



AWS API Gateway

Verified 2026-10-24

The managed front door for your backend — choose an API type, wire integrations, secure routes, throttle traffic, and deploy stages.

Contents

Mental model	2
Choosing an API type	2
Integrations	4
Routes and paths	6
Authorization	8
Throttling, keys, and usage plans	10
Stages, deployments, and domains	11
Errors and CORS	13
WebSocket APIs	15
Limits that bite	16
Further reading	17

Mental model

API Gateway is a managed front door. A request arrives, is matched to a route or method, passes authorization, validation and throttling, and is forwarded to an integration (Lambda, an HTTP backend, or an AWS service). It comes in three products with different feature sets, so most early mistakes are picking the wrong one, or editing a REST API and forgetting that nothing is live until you deploy it.

Choosing an API type

Type	Pick it when	Notes
HTTP API	Proxying to Lambda or HTTP, JWT auth	Cheaper, lower latency
REST API	You need keys, caching, WAF, validation	Most features

Type	Pick it when	Notes
WebSocket API	Clients and server push messages	Stateful connections

REST APIs also choose an endpoint type: edge-optimized (served through CloudFront), Regional, or private (reachable only through a VPC endpoint). HTTP and WebSocket APIs are Regional only.

Gotcha: HTTP APIs have no usage plans, API keys, response caching, request validation, WAF integration, or private endpoints. Check this list before choosing HTTP for the price, or you will migrate later.

Integrations

Integration	Backend	Notes
Lambda proxy	A Lambda function	Event passed through as is
HTTP proxy	A public HTTP URL	Request and response forwarded
AWS service	SQS, SNS, Step Functions...	REST uses mapping templates
Mock	None	REST only; canned response
VPC link	Private load balancer	Reaches resources inside a VPC

Prefer proxy integrations: API Gateway forwards the request untouched and your code shapes the response. Non-proxy integrations (REST only) transform data with

VTL mapping templates, which is powerful but hard to debug.

```
export const handler = async (event) => {  
  const { id } = event.pathParameters;  
  return {  
    statusCode: 200,  
    headers: {  
      "content-type": "application/json",  
    },  
    body: JSON.stringify({ id }),  
  };  
};
```

HTTP APIs default to payload format 2.0, where you may return a plain object and it is serialized as a 200 JSON response. REST APIs always need the `statusCode` / `headers` / `body` shape above.

Gotcha: A Lambda proxy response with a bad shape (body as an object, a missing statusCode) returns **502 Bad Gateway** with “Malformed Lambda proxy response”. Always `JSON.stringify` the body.

Routes and paths

API	Route form	Catch-all
REST	Resource <code>/items/{id}</code> plus a method	<code>{proxy+}</code>
HTTP	Route key <code>GET /items/{id}</code>	<code>\$default</code>
WebSocket	Route key from a message field	<code>\$default</code>

A `{name}` segment captures one path segment. A greedy `{proxy+}` captures the rest of the path and must be the last segment. In HTTP APIs the most specific matching route wins, and `$default` catches whatever matches nothing else.

Transform: AWS::Serverless-2016-10-31

Resources:

Fn:

Type: AWS::Serverless::Function

Properties:

Runtime: nodejs22.x

Handler: index.handler

CodeUri: src/

Events:

Get:

Type: HttpApi

Properties:

Path: /items/{id}

Method: GET

Gotcha: In a REST API, a `/ {proxy+}` resource does not match the root `/`. Add a method on the root resource too, or `GET /` fails.

Authorization

Option	API types	Use for
IAM (SigV4)	REST, HTTP, WebSocket	AWS callers, service-to- service
Cognito user pool	REST	Cognito-issued tokens
JWT authorizer	HTTP	Any OIDC or OAuth 2.0 issuer
Lambda authorizer	REST, HTTP, WebSocket	Custom token or header logic

A REST Lambda authorizer returns an IAM policy that allows or denies `execute-api:Invoke`. API Gateway caches that policy per identity source for a TTL (default 300 seconds, up to 3600). The optional `context` map is passed to your backend, and its values must be strings, numbers, or booleans.

```
export const handler = async (event) => {  
  const t = event.authorizationToken;  
  const ok = t === process.env.TOKEN;  
  return {  
    principalId: "user",  
    policyDocument: {  
      Version: "2012-10-17",  
      Statement: [{  
        Action: "execute-api:Invoke",  
        Effect: ok ? "Allow" : "Deny",  
        Resource: event.methodArn,  
      }],  
    },  
  };  
};
```

HTTP API Lambda authorizers can instead use the simple response `{ isAuthorized: true }` when simple responses are enabled.

Gotcha: With caching on, a policy scoped to `event.methodArn` is reused for the same token on every other route, and those calls fail with 403. Return a wildcard resource for the API, or set the TTL to 0.

Throttling, keys, and usage plans

Level	Scope	Default
Account	All APIs in a Region	10,000 rps, 5,000 burst
Stage or method	One API	Set per stage or method
Usage plan	Per API key (REST only)	Rate, burst, and quota

Throttling uses a token bucket: the rate refills tokens and the burst is the bucket size. A throttled request gets **429 Too Many Requests**. Some Regions (for example

Cape Town, Milan, Jakarta) default to 2,500 rps and 1,250 burst.

```
aws apigateway create-usage-plan \  
  --name basic \  
  --throttle burstLimit=20,rateLimit=10 \  
  --quota limit=1000,period=DAY \  
  --api-stages apiId=abc123,stage=prod
```

Gotcha: API keys meter traffic, they do not authenticate it. A method only checks them when `apiKeyRequired` is on, and a missing or invalid key returns **403 Forbidden**. Pair keys with a real authorizer.

Stages, deployments, and domains

Concept	REST	HTTP
Go live	Create a deployment	Auto-deploy option

Concept	REST	HTTP
Default stage	Named by you	<code>\$default</code>
Stage variables	Yes	Yes
Canary releases	Yes	No

The invoke URL is `https://{api-id}.execute-api.{region}.amazonaws.com/{stage}$default`. The `$default` stage drops the stage segment from the path. Stage variables (such as a Lambda alias) are referenced as `${stageVariables.name}`.

```
aws apigateway create-deployment \  
  --rest-api-id abc123 \  
  --stage-name prod
```

Custom domains need an ACM certificate: in the API's Region for Regional APIs, and in `us-east-1` for edge-optimized REST APIs. API mappings attach a stage to a domain path.

Gotcha: Editing a REST API changes nothing live until you create a new deployment to the stage. Run `create-deployment` (or `redeploy` in the console) after every change.

Errors and CORS

Status	Typical cause
400	Request failed REST validation
403	Authorizer denied, bad API key, WAF block
429	Throttled
502	Malformed or failed Lambda proxy response
504	Integration exceeded its timeout

REST request validators check required parameters, headers, and body schemas, and reject bad requests with 400 before your backend runs. Integration timeouts return 504 once they pass the limit (29 seconds by default on REST).

HTTP APIs have built-in CORS configuration. For REST proxy integrations, API Gateway answers the `OPTIONS` preflight but your function must return the CORS headers itself.

```
const ORIGIN = "https://app.example.com";

export const handler = async () => ({
  statusCode: 200,
  headers: {
    "access-control-allow-origin": ORIGIN,
  },
  body: "{}",
});
```

Gotcha: 403 “Missing Authentication Token” on a REST API usually means the path, method, or stage does not exist, not that you forgot a token. Check the URL, the HTTP verb, and the deployment.

WebSocket APIs

Route	Fires when	Use for
<code>\$connect</code>	Connection opens	Authenticate, store the ID
<code>\$disconnect</code>	Connection closes	Clean up stored IDs
<code>\$default</code>	No route matches	Fallback handler
Custom key	Selection expression matches	Dispatch by message action

The API-level route selection expression picks a JSON property from each message (for example `${request.body.action}`) and matches its value to a route key. Your backend pushes data to a client by sending a request to

```
POST https://{api-id}.execute-api.  
{region}.amazonaws.com/{stage}/@connections/  
{connectionId}
```

, signed

with SigV4. A `GET` on the same URL reads connection status and a `DELETE` closes it.

Gotcha: Connections close after 10 idle minutes or a 2-hour lifetime, whichever comes first. Clients must reconnect, and you must remove stale connection IDs, or posts to them fail.

Limits that bite

Limit	REST	HTTP
Integration timeout	29 s default	30 s max
Payload size	10 MB	10 MB
Routes or resources	300 resources	300 routes
Stages per API	10	10
Authorizers per API	10	10

Most are adjustable except the payload size and the HTTP API timeout. REST Regional and private APIs can raise the integration timeout above 29 seconds, but that may cost you account-level throttle quota. Lambda's

own 15-minute limit does not help: the gateway gives up first, so move long work behind a queue or an async invocation.

Further reading

- [API Gateway Developer Guide](#)
- [Choosing between REST and HTTP APIs](#)
- [REST API quotas](#)
- [HTTP API quotas](#)
- [Lambda authorizers](#)