AWS EventBridge

Verified 2026-11-02

Route JSON events with EventBridge — event patterns and operators, targets, retries and DLQs, Pipes, Scheduler, replay, cross-account buses and cost.

Contents

Mental model 2

Events and buses 2

Event patterns 3

Comparison operators 4

Targets, transformers and retries 5

Pipes 6

Scheduler 7

Archive, replay and cross-account 8

The enhanced Custom Event Bus 8

Quotas and cost 9

Further reading 10

Mental model

An event bus is a router: producers publish JSON events, and rules compare each event against JSON patterns and forward matches to targets. Producers never know who consumes their events, so adding a consumer is a new rule, not a code change. The pattern language decides what reaches whom, which makes it the part of EventBridge you actually need to master.

Events and buses

Field	Set by
`source`	Producer, e.g. `shop.orders`
`detail-type`	Producer, e.g. `OrderPlaced`
`detail`	Producer: the JSON payload
`id`, `time`, `account`, `region`	EventBridge

Every account has a `default` bus that receives AWS service events, such as EC2 state changes, for free. Create a custom bus for your own application events so their rules and permissions stay separate. Delivery is at-least-once, so targets must tolerate duplicates.

```
aws events put-events --entries '[{
  "EventBusName": "shop",
  "Source": "shop.orders",
  "DetailType": "OrderPlaced",
```

```
  "Detail": "{\"id\":\"o-9\",\"total\":120}"
}]'
```

> **Gotcha:** `PutEvents` can fail some entries and still succeed as a call. Check `FailedEntryCount`, then resend only the entries whose result carries an `ErrorCode`.

Event patterns

Pattern shape	Matches when
`"source": ["a", "b"]`	Field equals any listed value
Several fields	All fields match (AND)
Nested object	Path into `detail`
Event field is an array	Any element matches

A pattern mirrors the event's structure, and every leaf is an array of allowed values. Fields not in the pattern are ignored, so a pattern only ever narrows. Match on `source` and `detail-type` first, then on `detail` fields.

```
{
  "source": ["shop.orders"],
  "detail-type": ["OrderPlaced"],
  "detail": {
    "currency": ["EUR", "USD"]
```

```
  }
}
```

> **Gotcha:** A target whose action emits an event that matches its own rule fires that rule again, in a loop that bills every round. Make patterns precise enough that the target's output cannot match.

Comparison operators

Operator	Example
`prefix` / `suffix`	`{"suffix": ".png"}`
`anything-but`	`{"anything-but": ["test"]}`
`numeric`	`{"numeric": [">", 0, "<=", 100]}`
`exists`	`{"exists": false}`
`wildcard`	`{"wildcard": "img/*.png"}`
`equals-ignore-case` / `cidr`	`{"cidr": "10.0.0.0/24"}`

Operators sit inside the value array, so you can mix them with plain values. `anything-but` also accepts `prefix`, `suffix` and `wildcard`. Pipes support a subset: no `wildcard`, and no `anything-but` combined with prefix, suffix, ignore-case or wildcard.

```
{
  "detail": {
    "total": [{ "numeric": [">=", 100] }],
    "country": [{ "anything-but": ["XX"] }],
```

```
    "coupon": [{ "exists": false }]
  }
}
```

Targets, transformers and retries

Setting	Value
Targets per rule	5, not adjustable
Default retry	24 hours, 185 attempts
`MaximumRetryAttempts`	0–185
`MaximumEventAgeInSeconds`	60–86,400

A target receives the whole event unless an input transformer reshapes it: `InputPathsMap` extracts values with JSON paths, and `InputTemplate` places them with `<name>` placeholders. An event that exhausts its retries goes to the target's dead-letter queue, or is dropped if none is set.

```
{
  "InputPathsMap": {
    "id": "$.detail.id",
    "total": "$.detail.total"
  },
  "InputTemplate":
    "{\"orderId\": <id>, \"amount\": <total>}"
}
```

> **Gotcha:** The DLQ is an SQS queue whose policy must allow `events.amazonaws.com` to call `sqs:SendMessage`. Without it, undeliverable events are lost silently instead of parked.

Pipes

Stage	Options
Source	SQS, Kinesis, DynamoDB Streams, MSK, Kafka, MQ
Filter	Event pattern; filtered events are free
Enrichment	Lambda, API Gateway, API destination, Express workflow
Target	One AWS service, e.g. a bus or a queue

A pipe is point-to-point: one polled source, one target, with optional filtering and synchronous enrichment in between. It replaces the glue Lambda that reads a queue or stream and forwards it. Enrichment can drop an event by returning an empty response.

```
aws pipes create-pipe --name orders-pipe \
  --role-arn "$PIPE_ROLE" \
  --source "$QUEUE_ARN" --target "$BUS_ARN" \
  --source-parameters '{"FilterCriteria":
    {"Filters":[{"Pattern":
      "{\"body\":{\"type\":[\"order\"]}}"}]}}'
```

Scheduler

Expression	Runs
`at(2026-12-01T09:00:00)`	Once
`rate(5 minutes)`	Every interval
`cron(0 9 ? * MON-FRI *)`	Six-field cron
`--action-after-completion DELETE`	Removes a finished schedule

EventBridge Scheduler invokes a target directly on a schedule, with time zones, flexible time windows, and templated or universal (any API) targets. It is a separate service from scheduled rules, which the docs now call legacy.

```
aws scheduler create-schedule --name remind-o9 \
  --schedule-expression "at(2026-12-01T09:00:00)" \
  --schedule-expression-timezone Europe/Kyiv \
  --flexible-time-window '{"Mode":"OFF"}' \
  --target "{\"Arn\":\"$FN_ARN\",
    \"RoleArn\":\"$ROLE_ARN\"}" \
  --action-after-completion DELETE
```

> **Gotcha:** A completed one-time schedule still counts against your schedule quota until it is deleted. Set `DELETE` on one-off reminders.

Archive, replay and cross-account

Feature	Rule
Archive	Captures events from one bus, with retention
Replay	Back to the source bus only
Concurrent replays	10 per account per Region
Cross-account	Bus resource policy plus sender rule

Archive a bus to rebuild a consumer's state or re-run events after a bug fix. EventBridge adds a `replay-name` field to replayed events, so consumers can tell them apart. To send cross-account, the receiving bus allows `events:PutEvents` in its resource policy, and the sender adds a rule whose target is that bus, with an IAM role.

```
aws events start-replay --replay-name fix-81 \
  --event-source-arn "$ARCHIVE_ARN" \
  --event-start-time 2026-10-01T00:00:00Z \
  --event-end-time 2026-10-02T00:00:00Z \
  --destination "{\"Arn\":\"$BUS_ARN\"}"
```

The enhanced Custom Event Bus

Classic bus	Enhanced bus (Sep 2026)
Rules, up to 5 targets each	Subscribers, one target each
No retention: add an archive	Retention 1–365 days
JSON via `PutEvents`	Also raw bytes via `PutRawEvents`
No ordering or dedup	FIFO per event group, deduplication

AWS relaunched custom buses in September 2026; the original became Custom Event Bus - Classic, and its APIs are unchanged. The enhanced bus uses the `eventsv2` CLI and SDK namespace and bills per GB instead of per event. The two interoperate, so you can migrate one producer or consumer at a time.

```
{
  "Effect": "Allow",
  "Action": [
    "events:PutEvents", "events:PutRawEvents"
  ],
  "Resource": "arn:aws:events:*:*:event-bus/shop"
}
```

> **Gotcha:** IAM actions keep the `events:` prefix even though the CLI says `eventsv2`. A policy naming `eventsv2:PutEvents` is accepted but grants nothing, and callers get `AccessDeniedException`.

Quotas and cost

Item	Classic bus value
Request size	1 MB, all entries combined
Entries per `PutEvents` call	10
Rules per bus	300 default, adjustable
`PutEvents` rate	Per Region, e.g. 10,000/s in us-east-1
Custom events	$1.00 per million, per 64 KB chunk

Item	Classic bus value
AWS service events	Free on the default bus

Put payloads over 1 MB in S3 and send the object's URL instead. Large payloads cost more than their count suggests because each 64 KB chunk bills as one event. Check the quotas page for your Region's rates before load testing.

```
200 KB event  -> 4 chunks  -> billed as 4 events
1M x 200 KB   -> 4M events -> $4.00 ingestion
```

Further reading

- EventBridge User Guide
- Event pattern operators
- EventBridge quotas
- EventBridge Scheduler User Guide
- EventBridge pricing