# API Gateway

Concepts

* Supports
  + Stateful – WebSocket APIs
  + Stateless – REST APIs
* Utilizes CloudTrail – monitoring and logging API usage and changes
* Integration with AWS WAF – protecting API against web exploits
* Integration with AWS X-Ray – showcase API usage

Endpoint types

* **Edge optimized endpoint**
  + Requests routed to **nearest endpoint via cloud front**
* **Regional API endpoints**
  + Serves requests from same region
* **Private API endpoints**
  + Can only be accessed from VPC via ENI (endpoint n/w interface)

# Controlling access to REST API in API Gateway

* **Resource policies**
  + Used **to allow/deny access from specified source IP address/VPC endpoint**
  + Identities
    - Users **from a specified AWS account**
    - CIDR blocks/source IP ranges
    - Specific VPC or VPC endpoint
* **Standard IAM roles and policies**
  + **Control access for manage API**
  + Control **cross account access to API**
  + **Control access for invoking API**
    - For IAM policies to be effective, **you must have enabled IAM authentication on API methods by setting** **AWS\_IAM** for the methods' **[authorizationType](https://docs.aws.amazon.com/apigateway/api-reference/resource/method/" \l "authorizationType)** property. Failing to do so will make these API methods publicly accessible.
    - "Action": [ "**execute-api:Invoke**"

where action is an available API-executing action:

* \*, which represents all of the following actions.
* **Invoke**, used to invoke an API upon a client request.
* **InvalidateCache**, used to invalidate API cache upon a client request.
  + IAM Policy Example for managing API

{

"Version": "2012-10-17",

"Statement": [

{

"Sid": "Stmt1467321237000",

"Effect": "Deny",

"Action": [

"apigateway:POST",

"apigateway:PUT",

"apigateway:PATCH",

"apigateway:DELETE"

],

"Resource": [

"arn:aws:apigateway:us-east-1::/\*"

]

},

{

"Sid": "Stmt1467321341000",

"Effect": "Deny",

"Action": [

"apigateway:GET"

],

"Resource": [

"arn:aws:apigateway:us-east-1::/",

"arn:aws:apigateway:us-east-1::/account",

"arn:aws:apigateway:us-east-1::/clientcertificates",

"arn:aws:apigateway:us-east-1::/domainnames",

"arn:aws:apigateway:us-east-1::/apikeys"

]

},

{

"Sid": "Stmt1467321344000",

"Effect": "Allow",

"Action": [

"apigateway:GET"

],

"Resource": [

"arn:aws:apigateway:us-east-1::/restapis/\*"

]

}

]

}

* + IAM policy example for execution

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"**execute-api:Invoke**"

],

"Resource": [

"arn:aws:execute-api:us-east-1:\*:a123456789/test/POST/mydemoresource/\*"

]

}

]

}

* IAM tags
* VPC Endpoint policies for Interface VPC endpoints
  + A **VPC endpoint policy is an IAM resource policy** that you can attach to an interface VPC endpoint to control access to the endpoint.
  + **VPC endpoint policies can be used together with API Gateway resource policies**.
    - The **resource policy** is used to specify which principals can access the API.
    - The **endpoint policy** specifies **which private APIs can be called via the VPC** endpoint
* Lambda authorizers
  + **Bearer token used**
* Cognito user pools

API Gateway resource policies

* To allow API users in **another AWS account**
  + **SigV4 protocols**
  + **execute-api:Invoke** – Allow/Deny
* Deny API traffic based on source IP address

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Principal": "\*",

"Action": "execute-api:Invoke",

"Resource": [

"arn:aws:execute-api:region:account-id:api-id/\*"

]

},

{

"Effect": "Deny",

"Principal": "\*",

"Action": "execute-api:Invoke",

"Resource": [

"arn:aws:execute-api:region:account-id:api-id/\*"

],

"Condition" : {

"IpAddress": {

**"aws:SourceIp": ["192.0.2.0/24", "198.51.100.0/24" ]**

}

}

}

]

}

* Allow **private API traffic based on VPC or VPC endpoint**

Control access to an API with IAM Permissions

* API Developer permission
* API Caller permission
* IAM Policy for Cross-Account Lambda Authorizer
* IAM Policy for Cross-Account Lambda integration
  + To grant another AWS account permission to call **integration:put/put-integration**
* Allow another account to manage lambda function when importing OpenAPI
  + Call restapi:import/import-rest-api

Lambda Authorizers

* It’s also called as custom authorizer
* Used to implement custom authorization **with bearer token such as OAuth, SAML**
* It takes user’s identity as input and return IAM policy as output
* Two types of LA
  + Token-based
    - Receives bearer token such **as JWT or OAuth token**
  + Request parameter based
    - Receives identity as combination of **headers, query string parameters, stageVariables, $context variables**
    - **WebSocket API** only support request based
* Cross account authorization is possible

# Supported Integration types

API Gateway supports,

* AWS
  + This type of **integration lets an API expose AWS service actions**. In AWS integration, you must configure both **the integration request and integration response** and set up **necessary data mappings from the method request to the integration request**, and from the **integration response to the method response**.
* HTTP – Custom HTTP integration. Expose HTTP endpoint
* HTTP\_PROXY – HTTP backend endpoint
* AWS\_PROXY - Lambda
* MOCK – Mock API


                API Gateway Lambda authorization workflow
            

**Flow**

1. Client calls API passing bearer token/request params
2. API Gateway checks whether a lambda authorizer is configured for the method. If it is, APIG calls lambda *fn.*
3. Authentication by Lambda *fn*
   1. Call out OAuth provider to get OAuth access token
   2. Call out SAML provider to get a SAML assertion
   3. Generate IAM policy based on request parameter values
   4. Retrieving credentials from DB
4. If call succeeds, returns output object with **IAM policy + principal**
5. APIG evaluates policy
   1. If access denied, return 403 ACCESS\_DENIED (forbidden error)
   2. If access allowed, executes API method. If caching enabled, it APIG also catches policies. (200, Ok)
   3. If access is ‘unauthorized’ , 401 Unauthorized HTTP response
   4. If token is invalid, Http 500 – Invalid Token response

Configure Lambda authorizer using console

* Choose Lambda Event payload
  + **Token** – token authorizer
  + **Request** – Request authorizer
* Token based
  + - API Client should include “**Token Source**” header
  + Authorization Caching
    - Enabled – to cache the policy or not
    - TTL – life span of caching, 0 means no caching
    - Min 300 to Max 3600 seconds
* Request based
  + **Identity Sources** - choose the param type
    - Header, Query String, Stage Variables, Context
    - If nothing specified, returns **401 Unauthorized**
  + Authorization Caching

Input to lambda authorizer

* It uses “**authorizationToken**”

{

"type":"TOKEN",

"authorizationToken":"allow",

"methodArn":"arn:aws:execute-api:us-west-2:123456789012:ymy8tbxw7b/\*/GET/"

}

Output from authorizer

* Its dictionary like object
* Max length 1600 bytes
  + If exceeds, 414 - too long response

Call an API with API lambda authorizers

* When calling an authorizer-enabled method, API Gateway does not log the call to CloudWatch if the required token for the **TOKEN** authorizer is not set, is null, or is invalidated by the specified **Token validation expression**.

**AWS WAF – Web Application Firewall**

* **Used to protect API from common web exploits – SQL Injection, XSS**.
* **Rules can be created to validate – HTTP headers, method, query string, URI – 8 KB size**

**Usage Plans and API Keys**

* Usage plans
  + How **fast client access**
  + **Throttle limits – request rate limits**
  + **Quota limits – maximum request limits**
* API Keys
  + APIG generates key/value pair
  + Alphanumeric
  + 30 and 128 chars
  + Can be mapped to more than one API stage and plan
* Best practices
  + **Should not rely on it to authenticate/authorize API, use IAM role instead**

# Tracing, Logging & Monitoring API

* **X-Ray**
  + Service graph
  + Illustrating latency
  + Trace API gateway requests
  + **Monitor performance (latency issues** etc).
  + X-Ray Tracing modes
    - The path of a request through your application is tracked with a **trace ID**. A trace collects all of the segments generated by a single request, typically an HTTP GET or POST request
    - Passive
      * Default setting. API is traced only if enabled on upstream service
    - Active
      * If enabled, API automatically be traced by sampling algorithm
      * API Gateway creates **AWSServiceRoleForAPIGateway**  role if not exists. It has **APIGatewayServiceRolePolicy** policy
  + X-Ray Sampling rules
    - X-Ray applies a sampling algorithm to ensure that tracing is efficient, while still providing a representative sample of the requests that your API receives. The default sampling algorithm is **1 request per second, with 5 percent of requests** sampled past that limit
* **CloudWatch**
  + **Troubleshoot API problems thru**
  + **Trace API errors**
  + **Calls against execute-api**
  + **Monitor API execution, debugging etc**
    - Statistics are recorded for a period of two weeks
    - IntegrationLatency – measure the responsiveness of the backend
    - Latency – measure overall responsiveness of API calls
    - **CacheHitCount/CacheMissCount** – metrics to optimize cache capacities
    - Count – total number of requests in a period
* **CloudTrail**
  + **API management. calls to apigateway**
  + Trail can deliver log files into S3 bucket
  + Calls to API gateway for REST API calls
  + eventSource: "apigateway.amazonaws.com",

**API Gateway limits**

* Header value are limited to 10240 bytes
* Account level limits
  + Throttle limit
    - 10000 requests per second with additional burst capacity (bucker 5000)
    - Customer can’t change this
  + Regional API – 600
  + Edge-optimized API – 120

|  |  |  |
| --- | --- | --- |
| **Resource or Operation** | **Default Limit** | **Can Be Increased** |
| Custom domain names per account per region | 30 | Yes |
| Private APIs per account per region | 600 | No |
| Length, in characters, of API Gateway resource policy | 8092 | Yes |
| API keys per account per region | 500 | Yes |
| Client certificates per account per region | 60 | Yes |
| **Authorizers per API (AWS Lambda and Amazon Cognito)** | **10** | Yes |
| Documentation parts per API | 2000 | Yes |
| Resources per API | 300 | Yes |
| **Stages per API** | **10** | Yes |
| **Usage plans per account per region** | **300** | Yes |
| **Usage plans per API key** | **10** | Yes |
| Per-method throttling limit settings per API stage | 20 | Yes |
| VPC links per account per region | 20 | Yes |
| **API caching TTL** | **300** seconds by default and configurable between 0 and 3600 by an API owner. | Not for the upper bound (3600) |
| Cached response size | 1048576 Bytes. Cache data encryption may increase the size of the item that is being cached. | No |
| **Integration timeout** | **50 milliseconds - 29 seconds for all integration types, including Lambda, Lambda proxy, HTTP, HTTP proxy, and AWS integrations.** | Not for the lower or upper bounds. |
| Header value size | 10240 Bytes | No |
| **Payload size** | **10 MB** | No |
| Tags per stage | 50 | No |
| Number of iterations in a #foreach ... #end loop in mapping templates | **1000** | No |
| ARN length of a method with authorization | **1600 bytes** | No |

# Enabling CORS for a REST API resource

Pre-requisites suits cross-origin

* Domain, Subdomain, Port, Protocol (http to https)
* Should be enabled CORS support for non-simple requests

To support CORS, therefore, a REST API resource needs to implement an **OPTIONS** method that can respond to the **OPTIONS** pre-flight request with at least the following response headers mandated by the Fetch standard:

API gateway must implement an **OPTIONS** method to support CORS.

* **Access-Control-Allow-Methods**
* **Access-Control-Allow-Headers**
* **Access-Control-Allow-Origin**

Enabling CORS support for Lambda or HTTP Non-proxy integrations

* Create OPTIONS method
* Need to return this Access-Control-Allow-Origin header in method response and integration response
* **Need to manually configure integration response to** **return Access-Control-Allow-Origin: ‘\*’ or domain**

Enabling CORS support for Lambda or HTTP proxy integrations

* Setup OPTIONS method
* Backend is responsible for,
  + Access-Control-Allow-Headers
  + Access-Control-Allow-Origin

After CORS is enabled on the GET method, an OPTIONS method is added to the resource, if it is not already there. The 200response of the **OPTIONS method is automatically configured to return the three Access-Control-Allow-\* headers to fulfill preflight handshakes**. In addition, the actual (GET) method is also configured by default to return the Access-Control-Allow-Origin header in its **200 response as well**. For **other types of responses, you will need to manually configure them to return Access-Control-Allow-Origin' header with '\*' or** specific origins, if you do not want to return the Cross-origin access error.

Proxy integration model

* HTTP Proxy
  + Pass request/response between front-end and back-end services
  + HTTP\_PROXY
* Lambda Proxy
  + AWS\_PROXY
  + Applies default mapping template
  + Send request to lambda and return lambda result to front-end

# Trace API Gateway API Execution with AWS X-Ray

**You can use**[**AWS X-Ray**](https://docs.aws.amazon.com/xray/latest/devguide/xray-services-apigateway.html)**to trace and analyze user requests as they travel through your Amazon API Gateway APIs to the underlying services.** API Gateway supports AWS X-Ray tracing for all API Gateway endpoint types: **regional, edge-optimized, and private**. You can use AWS X-Ray with Amazon API Gateway in all regions where X-Ray is available.

**X-Ray gives you an end-to-end view of an entire request, so you can analyze latencies in your APIs and their backend services. You can use an X-Ray service map to view the latency of an entire request and that of the downstream services that are integrated with X-Ray**. And you can configure sampling rules to tell X-Ray which requests to record, at what sampling rates, according to criteria that you specify. **If you call an API Gateway API from a service that's already being traced, API Gateway passes the trace through, even if X-Ray tracing is not enabled on the API.**

You can enable X-Ray for an API stage by using the API Gateway management console, or by using the API Gateway API or CLI.

# Amazon API Gateway Stage Variables Reference

Use cases

* Parameter mapping expressions
* Mapping templates
* HTTP Integration URIs
* AWS Integration URIs
* Lambda URIs
* Integration Credentials

# Parameter Mapping Expressions

A stage variable can be used in a **parameter mapping expression for an API method's request or response header parameter**, without any partial substitution. In the following example, the stage variable is referenced without the **$**and the enclosing **{...}.**

* stageVariables.<variable\_name>

Mapping Templates

A stage variable can be used anywhere in a mapping template, as shown in the following examples.

* { "name" : "$stageVariables.<variable\_name>"}
* { "name" : "${stageVariables.<variable\_name>}"}

HTTP Integration URIs

A stage variable can be used as part of an HTTP integration URL, as shown in the following examples.

* A full URI without protocol, e.g., http://${stageVariables.<variable\_name>}
* A full domain: e.g., http://${stageVariables.<variable\_name>}/resource/operation
* A subdomain: e.g., http://${stageVariables.<variable\_name>}.example.com/resource/operation
* A path, e.g., http://example.com/${stageVariables.<variable\_name>}/bar
* A query string, e.g., http://example.com/foo?q=${stageVariables.<variable\_name>}