aws

AWS Events API Developer Guide

AWS Events API

AWS Events API: AWS Events API Developer Guide

Table of Contents

What is the AWS Events API?

> **Note**
>
> Reserved seating for AWS re:Invent opens on October 6, 2026, and opens through this API on October 8, 2026. Until then, reserving and canceling return 409. Reading the catalog and marking favorites work as normal.

AWS re:Invent, AWS Summits, and other AWS conferences publish session catalogs listing talks, workshops, and labs.

With this API you can list events, read a session catalog, and manage your own schedule: reserve sessions, mark favorites, and add personal time.

You sign in on your own machine. There is no hosted option. An application you distribute runs on each attendee's machine and signs that person in. The API always acts for the attendee who is signed in.

You can use the API two ways. Both offer the same features.

- A *REST API*, for applications and scripts. For more information, see *Using the REST API*.
- A *Model Context Protocol (MCP) server*, for AI assistants and agents. An MCP client finds the available tools and calls them for you. You can ask an assistant to find sessions or build a schedule. Unlike the REST API, the MCP server requires sign-in for every call, even catalog reads. For more information, see *Using the MCP server*.

Which events you can read, and when you need to sign in

Whether an event's session catalog is public depends on whether that event requires registration. That same requirement decides whether there is a schedule to manage.

- *Events that do not require registration*: the session catalog is public. Anyone can read it with no credentials. These events have no schedule to manage.
- *Events that require registration*: the session catalog is not public. Reading it needs an attendee who has signed in and is registered for that event. Those attendees can also read and change

their own schedule: make and cancel reservations, add and remove favorites, and manage personal time.

`ListEvents` and `GetEvent` need no credentials for any event. An event that requires registration still appears in the list even when its sessions are not readable.

What the API does not do

- It does not register you for an event. It does not return a registration link. Register through the event's own registration site first.
- It does not expose other attendees, or any data belonging to them.
- It does not guarantee a reservation. A request can be refused for one session while succeeding for others. For more information, see the section called “Reading the result”.
- It does not search or filter the catalog. You retrieve an event's sessions and filter them in your own application.

Getting started

Listing events needs no credentials. Whether you can go on to read an event's sessions depends on that event.

List events without signing in

Every event currently running or upcoming:

```
curl https://api.awsevents.com/v1/events
```

To include events that have already ended, add `includePast=true`.

Each entry reports whether that event requires registration. For an event that does not, read its sessions the same way, a page at a time:

```
curl https://api.awsevents.com/v1/events/$EVENT_ID/sessions
```

Pages can come back short. Only an absent `nextToken` means the end. See the section called "Paginating sessions".

> **Note**
>
> An event that requires registration has no public catalog: its sessions return `401` to an anonymous caller. The event itself still appears in `ListEvents`.

Sign in to read a registration event and manage your schedule

For an event that requires registration, both its session catalog and your own schedule need an access token. You sign in with your AWS Builder ID. For more information, see *Authentication*. In outline, the process is:

1. Register for the event through its registration site, if you have not already.
2. Sign in with your Builder ID. Obtain an access token.
3. Send the token as a bearer token on each request:

```
curl -H "Authorization: Bearer $ACCESS_TOKEN" \
  https://api.awsevents.com/v1/events/reinvent2026/schedule
```

If the token is valid but you are not registered for that event, the response is `403` rather than `401`. Signing in again will not fix it. For more information, see the section called “Access token and event registration”.

Or connect an AI assistant

If you would rather work through an AI assistant than write requests, point an MCP client at the server. It will discover the available tools itself. For more information, see *Using the MCP server*.

Authentication

Listing events, and reading the catalog of an event that does not require registration, need no credentials. For everything else the attendee signs in with an AWS Builder ID, the free personal sign-in that AWS uses for its developer tools and events, and your application sends the resulting OAuth 2.0 access token as a bearer token:

```
curl -H "Authorization: Bearer $ACCESS_TOKEN" \
  https://api.awsevents.com/v1/events/reinvent2026/schedule
```

Topics

Access token and event registration

To read the catalog of an event that requires registration, or to reach your own schedule, you need both of the following. Each fails in its own way.

- *A valid access token.* A missing, expired, or malformed token produces `401`. The `WWW-Authenticate` header names where to sign in.
- *Registration for that event.* A valid token from someone not registered for the event produces `403`. One token covers every event you are registered for. The same token can succeed for one event and return `403` for another.

Registration happens on the event's own site. You cannot fix a `403` by signing in again.

Endpoints and values

Sign-in uses the following endpoints and values. They are the same for every caller. The client ID is public and shared. The flow is protected by Proof Key for Code Exchange (PKCE), not a client secret, so there is no secret to hold.

Authentication endpoints and values

Item	Value
Authorization endpoint	`https://oauth.awsevents.com/oauth2/authorize`
Token endpoint	`https://oauth.awsevents.com/oauth2/token`
Revocation endpoint	`https://oauth.awsevents.com/oauth2/revoke`
Logout endpoint	`https://oauth.awsevents.com/logout`
Builder ID sign-out endpoint	`https://idp.awsevents.com/oidc/logout`
Client ID	`7vmom55m1qstvq8i71ph127bfq`
Scope	`openid email events/access`
Identity provider	`AWSBuilderID`
Access token lifetime	60 minutes
Refresh token lifetime	30 days

Note

Always send `identity_provider=AWSBuilderID` on the authorization request. Without it, the attendee sees an extra page that asks which provider to use. Builder ID is the only choice.

Signing an attendee in

Sign-in is the OAuth 2.0 authorization code flow with PKCE.

Your application must run on the attendee's machine. It listens at `/callback` on one of six reserved loopback ports, 8484 through 8489. Both host forms are registered for each, so nothing needs registering, and the following examples use `http://localhost:8484/callback`. `redirect_uri` is matched exactly, with no wildcards. Any other port fails for an application you build. Installed MCP clients have their own registered callbacks; see the section called “Connecting a client”. The authorization request and the token exchange must send the identical string. There is no hosted redirect URI, so you can distribute an application that signs attendees in, but it has to run locally.

1. Generate a PKCE code verifier from a cryptographically secure random source, 43 to 128 characters from the unreserved set, fresh for every sign-in attempt. Derive the code challenge by taking the SHA-256 digest of the verifier and encoding it as base64url with no padding. Generate a random `state` value and store it with the verifier.
2. Redirect the attendee's browser to the authorization endpoint:

```
https://oauth.awsevents.com/oauth2/authorize
  ?response_type=code
  &client_id=7vmom55m1qstvq8i71ph127bfq
  &redirect_uri=http://localhost:8484/callback
  &scope=openid+email+events/access
  &identity_provider=AWSBuilderID
  &code_challenge=$CODE_CHALLENGE
  &code_challenge_method=S256
  &state=$STATE
```

3. The attendee signs in with their Builder ID. Their browser returns to your redirect URI with `code` and `state` query parameters.
4. Check that `state` matches what you stored, then exchange the code for tokens:

```
curl -X POST https://oauth.awsevents.com/oauth2/token \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d "grant_type=authorization_code" \
  -d "client_id=7vmom55m1qstvq8i71ph127bfq" \
  -d "redirect_uri=http://localhost:8484/callback" \
  -d "code=$CODE" \
```

```
  -d "code_verifier=$CODE_VERIFIER"
```

> **Note**
>
> Send `code_challenge_method=S256` explicitly, and send the challenge base64url-encoded. A hex or standard-base64 challenge, or a verifier reused between attempts, fails the exchange with an error that does not say which value was wrong.

The response returns three tokens:

- *Access token* — a JSON Web Token (JWT) valid for 60 minutes. This is the only one you send to this API.
- *ID token* — describes who signed in, for your application's own use. This API does not accept it in place of the access token.
- *Refresh token* — opaque, valid for 30 days, and used only against the token endpoint.

Keeping the attendee signed in

When the access token expires, exchange the refresh token for a new one, rather than a fresh sign-in:

```
curl -X POST https://oauth.awsevents.com/oauth2/token \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d "grant_type=refresh_token" \
  -d "client_id=7vmom55m1qstvq8i71ph127bfq" \
  -d "refresh_token=$REFRESH_TOKEN"
```

Schedule the next refresh against the response's `expires_in`, not a hardcoded constant. If the response carries a new refresh token, replace the one you stored. The previous one may not work again.

Treat a `401` as "refresh once, then retry the request." If the retry also returns `401`, the refresh token is no longer good. Start sign-in again rather than retrying in a loop.

> **Note**
>
> A refresh provides a new valid access token. It does not extend the Builder ID sign-in session, which has its own lifetime. A long-running application will eventually need an interactive sign-in, however recently it refreshed.

Signing an attendee out

Discarding your copy of the tokens is not a sign-out. Three pieces of state are involved: your tokens, the AWS Builder ID browser session, and the session that brokers it. A complete sign-out clears all three.

1. Revoke the refresh token, then delete both tokens from wherever you keep them:

   ```
   curl -X POST https://oauth.awsevents.com/oauth2/revoke \
     -H "Content-Type: application/x-www-form-urlencoded" \
     -d "client_id=7vmom55m1qstvq8i71ph127bfq" \
     -d "token=$REFRESH_TOKEN"
   ```

   Revoking stops the refresh token and any new access token it can mint. It does not invalidate an access token already issued. That token is accepted until it expires, so discard every copy your application holds.

2. End both browser sessions. Build the URL in two parts. The inner URL ends the brokering session and returns to your application:

   ```
   https://oauth.awsevents.com/logout
     ?client_id=7vmom55m1qstvq8i71ph127bfq
     &logout_uri=http://localhost:8484/logout
   ```

   Percent-encode that whole URL as the `redirect_uri` value on the Builder ID sign-out endpoint, and send the attendee there:

   ```
   https://idp.awsevents.com/oidc/logout?redirect_uri=encoded-url
   ```

   That one navigation clears both sessions and lands back on your application. The Builder ID endpoint has to come first: it accepts a redirect back to the sign-in domain, not to your

application. `logout_uri` is exact-matched like `redirect_uri`, but against a separate list of sign-out URLs — `/logout` on the same reserved ports, in both host forms.

> **Important**
>
> Both endpoints are required. Skip the Builder ID one and the attendee appears to sign out, then is signed straight back in without a prompt. You cannot force a prompt instead: `prompt=login` and `max_age=0` are dropped from authorization requests. It does not reproduce unless you test sign-out and sign-in in the same browser.

Offer both steps separately. Clearing your own tokens signs the attendee out of your application. The second step also ends their AWS Builder ID session in that browser, which is not always what they want.

> **Note**
>
> An attendee can also end the Builder ID session themselves at `https://profile.aws.amazon.com`, which offers signing out of the current browser or signing out everywhere. Point them there when your application cannot drive the redirect chain, such as from a command line tool.

Handling tokens safely

- Send the access token only to `https://api.awsevents.com`. If you send it elsewhere, you disclose it to that party.
- Do not log tokens, put them in URLs, or store them where other code on the device can read them. A token identifies one attendee. Anything holding it can act on that attendee's schedule.
- In a browser application, keep both tokens in memory only. To survive a page reload, run the authorization request again rather than storing a token.
- Elsewhere, use the operating system keychain or a file only the user can read. Keep server-side tokens server side. Treat the refresh token as the more sensitive of the two. It can mint new access tokens long after the original expired.

Using the REST API

All requests go to `https://api.awsevents.com`. Every path is versioned and begins with `/v1`. Requests and responses are JSON.

Reading a registration event's catalog, and every operation on your own schedule, needs a bearer token from an attendee registered for that event. For more information, see *Authentication*. Examples use `reinvent2026`, which requires registration, and shell variables for the IDs and token you supply.

> **Note**
>
> The full OpenAPI description of these operations is served at `https://api.awsevents.com/v1/openapi.json`.

Topics

ListEvents

Lists AWS events. By default, it returns running and upcoming events. Set `includePast=true` to also include events that have ended. No sign-in is needed. Use the `eventId` in each entry for `{eventId}` in the other paths in this chapter.

```
GET /v1/events
```

```
curl "https://api.awsevents.com/v1/events?includePast=true"
```

GetEvent

Gets one event, with the same fields as a list entry. Not filtered by date, so a link to an event keeps working after it ends. No sign-in is needed.

```
GET /v1/events/{eventId}
```

```
curl https://api.awsevents.com/v1/events/reinvent2026
```

ListSessions

Lists an event's sessions, a page at a time. The service sets the page size. Pass `nextToken` to continue. Every page reports `totalCount`, the number of sessions in the catalog, so you can size a walk from the first page. Set `includeAbstracts=false` to omit each session's `abstract`, the largest field, when you only need enough to choose between sessions. An optional `locale` query parameter takes a language tag. If it is unavailable or omitted, the catalog is served in `en-US`. The response reports what was actually served. No sign-in is needed, unless the event requires registration.

```
GET /v1/events/{eventId}/sessions
```

```
curl https://api.awsevents.com/v1/events/reinvent2026/sessions
```

To fetch the next page, pass the `nextToken` from the previous response:

```
curl "https://api.awsevents.com/v1/events/reinvent2026/sessions?nextToken=$NEXT_TOKEN"
```

What a session contains

`ListSessions` and `GetSession` return the same session shape. A session carries a `sessionId`, a title, an abstract, and a short session code; its type and level, and taxonomy lists such as tracks, topics, industries, roles, and services; its start, end, room, and venue, and whether it runs all day; whether it can be reserved, and when it can, a coarse indication of how full it is; and the names of the speakers presenting it.

A field is present only when the event provides it. Treat any single field as optional, and read defensively.

Paginating sessions

> **Note**
>
> Page sizes can vary, and a short page does not mean the last page. Stop when `nextToken` is absent, never when a page looks short. A client that stops early silently misses sessions.

GetSession

Gets one session in an event. The session has the same shape as a list entry. See the section called "What a session contains". Session IDs come from `ListSessions`. No sign-in is needed, unless the event requires registration.

```
GET /v1/events/{eventId}/sessions/{sessionId}
```

```
curl https://api.awsevents.com/v1/events/reinvent2026/sessions/$SESSION_ID
```

GetSchedule

Gets your own schedule for an event. This includes the sessions you reserved, your favorites, and your personal time entries. You must sign in.

It is the source of truth for your own data. Read it back after any change. This confirms the result and returns IDs the write operations do not.

```
GET /v1/events/{eventId}/schedule
```

```
curl -H "Authorization: Bearer $ACCESS_TOKEN" \
  https://api.awsevents.com/v1/events/reinvent2026/schedule
```

ReserveSessions

Reserves one or more sessions for you. Send the session IDs in the request body. One request carries 1 to 10 session IDs. They must be distinct. You must sign in.

```
POST /v1/events/{eventId}/reservations
```

```
curl -X POST https://api.awsevents.com/v1/events/reinvent2026/reservations \
  -H "Authorization: Bearer $ACCESS_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"sessionIds": ["'$SESSION_ID'"]}'
```

Reading the result

The response reports each session independently, as succeeded or failed. Each failure names the session and the reason: the session is full, its time conflicts with something already on your schedule, or you already have it.

> **Important**
>
> A `200` does not mean everything succeeded. Always read the failures. A session that was already reserved counts as a failure too, so re-sending the same request is not a safe retry.

CancelReservation

Cancels one of your reservations, by session ID. If you have none for that session, the call returns `404`. So a `404` on a retry means it was already gone. You must sign in.

```
DELETE /v1/events/{eventId}/reservations/{sessionId}
```

```
curl -X DELETE -H "Authorization: Bearer $ACCESS_TOKEN" \
```

```
  https://api.awsevents.com/v1/events/$EVENT_ID/reservations/$SESSION_ID
```

AssociateFavorites

Marks one or more sessions as favorites, 1 to 10 of them and all distinct. A favorite records interest only. It does not reserve the session. The result is per session, exactly as for the section called "ReserveSessions". Read the failures, and do not treat a repeat as a safe retry. You must sign in.

```
POST /v1/events/{eventId}/favorites
```

```
curl -X POST https://api.awsevents.com/v1/events/reinvent2026/favorites \
  -H "Authorization: Bearer $ACCESS_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"sessionIds": ["'$SESSION_ID'"]}'
```

DisassociateFavorite

Removes one session from your favorites. Give the session ID in the path. If it is not there, the call returns `404`. You must sign in.

```
DELETE /v1/events/{eventId}/favorites/{sessionId}
```

```
curl -X DELETE \
  -H "Authorization: Bearer $ACCESS_TOKEN" \
  https://api.awsevents.com/v1/events/reinvent2026/favorites/$SESSION_ID
```

CreatePersonalTime

Adds a block of time that is your own, rather than a session — travel, a meeting, a break. You must sign in.

```
POST /v1/events/{eventId}/personal-time
```

```
curl -X POST https://api.awsevents.com/v1/events/reinvent2026/personal-time \
  -H "Authorization: Bearer $ACCESS_TOKEN" \
```

```
  -H "Content-Type: application/json" \
  -d '{
    "title": "Booth duty",
    "description": "Staffing the partner booth",
    "startDateTime": "2026-12-01T19:00:00",
    "endDateTime": "2026-12-01T21:00:00",
    "location": "Expo hall"
  }'
```

Note

Create returns no body. To obtain the new entry's ID, read it back from the section called "GetSchedule".

Personal time fields

The following table describes the fields on a personal time entry.

Personal time fields

Field	Required	Notes
`title`	Yes	1–128 characters.
`description`	Yes	1–250 characters.
`startDateTime`	Yes	UTC, formatted `YYYY-MM-DDTHH:mm:ss`, with no offset or trailing `Z`.
`endDateTime`	Yes	Same format. The duration from start to end must be a whole number of 5-minute increments.
`location`	No	Up to 255 characters.

UpdatePersonalTime

Replaces a personal time entry that you already created. You must sign in.

Note

This replaces the entry. It does not merge. Resend every field, even the ones you are not changing. For the field list, see the section called "Personal time fields". If you omit `location`, it is cleared. The entry keeps its `personalTimeId`.

```
PUT /v1/events/{eventId}/personal-time/{personalTimeId}
```

```
curl -X PUT \
  https://api.awsevents.com/v1/events/reinvent2026/personal-time/$PERSONAL_TIME_ID \
  -H "Authorization: Bearer $ACCESS_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "title": "Booth duty",
    "description": "Staffing the partner booth",
    "startDateTime": "2026-12-01T19:30:00",
    "endDateTime": "2026-12-01T21:30:00",
    "location": "Expo hall"
  }'
```

DeletePersonalTime

Removes a personal time entry. Give its ID in the path. IDs come from the section called "GetSchedule". You must sign in.

```
DELETE /v1/events/{eventId}/personal-time/{personalTimeId}
```

```
curl -X DELETE \
  -H "Authorization: Bearer $ACCESS_TOKEN" \
  https://api.awsevents.com/v1/events/reinvent2026/personal-time/$PERSONAL_TIME_ID
```

Using the MCP server

The MCP server offers the same operations as the REST API, as tools that an AI assistant can call.

Connecting a client

The server endpoint is `https://api.awsevents.com/mcp`, over streamable HTTP. Point a client at that URL. Where a client lets you set a client ID, use `7vmom55m1qstvq8i71ph127bfq`, which is shared by all callers.

The server requires sign-in, and the client handles that flow for you. On the first call it receives a challenge naming where to sign in, opens that page in a browser, and stores the resulting token. The sign-in itself is the flow described in *Authentication*.

Kiro

Add the server to your MCP configuration file:

```
{
  "mcpServers": {
    "awsevents": {
      "url": "https://api.awsevents.com/mcp",
      "oauth": {
        "clientId": "7vmom55m1qstvq8i71ph127bfq",
        "redirectUri": "http://127.0.0.1:8976",
        "oauthScopes": ["openid", "email", "events/access"]
      }
    }
  }
}
```

End `redirectUri` at the port, as shown. Kiro reads the port from the last colon onward and adds the `/oauth/callback` path itself. Give it a value with a path and it cannot read the port, so it falls back to a random one and sign-in fails with `redirect_mismatch`. Reload the window after editing the file.

Claude Code

```
claude mcp add --transport http --scope user awsevents \
  https://api.awsevents.com/mcp \
```

```
  --callback-port 8484 --client-id 7vmom55m1qstvq8i71ph127bfq
```

Important

Use the preceding callback values, and always set the port. Both clients pick a random port otherwise, and a random port cannot sign in. The API matches a callback in full, port and path included, with no wildcards, and one it does not recognize fails without reporting the callback as the cause.

Note

The client stores your token on the machine it runs on, and not every client protects it well. Deleting a client's configuration does not invalidate the token it already holds. An MCP client has no redirect chain to sign you out with, so to sign out fully, revoke the token as described in the section called “Signing an attendee out”, then end the Builder ID session at `https://profile.aws.amazon.com`.

To confirm the connection, ask the assistant to list events.

Available tools

Each tool maps to one REST operation. The server itself requires sign-in, so every tool call is made as a signed-in attendee, including the catalog tools that need no credentials on the REST API. Reading the catalog of an event that requires registration, and anything that touches a schedule, additionally requires that the attendee be registered for that event.

MCP tools

Tool	Description
ListEvents	Lists AWS events that are running or upcoming, such as AWS re:Invent.
GetEvent	Gets one event by ID.
ListSessions	Lists an event's sessions, a page at a time.

Tool	Description
GetSession	Gets one session within an event.
GetSchedule	Gets your own schedule for an event.
ReserveSessions	Reserves one or more sessions.
CancelReservation	Cancels one of your reservations.
AssociateFavorites	Marks one or more sessions as favorites.
DisassociateFavorite	Removes one session from favorites.
CreatePersonalTime	Adds a personal time entry.
UpdatePersonalTime	Replaces a personal time entry.
DeletePersonalTime	Removes a personal time entry.

What to expect from an assistant

- Per-operation quotas apply as on the REST API, and an operation can be closed even when its tool is listed. For more information, see *Quotas and throttling* and the section called “When an operation is closed”.
- Reserving and favoriting succeed per session, not per request, so an assistant can report "done" when some sessions failed. Ask it to confirm from `GetSchedule`.

Quotas and throttling

Each operation in the following table has its own quota on how many requests you can make per minute.

Per-attendee quotas

Requests per minute, by operation

Operation	Quota per minute
GetSession	120
ListSessions	60
GetSchedule	60
ReserveSessions	30 sessions
CancelReservation	30
AssociateFavorites	30 sessions
DisassociateFavorite	30
CreatePersonalTime	30
UpdatePersonalTime	30
DeletePersonalTime	30

Note

`ReserveSessions` and `AssociateFavorites` count each session named in the request, not each request. A batch is no cheaper than one request per session. It only saves round trips.

Handling throttling

When you exceed a quota, the API returns 429 with a `Retry-After` header giving the seconds to wait, which is the time left in the current minute before the quota resets. Wait that long before retrying. If a batch is larger than the quota you have left, make it smaller and retry right away, because a refused request spends none of the quota.

Handling errors

The API returns errors with an HTTP status code and a JSON body. The body contains a `message`.

Error reference

Errors returned by the AWS Events API

Status	Meaning	What to do
`400`	The API could not accept the request as sent. A field is missing, too long, or malformed. The API also returns this for a `nextToken` it did not issue.	Do not retry. Fix the request. The message says which field.
`401`	The operation needs a signed-in attendee, but you sent no valid token. The `WWW-Authenticate` header says where to sign in. An event that requires registration has no public catalog. For that event, `ListSessions` and `GetSession` return `401` to an anonymous caller.	Sign in, or refresh the token, then retry once. See the section called "Keeping the attendee signed in".
`403`	Two different causes, told apart by the body. With a JSON body, the caller is signed in but not registered for this event. That includes reading the catalog of an event that requires registration, so an unregistered caller gets `403` from `ListSessions` and `GetSession` too. With no body, the refusal came from the edge rather than the API, and requests from your address are being refused.	For a registration problem, do not retry. Register for the event first, because signing in again will not help. For an edge refusal, slow down, then retry.

Status	Meaning	What to do
404	The event, session, or personal time entry does not exist. For a removal, it is already gone.	Do not retry. For removals, treat it as already done.
409	The operation is not currently available. The request itself was valid.	Do not retry immediately. See the section called "When an operation is closed".
429	The caller has used up this operation 's quota for the current minute.	Wait for `Retry-After` seconds, then retry. See *Quotas and throttling*.
500	The service hit an internal error. It could not process the request.	Retry a read with backoff. Before retrying a write, reconcile from `GetSchedule` . See the section called "Retrying a write safely". If it persists, report it using the *Feedback* link on this page.
503	The service is temporarily unavailable.	Retry a read with backoff. For a write, see the section called "Retrying a write safely".

When an operation is closed

An operation can close for a time while the rest of the API keeps working. A closed operation returns 409 to every caller. The request itself was not the problem. The same request will not succeed until the operation opens.

Reservations might be closed until reserved seating opens for an event. Browsing and favoriting keep working.

Retrying a write safely

The API has no idempotency key. Re-sending a create makes a new one. This matters for the two bulk operations and for `CreatePersonalTime`. If you never learned the outcome of one of those

— a timeout, a dropped connection, a `500` or a `503` — do not simply re-send it. Read the section called “GetSchedule” and compare it with what you intended. Send only what is still missing.

The single removals are different. Canceling a reservation or removing a favorite that is already gone returns `404`, so a retry whose response you never saw is safe: treat `404` as already done.

A `429` and a `409` are also safe to retry later, because neither performed the write.

Document history for the AWS Events API

The following table describes the documentation releases for the AWS Events API Developer Guide.

Document history

Change	Description	Date
Initial release	Initial release of the AWS Events API Developer Guide.	September 21, 2026