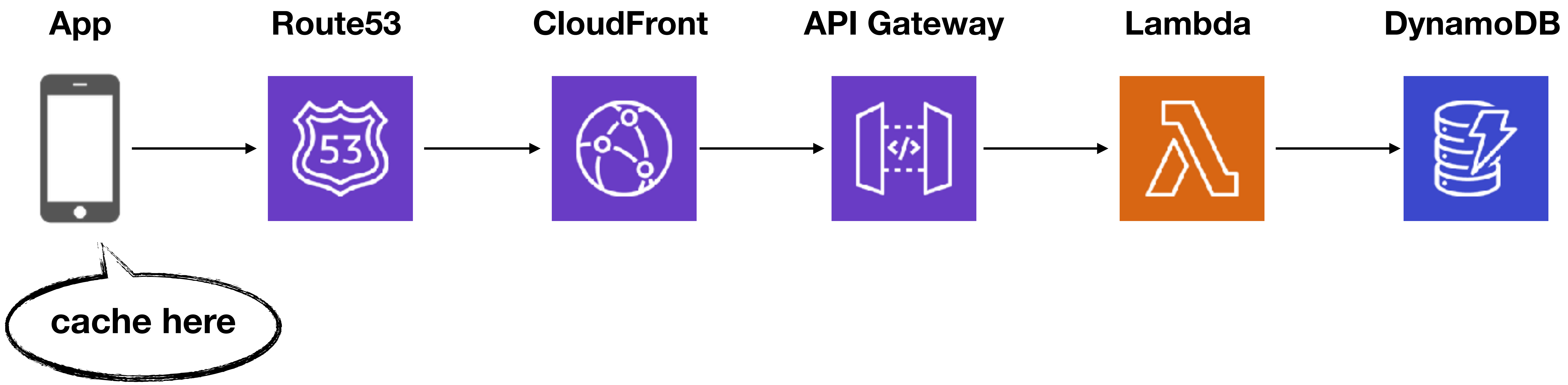
cache as much as you can

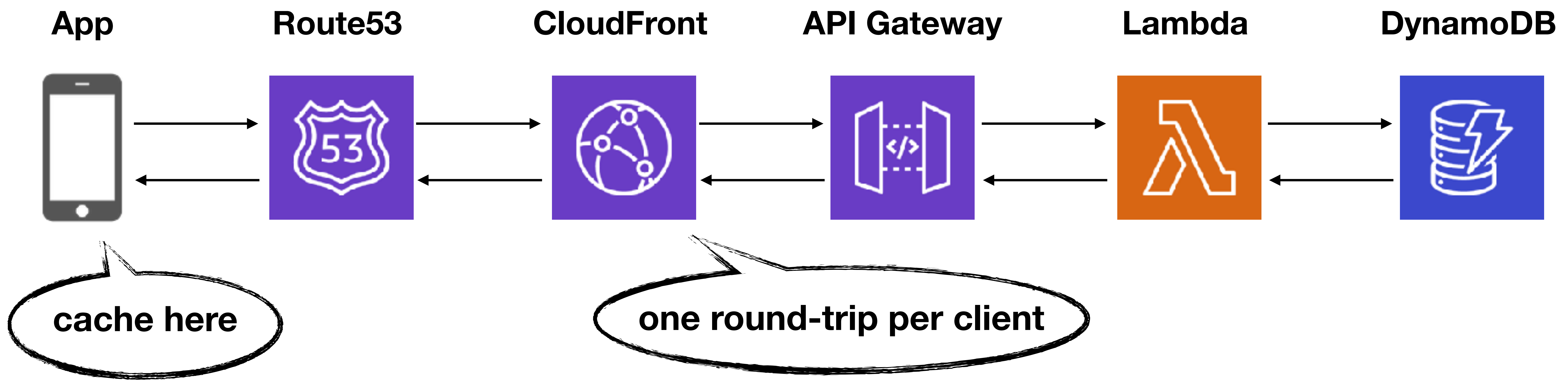
improve response time

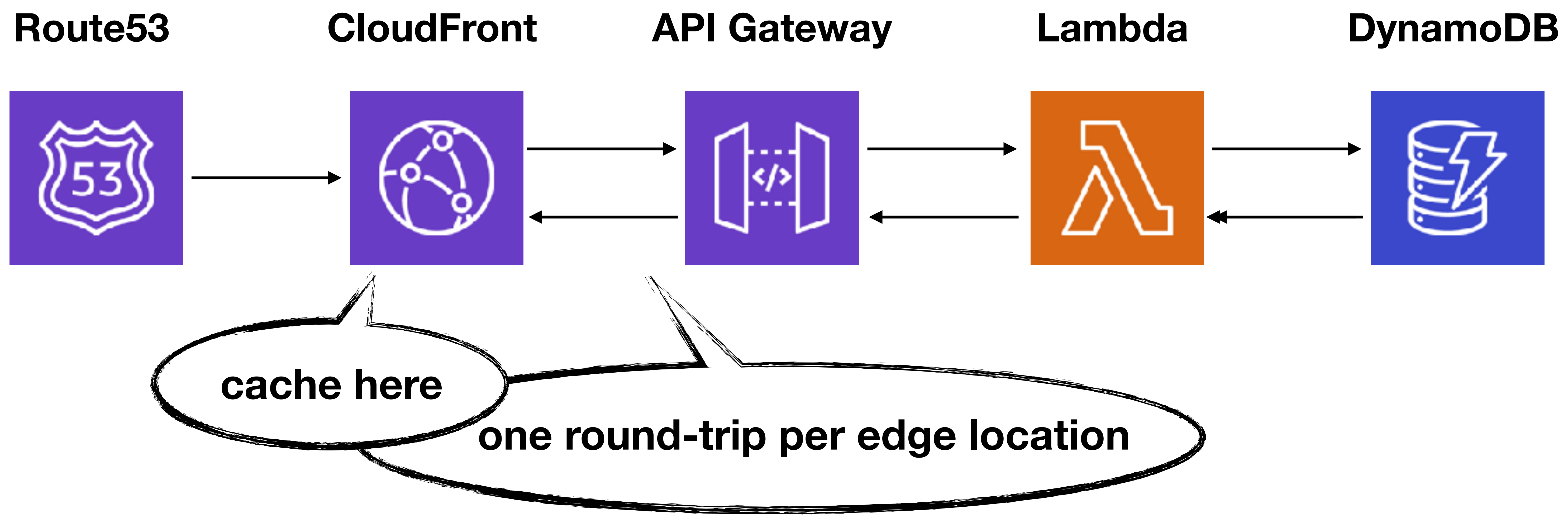
saves $$$

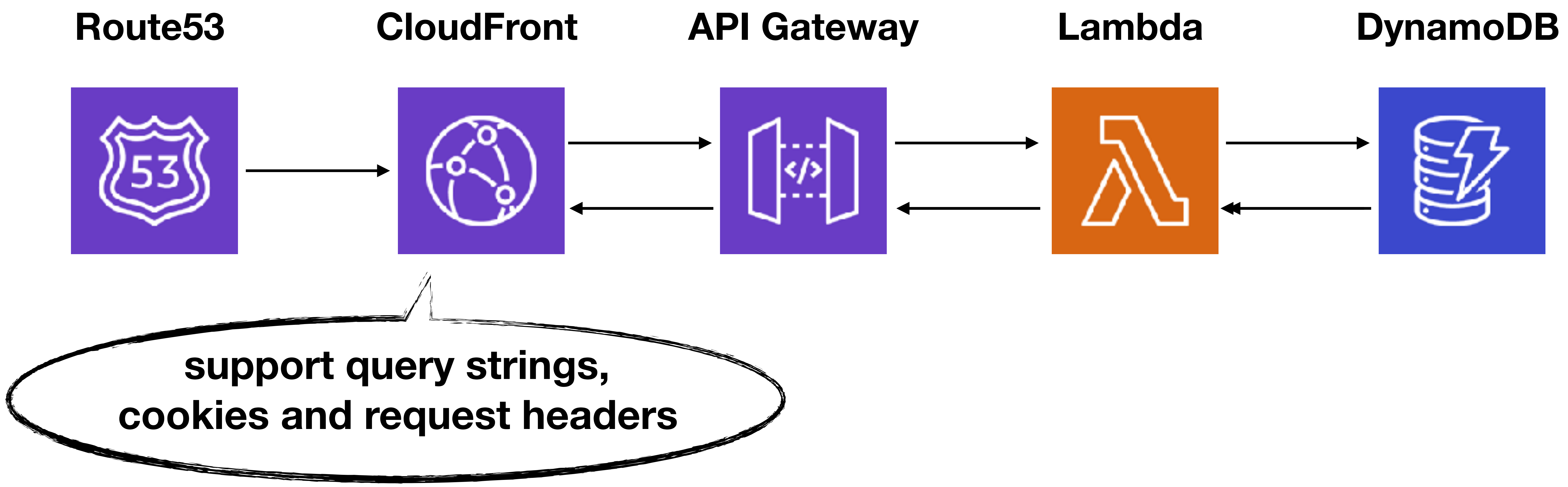

Route53
CloudFront
API Gateway
Lambda
DynamoDB

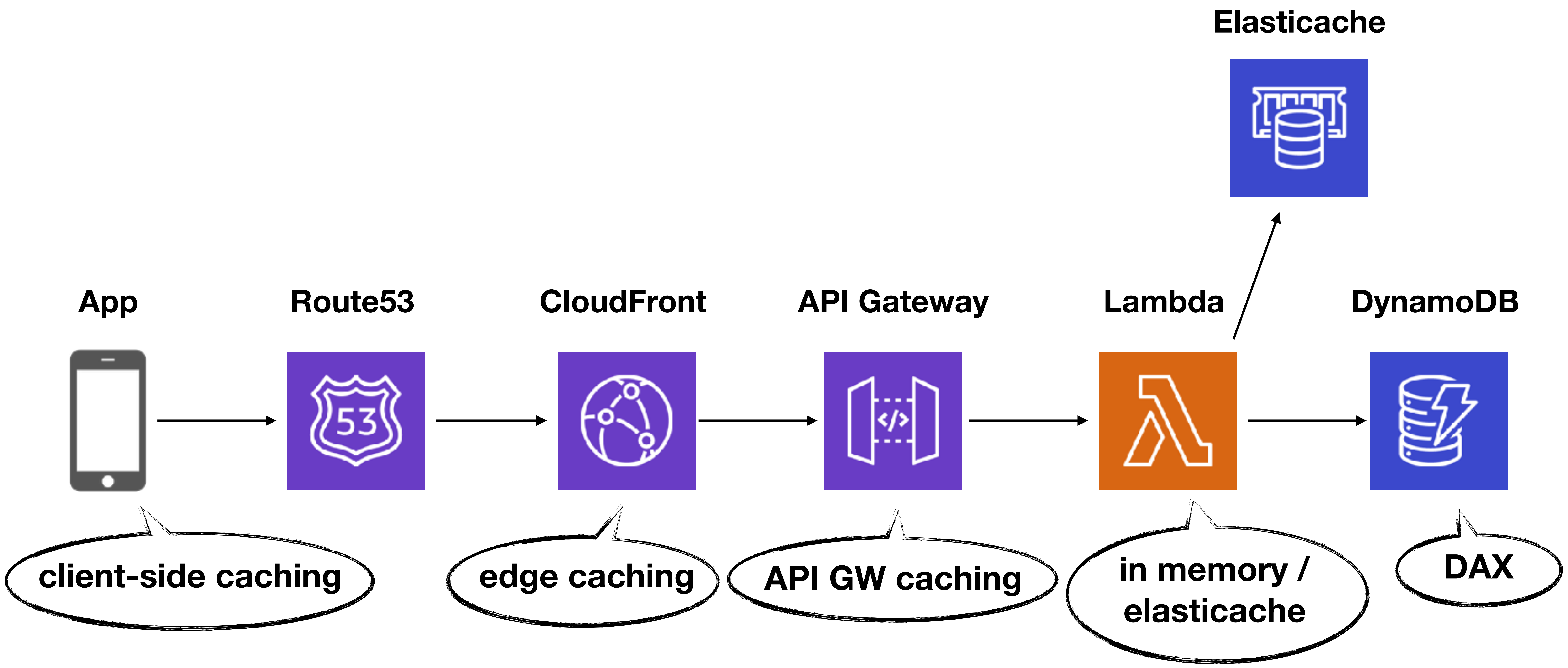
cache as close to the end user as possible


App
Route53
CloudFront
API Gateway
Lambda
DynamoDB
cache here

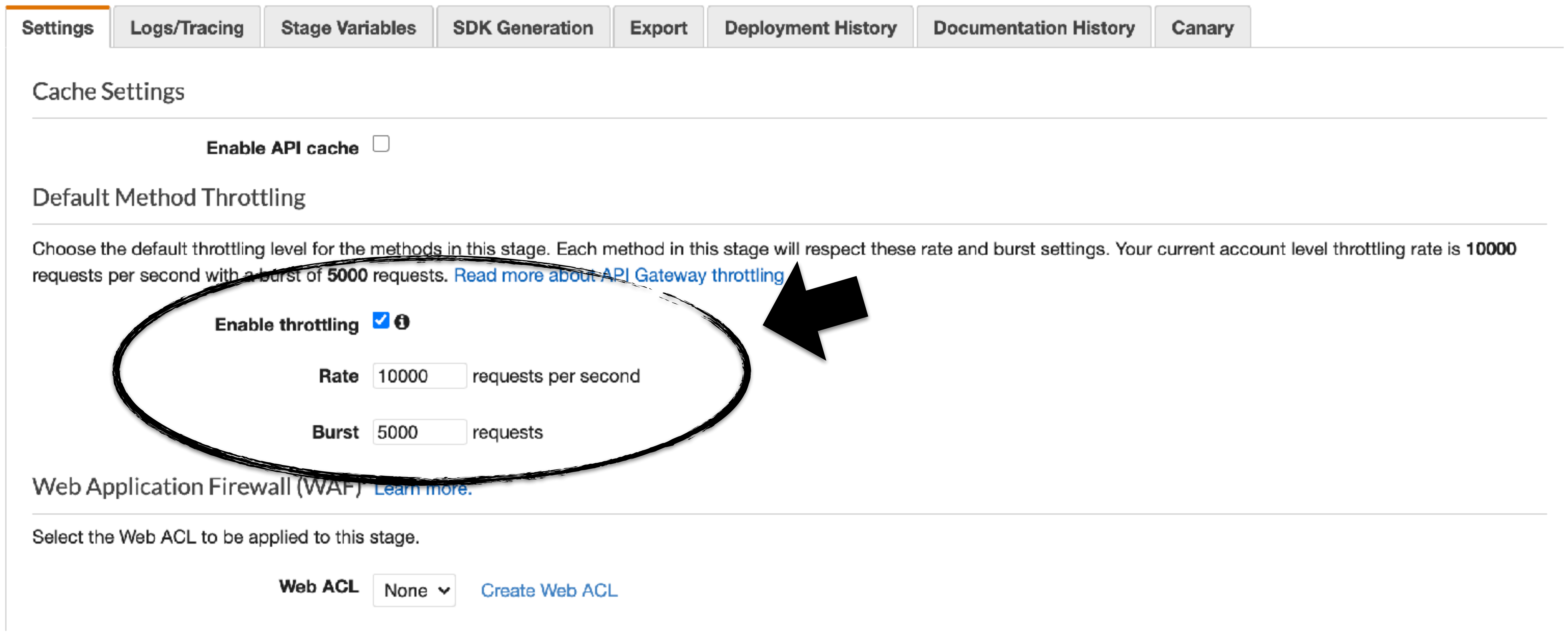

App
Route53
CloudFront
API Gateway
Lambda
DynamoDB
cache here
one round-trip per client


Route53
CloudFront
API Gateway
Lambda
DynamoDB
cache here
one round-trip per edge location


Route53
CloudFront
API Gateway
Lambda
DynamoDB
53
support query strings,
cookies and request headers



Learn more about these options at bit.ly/2SnOiRQ review default throttling limits



Settings | Logs/Tracing | Stage Variables | SDK Generation | Export | Deployment History | Documentation History | Canary

Cache Settings

Enable API cache ☐

Default Method Throttling

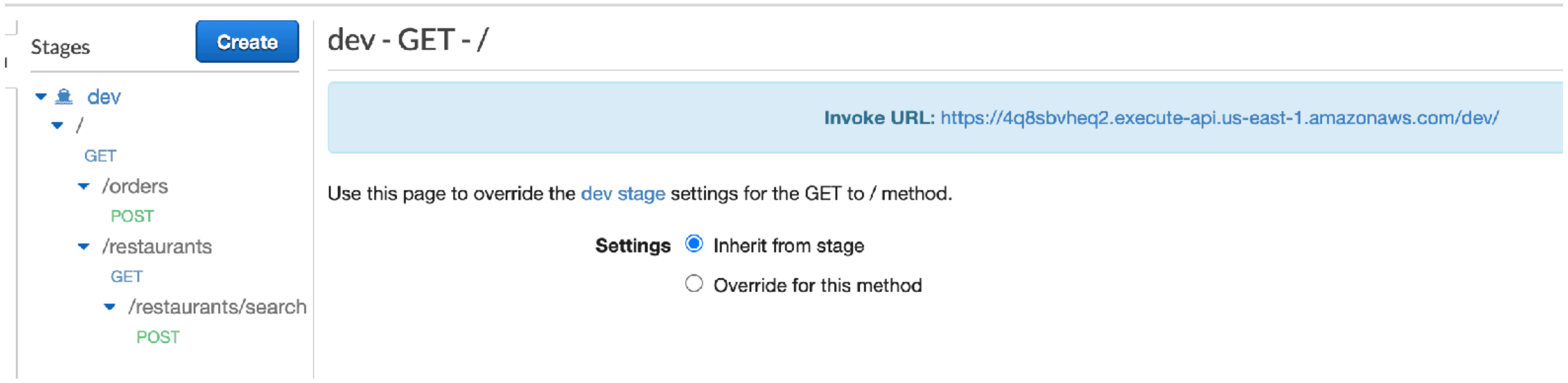
Choose the default throttling level for the methods in this stage. Each method in this stage will respect these rate and burst settings. Your current account level throttling rate is **10000** requests per second with a burst of **5000** requests. Read more about API Gateway throttling

Enable throttling	✓
Rate	10000 requests per second
Burst	5000 requests

Web Application Firewall (WAF) Learn more.

Select the Web ACL to be applied to this stage.

Web ACL None Create Web ACL


Stages
Create

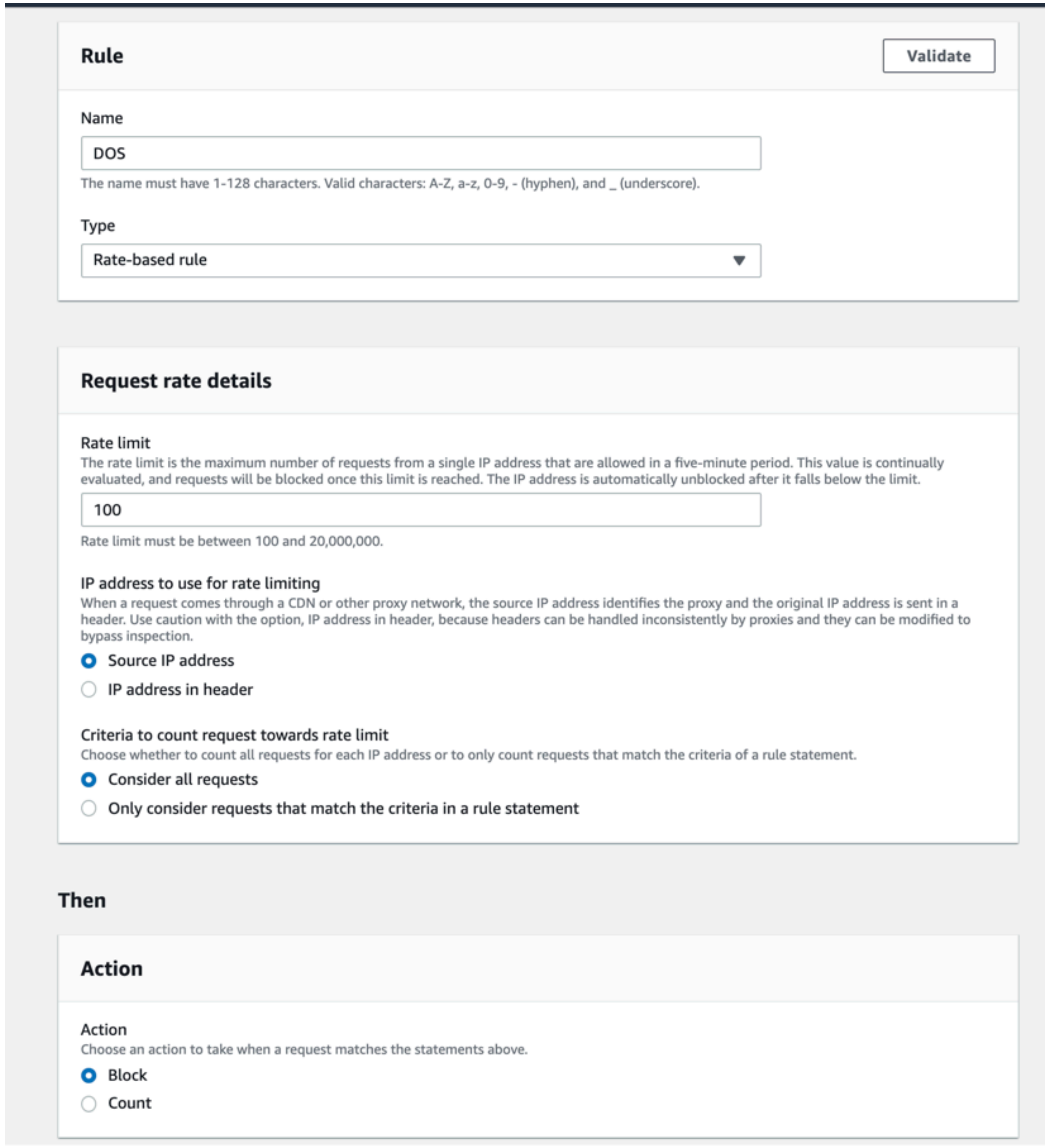
dev
/
GET
/orders
POST
/restaurants
GET
/restaurants/search
POST

dev - GET - /

Invoke URL: https://4q8sbvheq2.execute-api.us-east-1.amazonaws.com/dev/

Use this page to override the dev stage settings for the GET to / method.

Settings [✓ Inherit from stage]
Override for this method



Rule

Name

DOS

The name must have 1-128 characters. Valid characters: A-Z, a-z, 0-9, - (hyphen), and _ (underscore).

Type

Rate-based rule

Request rate details

Rate limit

The rate limit is the maximum number of requests from a single IP address that are allowed in a five-minute period. This value is continually evaluated, and requests will be blocked once this limit is reached. The IP address is automatically unblocked after it falls below the limit.

100

Rate limit must be between 100 and 20,000,000.

IP address to use for rate limiting

When a request comes through a CDN or other proxy network, the source IP address identifies the proxy and the original IP address is sent in a header. Use caution with the option, IP address in header, because headers can be handled inconsistently by proxies and they can be modified to bypass inspection.

[✓ Source IP address]
IP address in header

Criteria to count request towards rate limit

Choose whether to count all requests for each IP address or to only count requests that match the criteria of a rule statement.

[✓ Consider all requests]
Only consider requests that match the criteria in a rule statement

Then

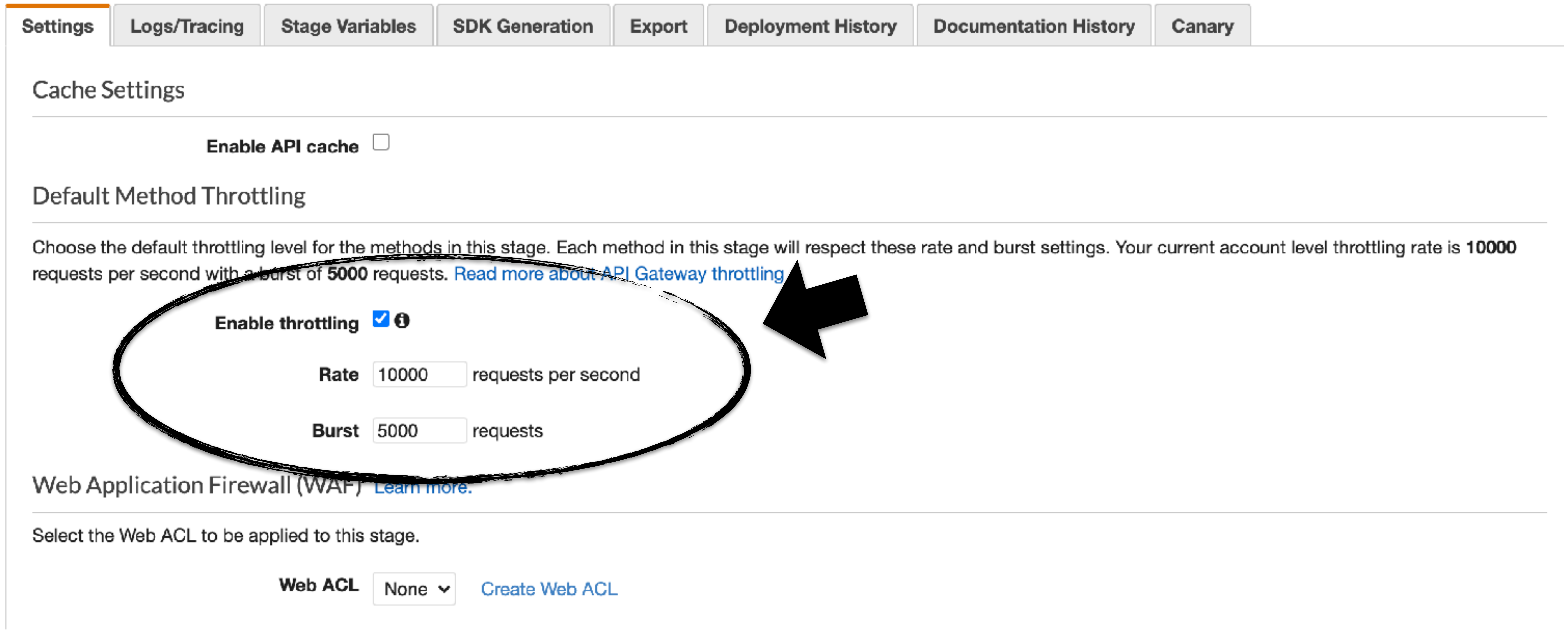
Action

Action

Choose an action to take when a request matches the statements above.

[✓ Block]
Count

AWS WAF



Cache Settings

Enable API cache ☐

Default Method Throttling

Choose the default throttling level for the methods in this stage. Each method in this stage will respect these rate and burst settings. Your current account level throttling rate is **10000** requests per second with a burst of **5000** requests. Read more about API Gateway throttling

Enable throttling ✓

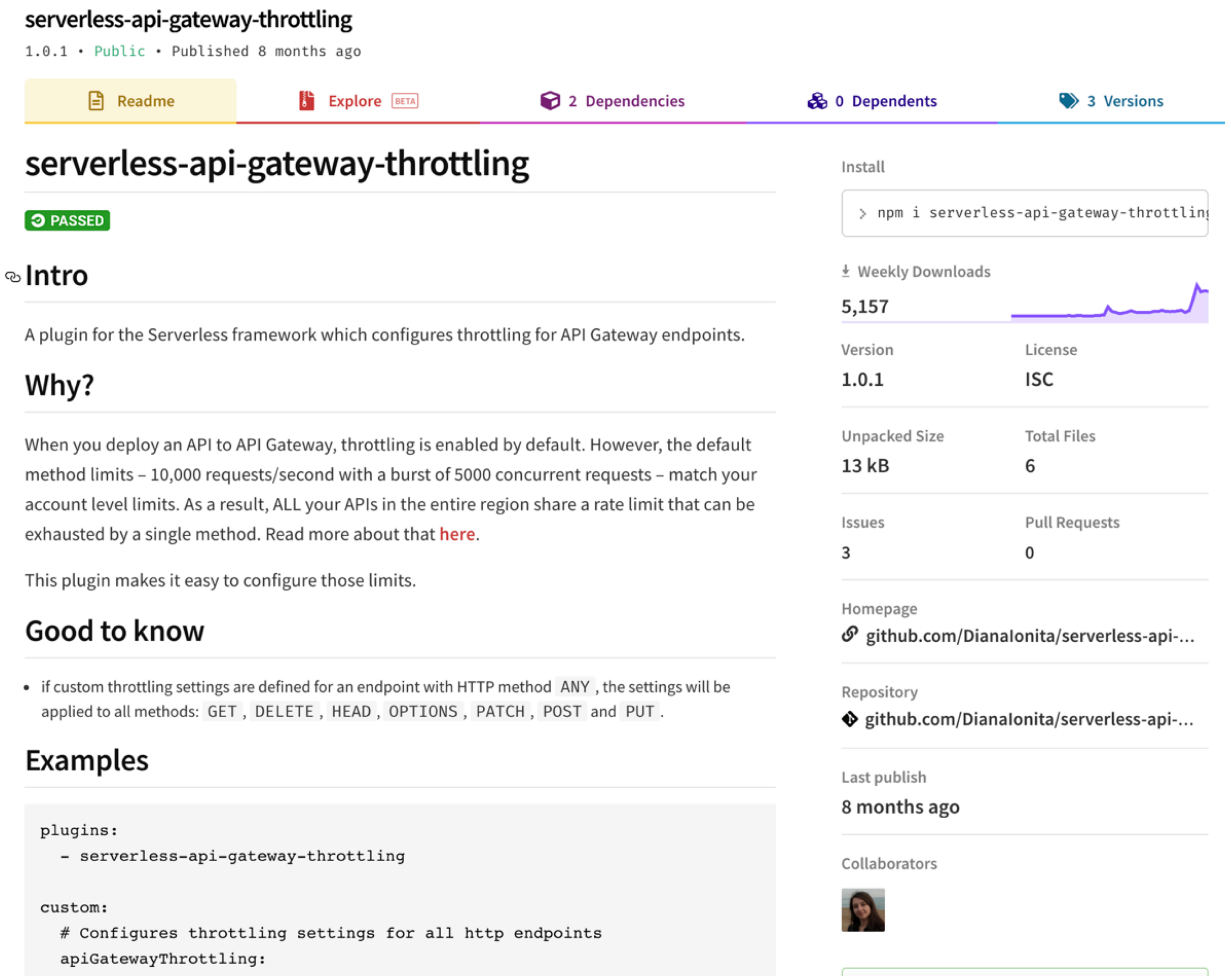
Rate 10000 requests per second

Burst 5000 requests

Web Application Firewall (WAF) Learn more.

Select the Web ACL to be applied to this stage.

Web ACL None



serverless-api-gateway-throttling

1.0.1 • Public • Published 8 months ago

Readme | Explore BETA | 2 Dependencies | 0 Dependents | 3 Versions

serverless-api-gateway-throttling

PASSED

Intro

A plugin for the Serverless framework which configures throttling for API Gateway endpoints.

Why?

When you deploy an API to API Gateway, throttling is enabled by default. However, the default method limits – 10,000 requests/second with a burst of 5000 concurrent requests – match your account level limits. As a result, ALL your APIs in the entire region share a rate limit that can be exhausted by a single method. Read more about that here.

This plugin makes it easy to configure those limits.

Good to know

- if custom throttling settings are defined for an endpoint with HTTP method `ANY`, the settings will be applied to all methods: `GET`, `DELETE`, `HEAD`, `OPTIONS`, `PATCH`, `POST` and `PUT`.

Examples

```
plugins:
  - serverless-api-gateway-throttling

custom:
  # Configures throttling settings for all http endpoints
  apiGatewayThrottling:
```

Install

```
> npm i serverless-api-gateway-throttlin
```

Weekly Downloads
5,157

Version	License
1.0.1	ISC
Unpacked Size	Total Files
13 kB	6
Issues	Pull Requests
3	0

Homepage
github.com/DianaIonita/serverless-api-...

Repository
github.com/DianaIonita/serverless-api-...

Last publish
8 months ago

Collaborators

```
plugins:
  - serverless-api-gateway-throttling

custom:
  # Configures throttling settings for all http endpoints
  apiGatewayThrottling:
    maxRequestsPerSecond: 1000
    maxConcurrentRequests: 500

functions:
  # Throttling settings are inherited from global settings
  update-item:
    handler: rest_api/item/post/handler.handle
    events:
      - http:
          path: /item
          method: post

  # Requests are throttled using this endpoint's throttling configuration
  list-all-items:
    handler: rest_api/items/get/handler.handle
    events:
      - http:
          path: /items
          method: get
          throttling:
            maxRequestsPerSecond: 2000
            maxConcurrentRequests: 1000
```

configure WAF

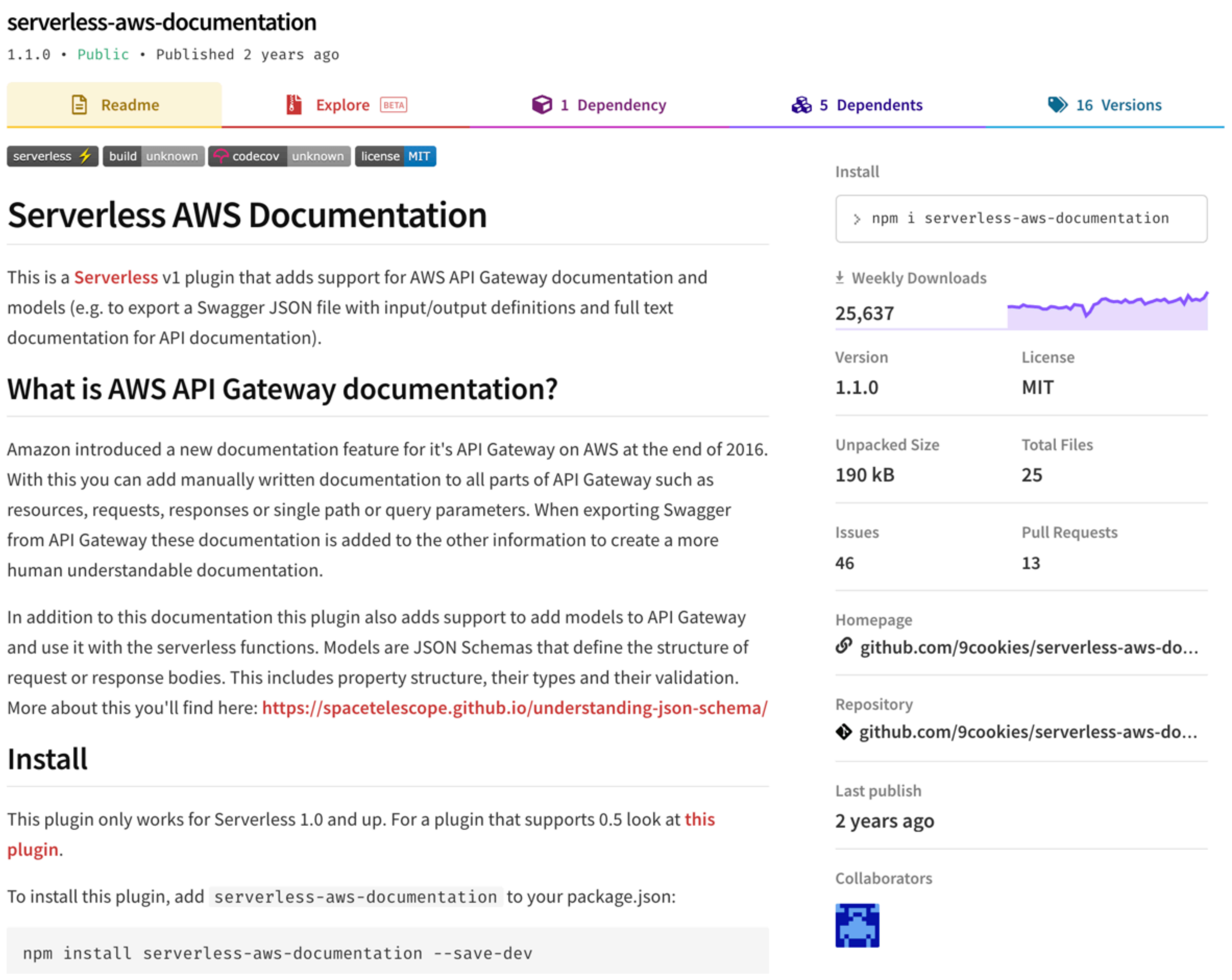
enable request model validation

```yaml
functions:
  create:
    handler: posts.create
    events:
      - http:
          path: posts/create
          method: post
          request:
            schema:
              application/json: ${file(create_request.json)}
```

```json
{
  "definitions": {},
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "title": "The Root Schema",
  "required": ["username"],
  "properties": {
    "username": {
      "type": "string",
      "title": "The Foo Schema",
      "default": "",
      "pattern": "^[a-zA-Z0-9]+$"
    }
  }
}
```

```yaml
functions:
  create:
    handler: posts.create
    events:
      - http:
          path: posts/create
          method: post
          request:
            schema:
              application/json: ${file(create_request.json)}
```

```json
{
  "definitions": {},
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "title": "The Root Schema",
  "required": ["username"],
  "properties": {
    "username": {
      "type": "string",
      "title": "The Foo Schema",
      "default": "",
      "pattern": "^[a-zA-Z0-9]+$"
    }
  }
}
```



serverless-aws-documentation

1.1.0 • Public • Published 2 years ago

Serverless AWS Documentation

This is a Serverless v1 plugin that adds support for AWS API Gateway documentation and models (e.g. to export a Swagger JSON file with input/output definitions and full text documentation for API documentation).

What is AWS API Gateway documentation?

Amazon introduced a new documentation feature for it's API Gateway on AWS at the end of 2016. With this you can add manually written documentation to all parts of API Gateway such as resources, requests, responses or single path or query parameters. When exporting Swagger from API Gateway these documentation is added to the other information to create a more human understandable documentation.

In addition to this documentation this plugin also adds support to add models to API Gateway and use it with the serverless functions. Models are JSON Schemas that define the structure of request or response bodies. This includes property structure, their types and their validation. More about this you'll find here: https://spacetelescope.github.io/understanding-json-schema/

Install

This plugin only works for Serverless 1.0 and up. For a plugin that supports 0.5 look at this plugin.

To install this plugin, add `serverless-aws-documentation` to your package.json:

```
npm install serverless-aws-documentation --save-dev
```

Install

```
> npm i serverless-aws-documentation
```

Weekly Downloads

25,637

Version	License
1.1.0	MIT
Unpacked Size	Total Files
190 kB	25
Issues	Pull Requests
46	13

Homepage

github.com/9cookies/serverless-aws-do...

Repository

github.com/9cookies/serverless-aws-do...

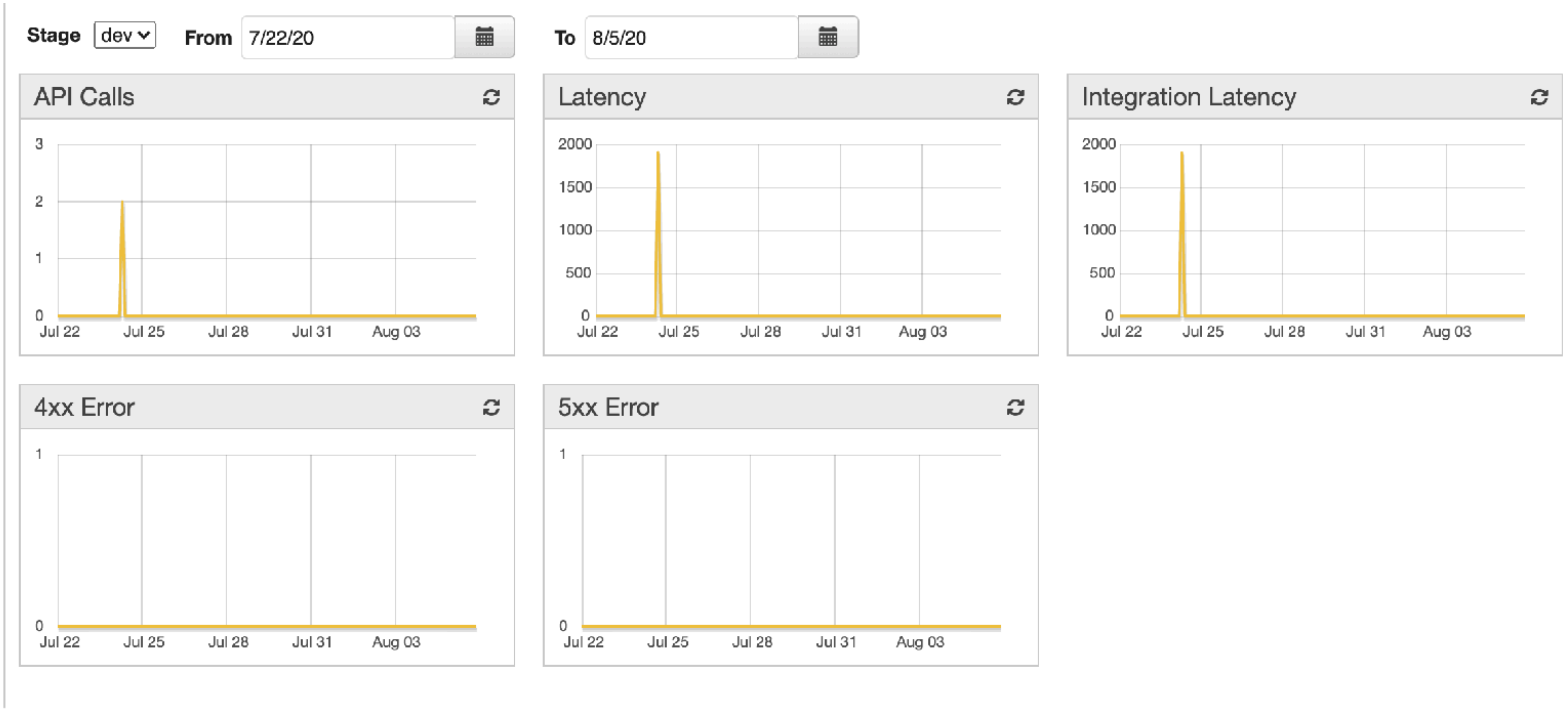
Last publish

2 years ago

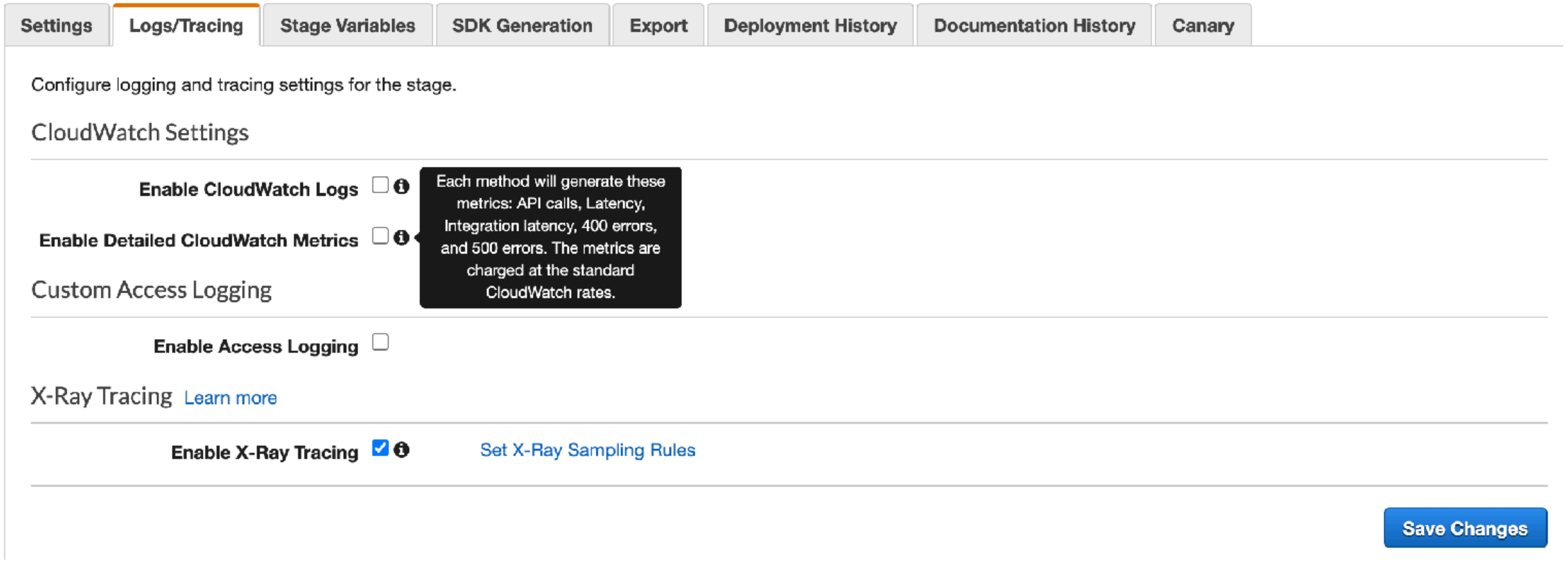
Collaborators

implement response validation

```javascript
const middy = require('@middy/core')
const validator = require('@middy/validator')

const handler = middy((event, context, cb) => {
  cb(null, {})
})

const schema = {
  required: ['body', 'statusCode'],
  properties: {
    body: {
      type: 'object'
    },
    statusCode: {
      type: 'number'
    }
  }
}

handler.use(validator({outputSchema: schema}))

handler({}, {}, (err, response) => {
  expect(err).not.toBe(null)
  expect(err.message).toEqual('Response object failed validation')
  expect(response).not.toBe(null) // it doesn't destroy the response so it can be used by other middlewares
})
```


Stage dev From 7/22/20 To 8/5/20

enable detailed CloudWatch metrics



Configure logging and tracing settings for the stage.

CloudWatch Settings

Enable CloudWatch Logs ☐

Enable Detailed CloudWatch Metrics ☐

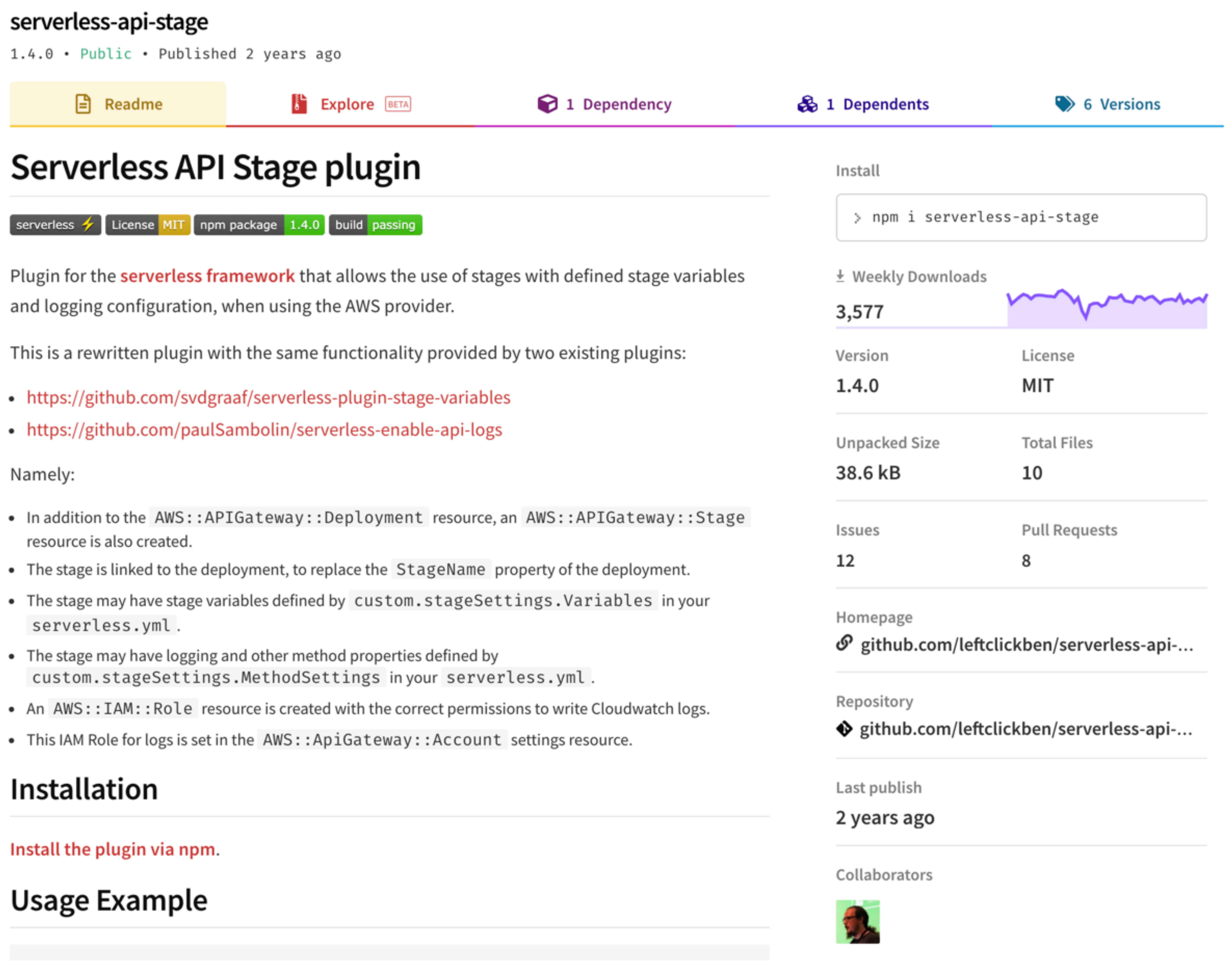
Each method will generate these metrics: API calls, Latency, Integration latency, 400 errors, and 500 errors. The metrics are charged at the standard CloudWatch rates.

Custom Access Logging

Enable Access Logging ☐

X-Ray Tracing Learn more

Enable X-Ray Tracing ☑



serverless-api-stage

1.4.0 • Public • Published 2 years ago

Readme | Explore BETA | 1 Dependency | 1 Dependents | 6 Versions

Serverless API Stage plugin

serverless ⚡ | License MIT | npm package 1.4.0 | build passing

Plugin for the serverless framework that allows the use of stages with defined stage variables and logging configuration, when using the AWS provider.

This is a rewritten plugin with the same functionality provided by two existing plugins:

- https://github.com/svdgraaf/serverless-plugin-stage-variables
- https://github.com/paulSambolin/serverless-enable-api-logs

Namely:

- In addition to the `AWS::APIGateway::Deployment` resource, an `AWS::APIGateway::Stage` resource is also created.
- The stage is linked to the deployment, to replace the `StageName` property of the deployment.
- The stage may have stage variables defined by `custom.stageSettings.Variables` in your `serverless.yml`.
- The stage may have logging and other method properties defined by `custom.stageSettings.MethodSettings` in your `serverless.yml`.
- An `AWS::IAM::Role` resource is created with the correct permissions to write Cloudwatch logs.
- This IAM Role for logs is set in the `AWS::ApiGateway::Account` settings resource.

Installation

Install the plugin via npm.

Usage Example

Install

```
> npm i serverless-api-stage
```

Weekly Downloads

3,577

Version	License
1.4.0	MIT
Unpacked Size	Total Files
38.6 kB	10
Issues	Pull Requests
12	8

Homepage

github.com/leftclickben/serverless-api-...

Repository

github.com/leftclickben/serverless-api-...

Last publish

2 years ago

Collaborators

CloudWatch Alarm

p90/p95/p99 Latency > Xms

CloudWatch Alarm

p90/p95/p99 Latency > Xms

CloudWatch Alarm

Average 4xxError or 5xxError > X %

record custom application metrics

Amazon CloudWatch Launches Embedded Metric Format

Posted On: Nov 18, 2019

CloudWatch Embedded Metric Format enables you to ingest complex high-cardinality application data in the form of logs and easily generate actionable metrics from them. It has traditionally been hard to generate actionable custom metrics from your ephemeral resources such as Lambda functions, and containers. With this launch, you do not have to rely on complex architecture or multiple third party tools to gain insights into these environments. By sending your logs in the new Embedded Metric Format, you can now easily create custom metrics without having to instrument or maintain separate code, while gaining powerful analytical capabilities on your log data.

There are several benefits of this new feature. You can embed custom metrics alongside detailed log event data, and CloudWatch will automatically extract the custom metrics so you can visualize and alarm on them, for real-time incident detection. Additionally, the detailed log events associated with the extracted metrics can be queried using CloudWatch Logs Insights to provide deep insights into the root causes of operational events.

You can generate log events in the Embedded Metric Format using the open-sourced client libraries available on GitHub or manually construct them conforming to a defined specification. Once generated, these events are sent to CloudWatch using the client libraries, the CloudWatch Agent or by directly calling the PutLogEvents API.

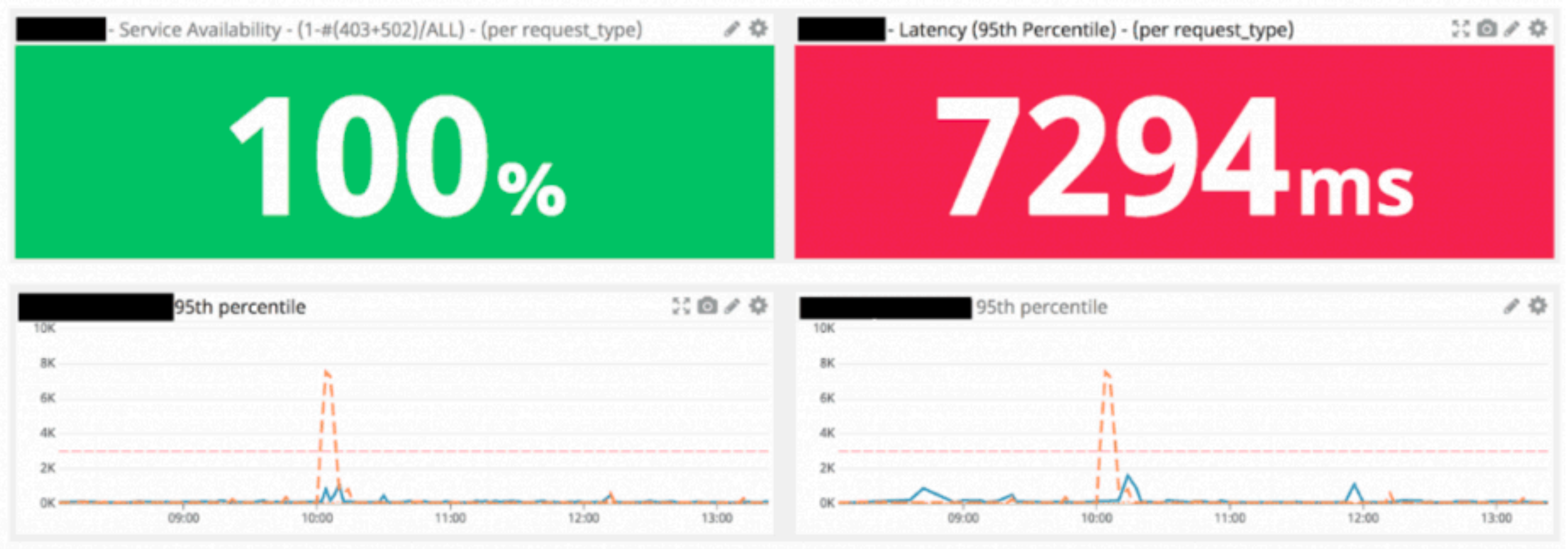
CloudWatch Embedded Metric Format is available in all AWS Regions where CloudWatch Logs is available. There are no additional charges for using this new feature, and you simply pay for usage of CloudWatch logs and metrics. To learn more, visit the documentation on CloudWatch Embedded Metric Format.

Embedded Metric Format Example and JSON Schema

The following is a valid example of embedded metric format.

```json
{
  "_aws": {
    "Timestamp": 1574109732004,
    "CloudWatchMetrics": [
      {
        "Namespace": "lambda-function-metrics",
        "Dimensions": [["functionVersion"]],
        "Metrics": [
          {
            "Name": "time",
            "Unit": "Milliseconds"
          }
        ]
      }
    ]
  },
  "functionVersion": "$LATEST",
  "time": 100,
  "requestId": "989ffbf8-9ace-4817-a57c-e4dd734019ee"
}
```

set up API dashboards


[redacted] - Service Availability - (1-#(403+502)/ALL) - (per request_type)
100%
[redacted] - Latency (95th Percentile) - (per request_type)
7294ms
[redacted] 95th percentile
10K
8K
6K
4K
2K
0K
09:00
10:00
11:00
12:00
13:00
[redacted] 95th percentile
10K
8K
6K
4K
2K
0K
09:00
10:00
11:00
12:00
13:00