# Amazon Bedrock and AgentCore

Version SDK v3 (3.1161) · verified 2026-11-02

Bedrock as an IAM-guarded model gateway — Converse, streaming, tool use, Knowledge Bases, Guardrails, AgentCore Runtime and friends, pricing, quotas.

## Contents


Mental model ..... 2

Model access and IDs ..... 2

The Converse API ..... 3

Streaming ..... 5

Tool use ..... 6

Knowledge Bases ..... 7

Guardrails ..... 8

AgentCore ..... 10

Pricing and quotas ..... 11

Further reading ..... 12

## Mental model

Bedrock is a model gateway with IAM in front: one regional API, one request shape, many vendors' models, and your AWS account's permissions, quotas, and bill around every call. You do not host models; you send messages and pay per token. AgentCore is a separate set of services for running *your own* agent code — a runtime, tool gateway, memory, and identity — not a no-code agent builder.

## Model access and IDs

| Item | Detail |
|---|---|
| IAM action | `bedrock:InvokeModel` (also covers Converse) |
| Serverless models | Enabled by default in commercial Regions |
| Anthropic models | One-time use-case form first |
| `us.` / `eu.` prefix | Geographic cross-Region profile |
| `global.` prefix | Routes worldwide, no residency limit |

The Model access page is retired: serverless models work immediately, and you restrict them with IAM policies and SCPs instead. Some models are sold through AWS Marketplace, and the first call subscribes the account, which needs `aws-marketplace` permissions. Many models are offered only through cross-Region inference profiles, so the model ID you pass often carries a prefix such as `global.`.

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

```
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "bedrock:InvokeModel",
      "bedrock:InvokeModelWithResponseStream"
    ],
    "Resource": "*"
  }]
}
```

> **Gotcha:** A `global.` profile may serve your request from any Region. Use a geographic prefix such as `eu.` when data must stay in a jurisdiction.

## The Converse API

| Field | Purpose |
|---|---|
| `modelId` | Model or inference profile ID |
| `messages` | Alternating `user` / `assistant` turns |
| `system` | System prompt blocks |
| `inferenceConfig` | `maxTokens`, `temperature`, `topP` |
| `stopReason` | `end_turn`, `tool_use`, `max_tokens`, … |

`Converse` gives every model the same message format, so switching vendors is a change of `modelId`. Each message has a `role` and a list of content blocks (`text`, `image`, `document`, `toolUse`, `toolResult`). The

response returns the assistant message, the stop reason, and token `usage`. Bedrock keeps no conversation: send the whole history each turn.

```
import {
  BedrockRuntimeClient, ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";
const client = new BedrockRuntimeClient({
  region: "us-east-1",
});
const cmd = new ConverseCommand({
  modelId: "global.anthropic.claude-sonnet-5-5",
  system: [{ text: "Answer in one sentence." }],
  messages: [{ role: "user",
    content: [{ text: "What is Bedrock?" }] }],
  inferenceConfig: { maxTokens: 300 },
});
const res = await client.send(cmd);
console.log(res.output.message.content[0].text);
```

> **Gotcha:** Check `stopReason` before trusting the text. `max_tokens` means the answer was cut off, and the default token limit varies by model.

## Streaming

| Event | Carries |
|---|---|
| `contentBlockDelta` | Text, or partial tool-input JSON |
| `contentBlockStop` | End of one block |
| `messageStop` | `stopReason` |
| `metadata` | `usage` and latency |

`ConverseStreamCommand` takes the same input and returns an async iterable `stream` of events. Tool input arrives as partial JSON fragments across deltas; join them and parse after the block stops. IAM needs `bedrock:InvokeModelWithResponseStream`.

```
import {
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";
const { stream } = await client.send(
  new ConverseStreamCommand({
    modelId, messages,
  }),
);
for await (const ev of stream) {
  const text = ev.contentBlockDelta?.delta.text;
  if (text) process.stdout.write(text);
  if (ev.messageStop) {
    console.log(ev.messageStop.stopReason);
```

```
  }
}
```

## Tool use

| Step | Who |
|---|---|
| Declare tools in `toolConfig.tools[].toolSpec` | You |
| Reply with `stopReason: "tool_use"` | Model |
| Run the tool named in each `toolUse` block | You |
| Send a `toolResult` with the same `toolUseId` | You |
| Repeat until `end_turn` | Both |

Bedrock never runs a tool you declare with `toolSpec`; it returns a request and waits for you to answer it. Each `toolSpec` has a `name`, a `description`, and a JSON Schema under `inputSchema.json`. Append the assistant message and a `user` message holding the results, then call `Converse` again. Mark failures with `status: "error"` so the model can recover.

```
const toolConfig = { tools: [{ toolSpec: {
  name: "get_order",
  description: "Look up an order by ID",
  inputSchema: { json: { type: "object",
    properties: { id: { type: "string" } } } },
} }] };
// after a response with stopReason "tool_use":
```

```
messages.push(res.output.message);
const results = res.output.message.content
  .filter((b) => b.toolUse)
  .map(({ toolUse: t }) => ({ toolResult: {
    toolUseId: t.toolUseId,
    content: [{ json: lookup(t.input) }] } }));
messages.push({ role: "user",
  content: results });
```

> **Gotcha:** Every `toolUse` in a turn needs a matching `toolResult` in the next user message. Answering only some of them — or dropping the assistant message from the history — fails validation.

## Knowledge Bases

| Piece | Purpose |
|---|---|
| Data source (for example S3) | Documents to ingest |
| Vector store | Holds the embeddings |
| `Retrieve` | Returns matching chunks |
| `RetrieveAndGenerate` | Retrieves, then answers with citations |

A Knowledge Base is managed retrieval-augmented generation: Bedrock chunks and embeds your documents into a vector store, then retrieves the relevant chunks at query time. Use `Retrieve` when you assemble the prompt yourself and `RetrieveAndGenerate` for a cited answer in one call. Both live in the separate `bedrock-agent-runtime` client.

```
import {
  BedrockAgentRuntimeClient as KbClient,
  RetrieveAndGenerateCommand as Rag,
} from "@aws-sdk/client-bedrock-agent-runtime";

const kb = new KbClient({});
const out = await kb.send(new Rag({
  input: { text: "What is our refund policy?" },
  retrieveAndGenerateConfiguration: {
    type: "KNOWLEDGE_BASE",
    knowledgeBaseConfiguration: {
      knowledgeBaseId: "KB12345678", modelArn },
  },
}));
console.log(out.output.text, out.citations);
```

## Guardrails

| Option | Effect |
|---|---|
| `guardrailConfig.guardrailIdentifier` | Guardrail ID or ARN |
| `guardrailConfig.guardrailVersion` | Version or `"DRAFT"` |
| `guardrailConfig.trace: "enabled"` | Explains interventions |
| `stopReason: "guardrail_intervened"` | Input or output blocked |
| `ApplyGuardrail` | Check text without a model call |

A guardrail is a reusable policy — denied topics, content filters, sensitive-information masking, contextual grounding — that you create once and attach to calls. Pass `guardrailConfig` on `Converse` and Bedrock checks both the prompt and the response. `ApplyGuardrail` evaluates any text, including output from models outside Bedrock.

```
const res = await client.send(
  new ConverseCommand({
    modelId, messages,
    guardrailConfig: {
      guardrailIdentifier: "gr-abc123",
      guardrailVersion: "1",
      trace: "enabled",
    },
  }));
if (res.stopReason === "guardrail_intervened") {
  console.log("blocked", res.trace?.guardrail);
}
```

> **Warning:** A guardrail filters content; it is not access control. Prompt injection that stays on-topic passes through, so keep tools and data behind IAM and least privilege.

## AgentCore

| Service | Does |
|---|---|
| Runtime | Hosts your agent code, one microVM per session |
| Gateway | Turns APIs and Lambdas into MCP tools |
| Memory | Short- and long-term memory |
| Identity | Inbound auth, outbound OAuth tokens |
| Code Interpreter, Browser | Sandboxed tools |
| Observability, Evaluations, Policy | Traces, quality scoring, tool rules |

Runtime, Gateway, Memory, Identity, Code Interpreter, Browser, and Observability have been GA since October 2025; Policy since March 2026. Runtime is framework-agnostic: each `runtimeSessionId` gets an isolated microVM that ends after 15 idle minutes or 8 hours by default. The `agentcore` CLI (`npm i -g @aws/agentcore`, 1.0) scaffolds, runs locally with `agentcore dev`, and deploys with `agentcore deploy`.

```
import {
  BedrockAgentCoreClient,
  InvokeAgentRuntimeCommand,
} from "@aws-sdk/client-bedrock-agentcore";

const ac = new BedrockAgentCoreClient({});
```

```
const res = await ac.send(
  new InvokeAgentRuntimeCommand({
    agentRuntimeArn: runtimeArn,
    runtimeSessionId: crypto.randomUUID(),
    payload: new TextEncoder().encode(
      JSON.stringify({ prompt: "Hello" })),
  }));
const body = res.response;
console.log(await body.transformToString());
```

> **Note:** Bedrock Agents (now “Agents Classic”) has been closed to new customers since 30 July 2026: `CreateAgent` and `InvokeInlineAgent` fail for accounts without prior use. AWS points new work to AgentCore, starting with its managed harness.

## Pricing and quotas

| Option | Cost model |
|---|---|
| Standard (on-demand) | Per input and output token |
| Priority | 75% premium over Standard |
| Flex | 50% discount, not time-sensitive |
| Batch | 50% off on-demand, select models |
| Reserved / Provisioned | Committed throughput, term-based |

Set the tier per request with `serviceTier`; support varies by model. Quotas are per model, per Region, in requests and tokens per minute,

and cross-Region profiles have their own. Some models burn output tokens against the quota at 5× — check the model's burndown rate before sizing. Over quota, calls fail with `ThrottlingException`; retry with exponential backoff and jitter, and request increases in Service Quotas.

```
aws service-quotas list-service-quotas \
  --service-code bedrock \
  --query "Quotas[].[QuotaName,Value]"
```

> **Note:** AWS prioritizes quota increases for accounts already consuming their current allocation. Ask before launch, with traffic to show.

## Further reading

- Converse API
- Tool use with Converse
- Guardrails with Converse
- Bedrock runtime quotas
- AgentCore sessions
- Agents Classic maintenance mode