



AWS Platform Technology Serverless Technical Architecture

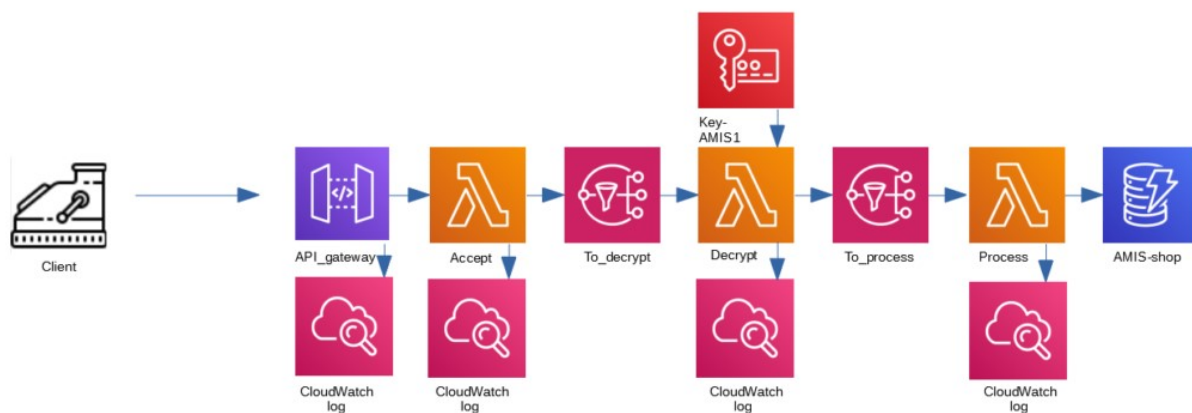
AWS Shop example: the API Gateway (1)



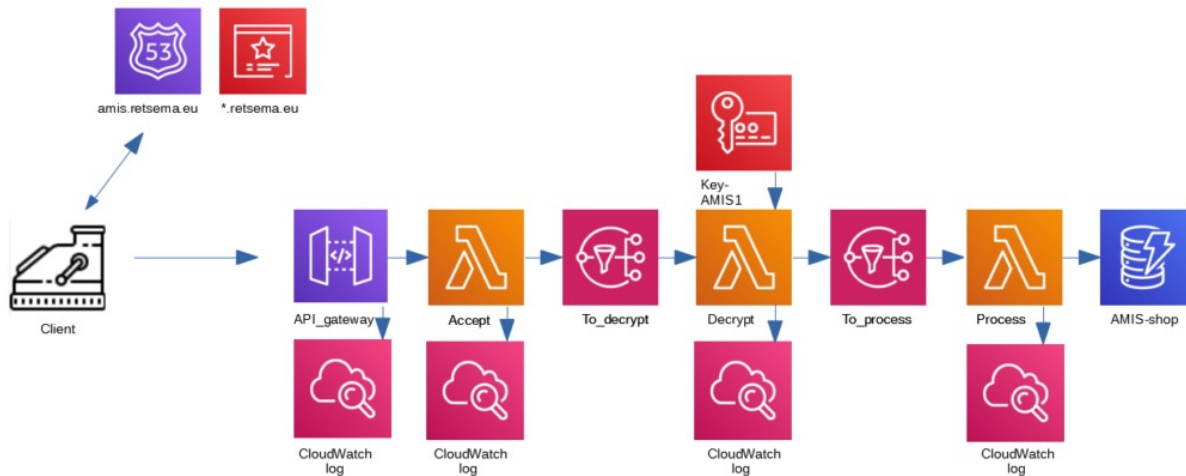
Frederique Retsema · May 9, 2020

Introduction

You might have noticed that we skipped the API Gateway up to now [1]. I will write two blogs about the API Gateway. In this one, we follow the simplest route: from sending the message to the API gateway, and let the API Gateway deliver the message at the Lambda function:

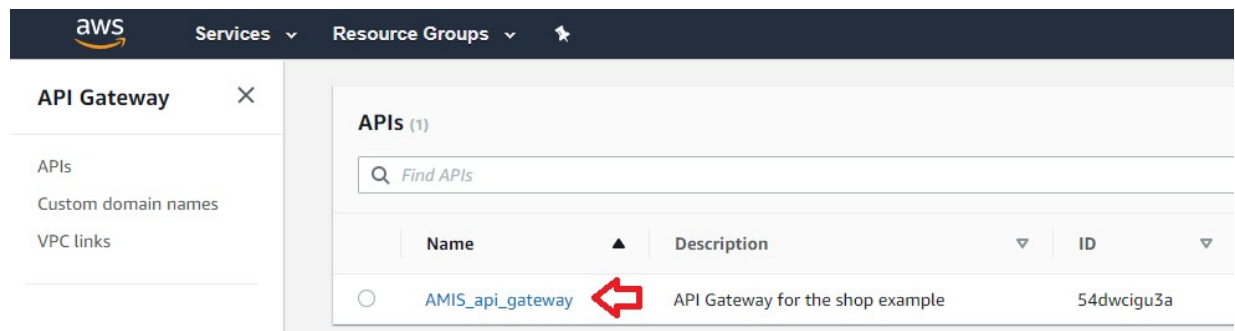


In the next blog, I will add the functionality for using a DNS entry:



API Gateway

When you go to the API gateway in the region where the shop is running, you can already see the APIs. Click on the link AMIS_api_gateway:



You will come on the screen with resources. Resources are your end points: it is the latest part of the link we use when we send data to the API Gateway. Let's look at these links: you might have seen the following text when you roll out the shop in your own environment:

Try for yourself: use the following commands to see if the rollout was successful:

```
cd ~/AMIS-Blog-AWS/shop-1/client
```

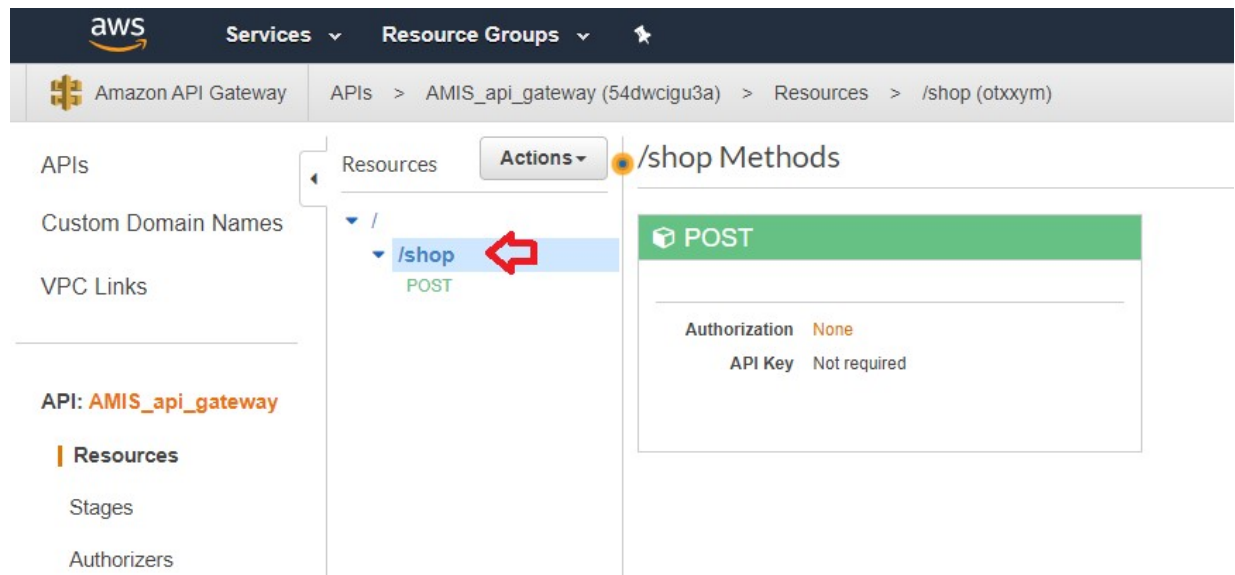
```
./encrypt_and_send.py AMIS1 https://54dwcigu3a.execute-api.eu-west-1.amazonaws.com/prod/shop
```

or

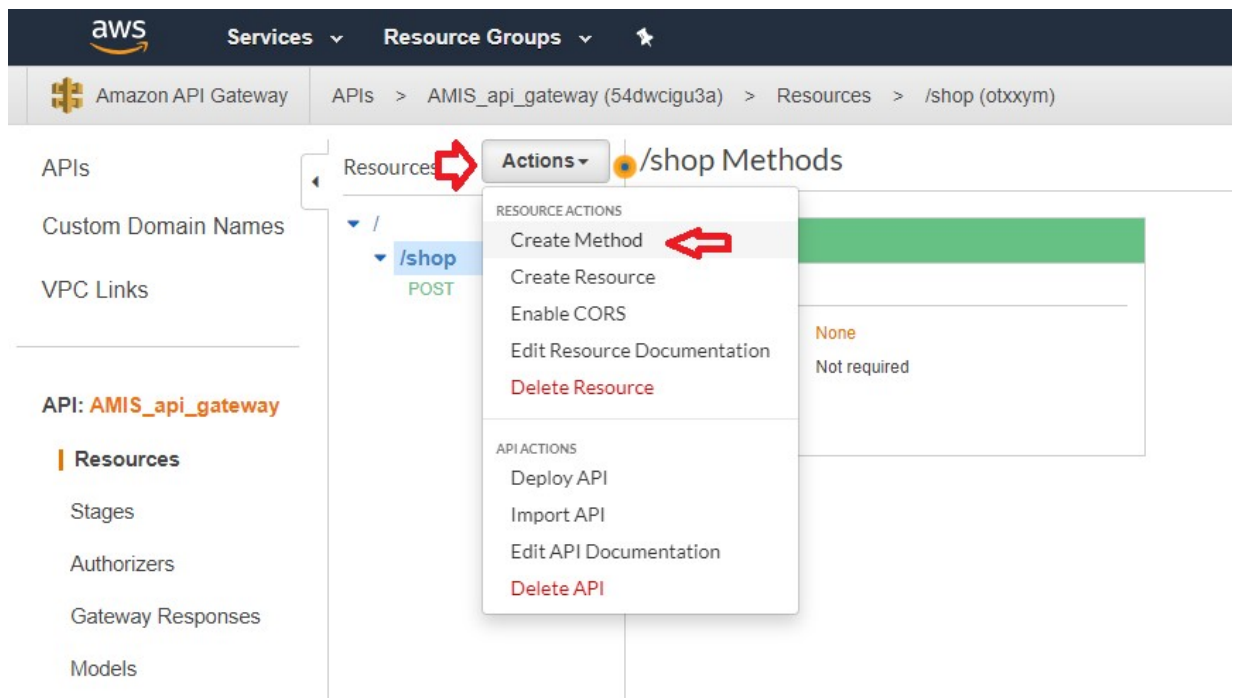
```
cd ~/AMIS-Blog-AWS/shop-1/client
```

```
./encrypt_and_send.py AMIS1 https://amis.retsema.eu/shop
```

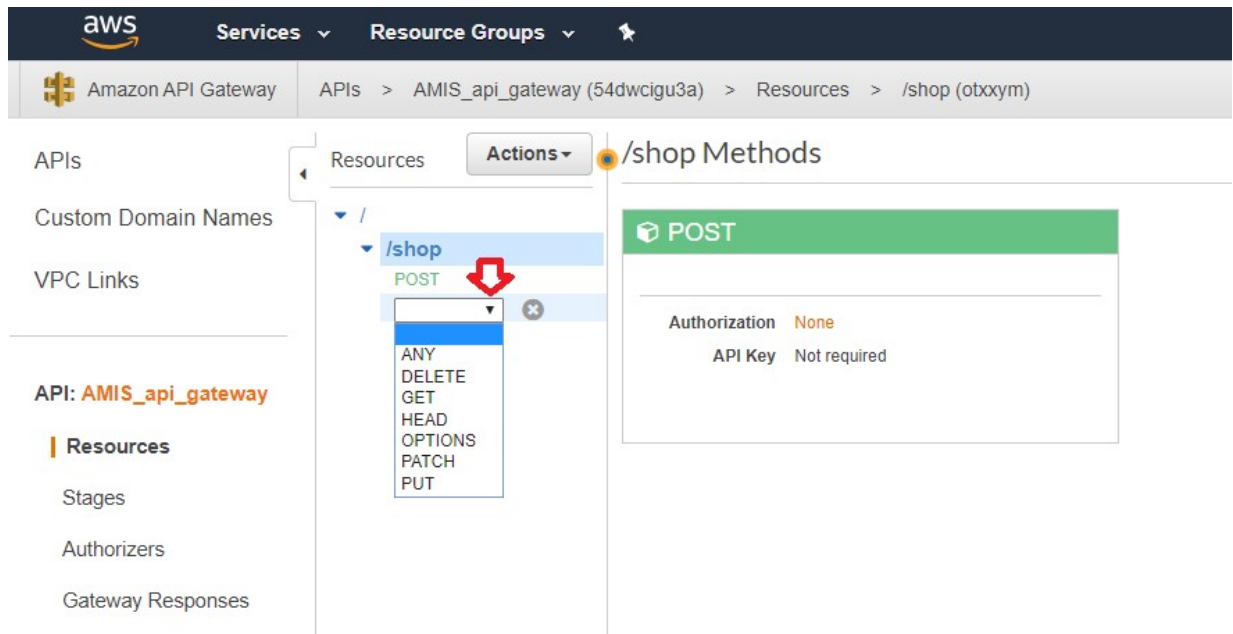
Both in the long URL and in the nice URL of our own domain, the last part of the URL is the shop resource. It is possible to use multiple resources and connect them to each other. In theory, we could have added a resource for the cash machines (/shop/sales), and add another resources under shop for when new stock arrives at a shop and use a url that ends with /shop/newstock for that. We didn't implement that, our cash machines use /shop, so click on /shop:



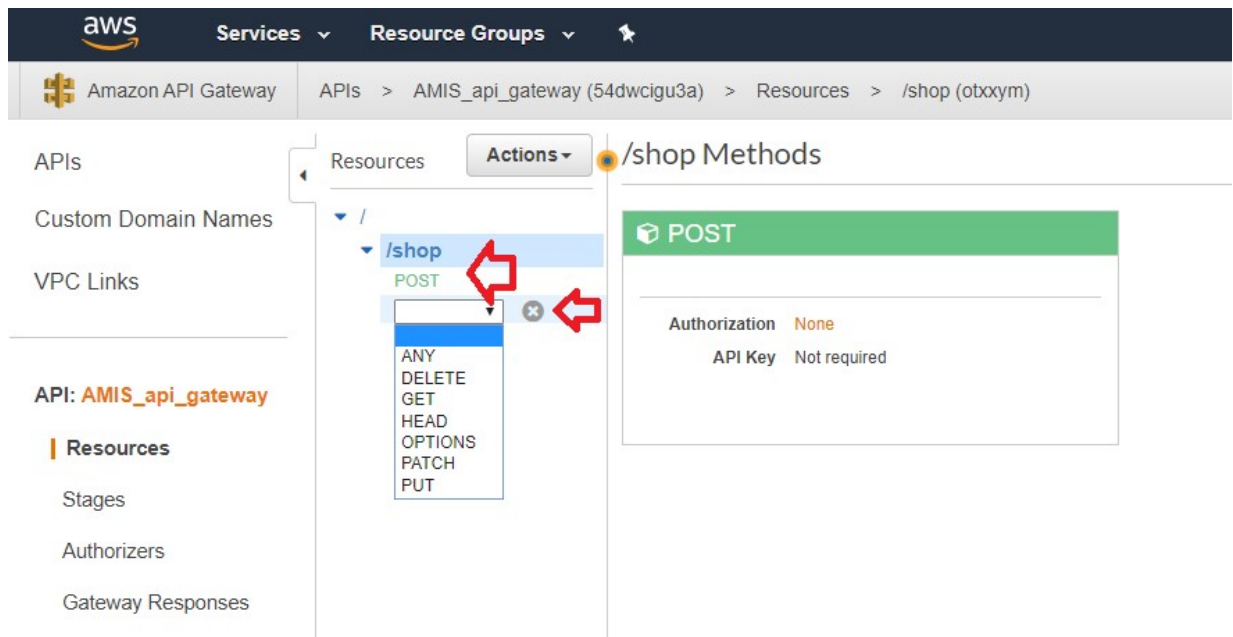
You can see, that we implemented just one method: a POST method. Let's see what is possible here: click on the Actions button, then on Create Method:



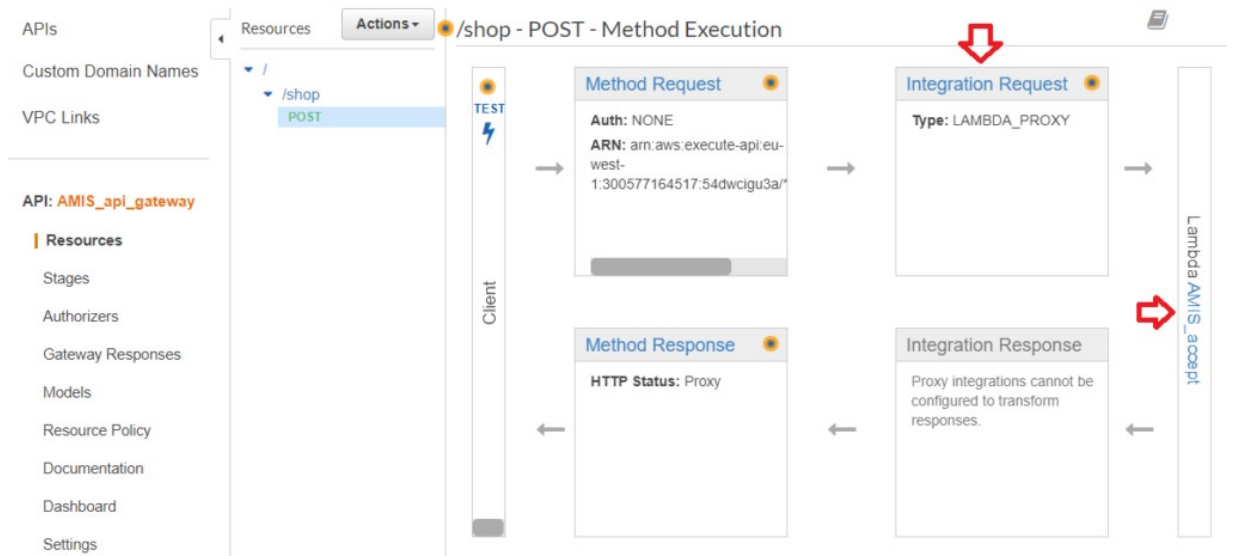
You can see an extra drop down list under POST, click on it: you can see that you can specify multiple kinds of methods to this resource. Or you can use just one method: ANY for all of them. With ANY you can use (for example) the Lambda function to determine the type of method and then act on that.



Click on the X next to the drop down window and on POST now:



You can now see what the API gateway does (or: can do) for us. We use the Lambda function AMIS_accept (as you see on the right side of this screen), but we could send the message to other AWS services. Click on Integration Request:



You can send the message to lots of other AWS services, for a list click on the radio button before AWS service:

APIs

Custom Domain Names

VPC Links

API: **AMIS_api_gateway**

Resources

Stages

Authorizers

Gateway Responses

Models

Resource Policy

Documentation

Dashboard

Settings

Usage Plans

API Keys

Resources

/

/shop

POST

Method Execution /shop - POST - Integration Request

Provide information about the target backend that this method will call and whether the incoming request data should be modified.

Integration type

- ☒ Lambda Function ⓘ
- ☐ HTTP ⓘ
- ☐ Mock ⓘ
- ☐ AWS Service ⓘ
- ☐ VPC Link ⓘ

Use Lambda Proxy integration ☒ ⓘ

Lambda Region eu-west-1 ⓘ

Lambda Function AMIS_accept ⓘ

Execution role ⓘ

Invoke with caller credentials ☐ ⓘ

Credentials cache Do not add caller credentials to cache key ⓘ

Use Default Timeout ☒ ⓘ

Fill in your region (on the moment I write this blog, I am using eu-west-1, but you can use any region). When you use the scrollbar to go down, you will see an impressive list of services...

API: **AMIS_api_gateway**

Resources

Stages

Authorizers

Gateway Responses

Models

Resource Policy

Documentation

Dashboard

Settings

Usage Plans

API Keys

Client Certificates

Settings

Integration type

- ☐ HTTP ⓘ
- ☐ Mock ⓘ
- ☒ AWS Service ⓘ
- ☐ VPC Link ⓘ

AWS Region eu-west-1 ⓘ

AWS Service

AWS Subdomain Amazon MQ

HTTP method POST

Action Type Auto Scaling

Path override (optional) Cloud9

Execution role CloudFormation

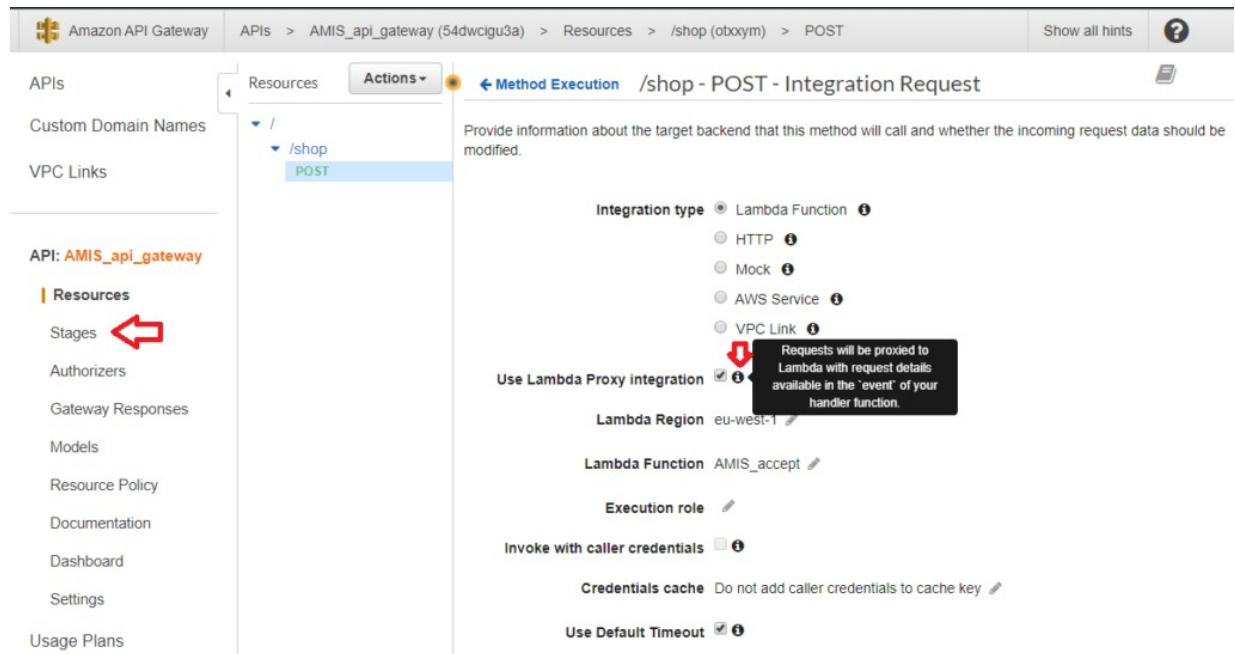
Content Handling CloudFront

Use Default Timeout ☒ ⓘ

Save

When you click on the Lambda Function radio button, then you can see the configuration for sending messages to our Lambda function. You see, that the checkbox Use Lambda Proxy integration is checked. Hoover over the information button next to it: when you get too much information about the request in the event data of the Lambda function (we currently don't do anything with it), you

might consider to uncheck this checkbox. For now, we leave it as it is. Click on Stages in the left menu:



Stages

On this moment, our shop uses just one stage: the prod stage. When you click on prod, you will see many settings of the API Gateway. The most important one is, the Invoke URL that you see on top of the screen. You might have noticed that this URL is the same as the URL you saw on the last lines of the output of `./init-all.sh`:

Try for yourself: use the following commands to see if the rollout was successful:

```
cd ~/AMIS-Blog-AWS/shop-1/client
```

```
./encrypt_and_send.py AMIS1 https://54dwcigu3a.execute-api.eu-west-1.amazonaws.com/prod/shop
```

The first part of the URL is the same, the only part that you don't see here is the resource (`/shop`). And that makes sense, as it is possible to add multiple resources under one URL.

Another option that can be useful is the API cache. Click on the checkbox for API cache:


Stages **Create** prod Stage Editor Delete Stage Configure Tags

prod

Invoke URL: <https://54dwcigu3a.execute-api.eu-west-1.amazonaws.com/prod>

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

Cache Settings

Enable API cache ☒ 

Enabling API cache increases cost and is not covered by the free tier. [See pricing for more details](#)

Cache capacity

Encrypt cache data ☐

Cache time-to-live (TTL)

Per-key cache invalidation

Require authorization ☒

Handle unauthorized requests

When you get a lot of GET requests and the answers are the same for some time, then you can enable the cache. Requests will not be sent to other AWS services, this might save you time, AWS resources and therefore money. You can specify the cache capacity, can specify to encrypt the cache data (or not) and can set the time to live (TTL) in seconds. For our shop example we don't use this, so switch it back to off.

Another nice option is throttling: this is enabled by default. In our case, the allowed rate might be much too high: there is no way in which our cash machines will send 10000 requests per second. Throttling the number of requests per second might lower your bill when some nice fellow will send you thousands of rubbish messages per second. Your cash machines should be able to resend the message, though. Click on Logs/Tracing now:

APIs Stages **Create** prod Stage Editor Delete Stage Configure

Custom Domain Names prod

VPC Links

API: **AMIS_api_gateway**

Resources

Stages

Authorizers

Gateway Responses

Models

Resource Policy

Documentation

Dashboard

Settings

Usage Plans

Invoke URL: <https://54dwcigu3a.execute-api.eu-west-1.amazonaws.com/prod>


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 requests per second

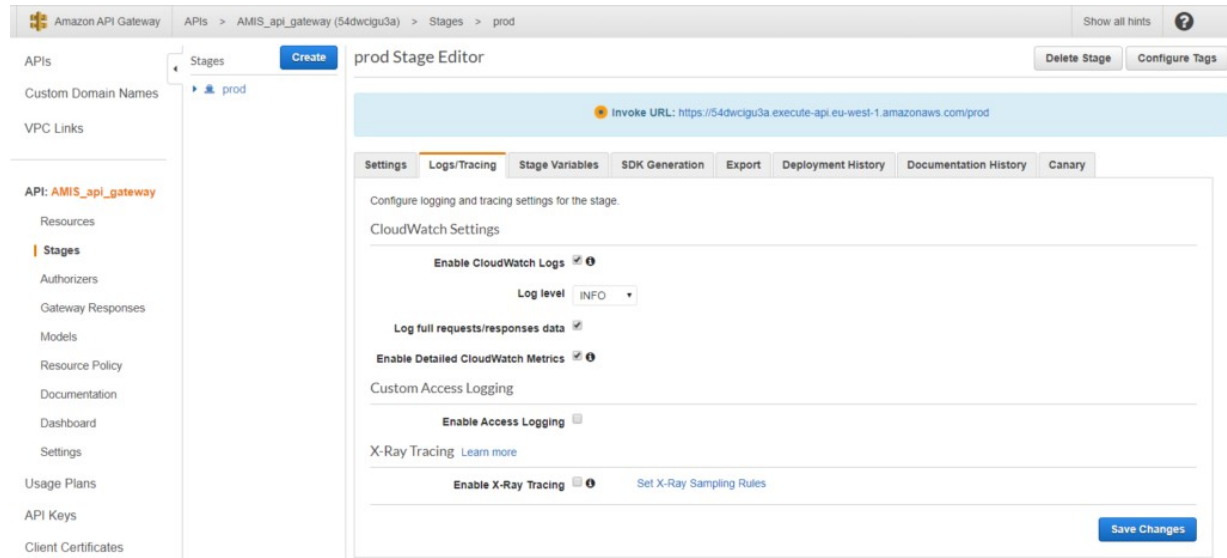
Burst requests

Web Application Firewall (WAF) [Learn more.](#)

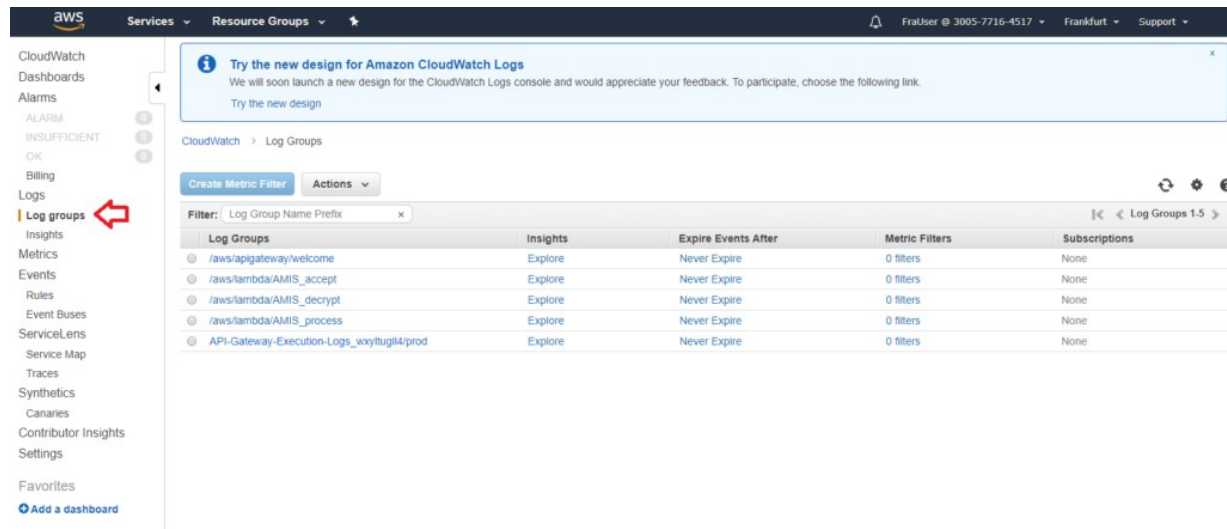
Select the Web ACL to be applied to this stage.

Logging and tracing

You might have seen in CloudWatch, that the API gateway is sending logs to CloudWatch. You can configure that in this screen:



With this configuration, you will get a lot of data in CloudWatch logs. Let's look at that: go to the CloudWatch service, and click on Log groups:



You see two log groups for the API Gateway: one with just a welcome message and the other one with data. Let's look in the log group with the data (in my case it is called API-Gateway-Execution-Logs_wxytlugll4). When you click on the link of the log stream, then you see the data that is generated for one message:

CloudWatch > Log Groups > Streams for API-Gateway-Execution-Logs_...

Search Log Group Create Log Stream Delete Log Stream

Filter: Log Stream Name Prefix x

Log Streams

b01631c245d0d460ed40290a394dd330 2020-04-16 21:24 UTC+2

CloudWatch > Log Groups > API-Gateway-Execution-Logs_wxytugll4/prod > b01631c245d0d460ed40290a394dd330

Expand all Row Text

Filter events all 2020-04-17 (19:24:27)

Time (UTC +00:00)	Message
2020-04-18	No older events found at the moment. Retry
19:24:25	(1593a40e-23d3-43aa-a11c-404f45829f91) Extended Request Id: LM0MhF1glAFg9A=
19:24:25	(1593a40e-23d3-43aa-a11c-404f45829f91) Verifying Usage Plan for request: 1593a40e-23d3-43aa-a11c-404f45829f91. API Key: API Stage: wxytugll4/prod
19:24:25	(1593a40e-23d3-43aa-a11c-404f45829f91) API Key authorized because method 'POST /shop' does not require API Key. Request will not contribute to throttle or quota limits
19:24:25	(1593a40e-23d3-43aa-a11c-404f45829f91) Usage Plan check succeeded for API Key and API Stage wxytugll4/prod
19:24:25	(1593a40e-23d3-43aa-a11c-404f45829f91) Starting execution for request: 1593a40e-23d3-43aa-a11c-404f45829f91
19:24:25	(1593a40e-23d3-43aa-a11c-404f45829f91) HTTP Method: POST, Resource Path: /shop
19:24:25	(1593a40e-23d3-43aa-a11c-404f45829f91) Method request path: ()
19:24:25	(1593a40e-23d3-43aa-a11c-404f45829f91) Method request query string: ()
19:24:25	(1593a40e-23d3-43aa-a11c-404f45829f91) Method request headers: {Accept: "", User-Agent: python-requests/2.23.0, X-Forwarded-Proto: https, X-Forwarded-For: 86.88.108.53, Host: w...
19:24:25	(1593a40e-23d3-43aa-a11c-404f45829f91) Method request body before transformations: {"shop_id": "AMIS1", "content_base64": "wKBFQav0301C1ZgyUxc019AAoVNuLUXoFLp3UlyUbf...
19:24:25	(1593a40e-23d3-43aa-a11c-404f45829f91) Endpoint request URI: https://lambda.eu-central-1.amazonaws.com/2015-03-31/functions/am:aws.lambda.eu-central-1.300577164517.function.
19:24:25	(1593a40e-23d3-43aa-a11c-404f45829f91) Endpoint request headers: {x-amzn-lambda-integration-lag: 1593a40e-23d3-43aa-a11c-404f45829f91, Authorization: *****}
19:24:25	(1593a40e-23d3-43aa-a11c-404f45829f91) Endpoint request body after transformations: {"resource": "/shop", "path": "/shop", "httpMethod": "POST", "headers": {"Accept": "", "Accept-Encoding": ...}}
19:24:25	(1593a40e-23d3-43aa-a11c-404f45829f91) Sending request to https://lambda.eu-central-1.amazonaws.com/2015-03-31/functions/am:aws.lambda.eu-central-1.300577164517.function.AM...
19:24:27	(1593a40e-23d3-43aa-a11c-404f45829f91) Received response. Status: 200. Integration latency: 1338 ms
19:24:27	(1593a40e-23d3-43aa-a11c-404f45829f91) Endpoint response headers: {Date: Sat, 18 Apr 2020 19:24:27 GMT, Content-Type: application/json, Content-Length: 86, Connection: keep-aliv...
19:24:27	(1593a40e-23d3-43aa-a11c-404f45829f91) Endpoint response body before transformations: {"statusCode": 200, "headers": {"Content-Type": "application/json"}, "body": "OK!"}
19:24:27	(1593a40e-23d3-43aa-a11c-404f45829f91) Method response body after transformations: "OK"
19:24:27	(1593a40e-23d3-43aa-a11c-404f45829f91) Successfully completed execution
19:24:27	(1593a40e-23d3-43aa-a11c-404f45829f91) The request completed with status: 200
19:24:27	(1593a40e-23d3-43aa-a11c-404f45829f91) AWS Integration Endpoint RequestId: f51f4d65-9765-4e58-8bd0-51566e9d8640
19:24:27	No newer events found at the moment. Retry

This is rather much, you can choose to log only error messages or not to log the full request/responses data.

To enable logging, you have to specify the role that the API Gateway may use. Go back to the API Gateway, and click on Settings:

Amazon API Gateway APIS > AMIS_api_gateway (54dwcigu3a) > Stages > prod

APIs Custom Domain Names VPC Links

API: AMIS_api_gateway

Resources Stages Authorizers Gateway Responses Models Resource Policy Documentation Dashboard Settings Usage Plans API Keys Client Certificates Settings

prod Stage Editor

Invoke URL: https://54dwcigu3a.execute-api.eu-west-1.amazonaws.com/prod

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

Configure logging and tracing settings for the stage.

CloudWatch Settings

Enable CloudWatch Logs ☒ [Info](#)

Log level INFO

Log full requests/responses data ☒

Enable Detailed CloudWatch Metrics ☒ [Info](#)

Custom Access Logging

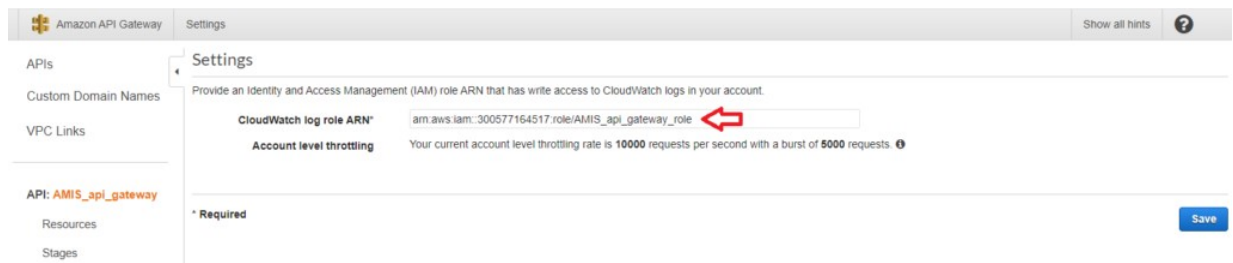
Enable Access Logging ☐

X-Ray Tracing [Learn more](#)

Enable X-Ray Tracing ☐ [Set X-Ray Sampling Rules](#)

Save Changes

You can see that the ARN (Amazon Resource Name) is filled with a role that will be used when something is logged. The role should use a policy that allows the access.

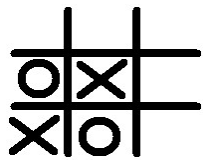


Conclusion

The API Gateway has very many options, in the beginning the number of screens and the number of options (and the possibilities) can be overwhelming. My advise is: play with it. Try things out. And get a feeling what option is where...

A small exercise to play along: I didn't show you where you could find the role that the API gateway uses to send log data to CloudWatch. Can you, when you see the name of the ARN, find the role and the policy that is used? Hint: look back at the blog about Lambda to see where the roles and the policies in AWS are configured...

Play along



I scripted the solution [2]. You can follow along and create this solution in your own environment, see the README.md file in the vagrant directory and see the introduction blog for more information.

Links

[1] This is the fifth blog about this shop. Links to the previous blogs:

- Introduction: <https://technology.amis.nl/2020/04/26/example-application-in-aws-using-lambda/>
- Lambda and IAM: <https://technology.amis.nl/2020/04/29/aws-shop-example-lambda/>
- SNS: <https://technology.amis.nl/2020/05/02/aws-shop-about-the-aws-simple-notification-service-sns/>
- DynamoDB: <https://technology.amis.nl/2020/05/05/aws-shop-dynamodb-the-aws-nosql-database/>

[2] <https://github.com/FrederiqueRetsema/AMIS-Blog-AWS> , directory shop-1

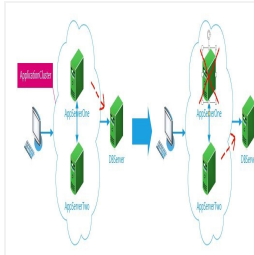
Related Posts:



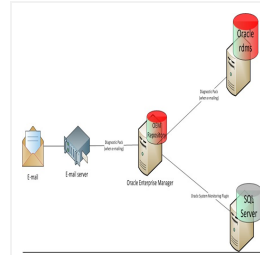
Example application in



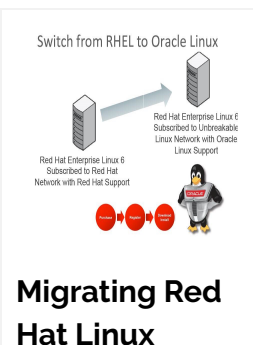
new Puppet 3 Weblogic



SQL*Plus / SQL*Net Dead



Oracle licenses needed while



Migrating Red Hat Linux

Leave a Reply

Enter your comment here...

This site uses Akismet to reduce spam. [Learn how your comment data is processed.](#)

You May Like