User Guide

Agent Toolkit for AWS

Agent Toolkit for AWS: User Guide

Table of Contents

What is the Agent Toolkit for AWS?

The Agent Toolkit for AWS gives AI coding agents the tools, knowledge, and guardrails they need to build, deploy, and manage applications on AWS. It works with the coding agents that developers already use, such as Kiro, Claude Code, Cursor, and Codex. You don't need to switch tools or learn a new workflow.

AI coding agents can handle common AWS tasks like creating an Amazon S3 bucket or launching an Amazon EC2 instance. However, they often struggle with complex, multi-step workflows. They choose the wrong service for a use case, misconfigure resources, or repeatedly retry operations on newer services they were not trained on. The Agent Toolkit for AWS addresses these gaps. It gives agents secure access to AWS services, current documentation, and step-by-step guidance.

Topics

Components

The Agent Toolkit for AWS includes four components that work together:

- **AWS MCP Server** – A managed server that gives agents access to AWS through the Model Context Protocol (MCP). Kiro, Claude Code, Cursor, Codex, and other MCP-compatible agents connect to this server. Agents can search AWS documentation and get service information without authentication. To run API calls, execute Python scripts, or follow curated skills, agents authenticate with your existing AWS Identity and Access Management (IAM) credentials. A single endpoint provides all these capabilities, with CloudWatch metrics and IAM-based access controls.
- **Agent skills** – Curated packages of instructions, code scripts, and reference materials that help agents complete specific AWS tasks. Agents load skills on demand. They discover and retrieve only the skills relevant to the current task, so they do not consume unnecessary context. Skills cover service selection, step-by-step procedures, troubleshooting, and SDK best practices.

- **Plugins** – Single-install packages for Claude Code and Codex that bundle the AWS MCP Server configuration and a curated set of agent skills. After you install a plugin, your agent is ready to work with AWS. Kiro connects to the AWS MCP Server directly and does not require a plugin.
- **Rules files** – Project-level configuration files that set guardrails and preferences for how agents work in your project. Rules files tell agents how to use AWS most effectively and securely. For example, they direct agents to use the AWS MCP Server, discover available skills, or search documentation before acting.

What can I do with the Agent Toolkit for AWS?

With the Agent Toolkit for AWS, your AI coding agent can:

- **Build and deploy applications on AWS** – Your agent creates AWS infrastructure, writes application code, and deploys to production. Skills guide your agent through choosing the right services, configuring them correctly, and following deployment best practices.
- **Stay up to date on the latest AWS services** – The AI models that power coding agents are trained on data that can be months or years old. Newer AWS services and recently launched features are often missing from an agent's knowledge. The Agent Toolkit for AWS gives your agent real-time access to current AWS documentation, API references, and service capabilities.
- **Follow tested procedures for complex workflows** – Instead of improvising from general knowledge, your agent follows tested procedures. These include configuring least-privilege IAM policies, setting up data pipelines, and deploying production-ready serverless applications.
- **Troubleshoot operational issues** – Point your agent at a failing deployment, a spike in error rates, or an unexpected cost increase. Skills help your agent work with CloudWatch logs and metrics, CloudFormation stack status, and troubleshooting procedures.
- **Operate with security and visibility** – The AWS MCP Server provides CloudWatch metrics for monitoring agent activity and IAM-based access controls. You can also set enterprise guardrails, such as restricting agents to read-only operations or blocking specific actions.
- **Work with any MCP-compatible agent** – The Agent Toolkit for AWS works with Claude Code, Cursor, Codex, Kiro, Windsurf, Cline, and any other agent that supports the Model Context Protocol.

How the Agent Toolkit for AWS works

The Agent Toolkit for AWS helps your AI coding agent build on AWS in three ways:

- **Skills provide structured guidance** – Your agent uses skills installed locally through a plugin or discovers them at runtime through the AWS MCP Server. Skills contain step-by-step instructions, decision guides, and reference materials. These guide your agent through complex procedures while following AWS best practices.
- **Knowledge tools provide current information** – When your agent needs up-to-date information, it can search AWS documentation and retrieve API references. It can also check regional availability and access the latest AWS service information.
- **API tools execute authenticated actions** – The AWS MCP Server translates requests into properly formatted AWS API calls, handles authentication with your IAM credentials, and returns detailed feedback about results and errors. For complex multi-step operations, your agent can write and run Python scripts in an isolated sandbox using the `run_script` tool.

The AWS MCP Server automatically adds two global condition context keys (`aws:ViaAWSMCPService` and `aws:CalledViaAWSMCP`) to all requests. These keys let you differentiate MCP-initiated actions from direct API calls in your IAM policies. CloudTrail logs all API calls for audit visibility.

Authentication and authorization use your existing AWS IAM roles and policies, so you maintain full control over what your agent can access and do. You can authenticate using SigV4 credentials through the MCP Proxy for AWS, or use OAuth 2.1 through AWS Sign-in for a direct connection without local proxy software. See the section called “OAuth authentication” for OAuth setup.

> **Note**
>
> We recommend scoping down IAM roles to the minimum permissions that your agent needs to perform its task.

Pricing

You can use the Agent Toolkit for AWS at no additional charge. You pay only for the AWS resources your agent provisions or interacts with, at standard AWS pricing. For more information about AWS pricing, see AWS Pricing. If you are new to AWS, you can get started with many services for free. For more information, see AWS Free Tier.

Getting started

The fastest way to get started with the Agent Toolkit for AWS is to install the plugin for your AI coding agent. The plugin bundles the AWS MCP Server configuration and a curated set of agent skills in a single install.

Sign up for an AWS account

To get started with AWS, you need an AWS account. For information about creating an AWS account, see Getting started with an AWS account in the *AWS Account Management Reference Guide*.

Prerequisites

- uv installed on your system (required for the MCP proxy).
- (Optional) An AWS account with IAM credentials set up on your local machine. Credentials are required for tools that execute AWS API calls and run scripts, but not for searching documentation or discovering skills. If you do not have credentials configured, see the section called "Setting up the AWS MCP Server" for detailed instructions.

Step 1: Install

Claude Code

In Claude Code:

```
/plugin install aws-core@claude-plugins-official
/reload-plugins
```

Codex

In your terminal:

```
codex plugin marketplace add aws/agent-toolkit-for-aws
```

Then launch Codex and run `/plugins` to browse and install the **aws-core** plugin.

Kiro

Add the following to your MCP configuration file (for example, `~/.kiro/settings/mcp.json`):

```
{
  "mcpServers": {
    "aws-mcp": {
      "command": "uvx",
      "timeout": 100000,
      "transport": "stdio",
      "args": [
        "mcp-proxy-for-aws-cli@latest",
        "https://aws-mcp.us-east-1.api.aws/mcp",
        "--metadata", "AWS_REGION=us-west-2"
      ]
    }
  }
}
```

Then install skills:

```
npx skills add aws/agent-toolkit-for-aws/skills
```

Other agents

If your agent supports MCP, you can configure the AWS MCP Server directly. See the section called "Setting up the AWS MCP Server" for instructions.

Then install skills:

```
npx skills add aws/agent-toolkit-for-aws/skills
```

AWS CLI

If you have the AWS CLI installed (version `2.35.0` or later), you can run an interactive setup wizard that detects your installed AI coding agents, installs default AWS skills, and configures the AWS MCP Server connection in a single command:

```
aws configure agent-toolkit
```

The wizard works with multiple agents, including Kiro, Cursor, and Claude Code. For more information, including how to install and manage individual skills from the command line, see the section called "AWS CLI".

Step 2: Verify your connection

After you install the plugin, verify that the AWS MCP Server is connected:

1. Start a new conversation with your agent.
2. Ask: *"What AWS Regions are available?"*

If the agent returns a list of AWS Regions, the connection is working. If you see an authentication error, see the section called "Troubleshooting authentication errors".

Step 3: Try it out

Ask your agent to perform an AWS task:

- "What AWS services should I use to build a serverless API?"
- "Create an Amazon S3 bucket with versioning enabled and a lifecycle policy that transitions objects to Glacier after 90 days."
- "Help me troubleshoot why my CloudFormation deployment failed."

The agent discovers and uses relevant skills automatically. You do not need to know which skills are available — the agent finds them based on your request.

Configure the recommended rules file

We recommend that you add the Agent Toolkit for AWS rules file to your project. A rules file provides persistent instructions that tell your agent how to work with AWS in every session. Without it, your agent might apply AWS best practices less consistently.

Copy the latest recommended rules file from the Agent Toolkit for AWS repository on GitHub into the rules file for your agent (for example, `CLAUDE.md` for Claude Code or `AGENTS.md` for Codex). For the file location for each agent and how to customize it, see the section called "Rules files".

Additional plugins

After you install aws-core, you can install additional plugins for specialized workflows:

- **aws-agents** — Skills for building AI agents on AWS with API Gateway and AgentCore.
- **aws-data-analytics** — Skills for data lake, analytics, and ETL workflows.
- **aws-agents-for-devsecops** — Skills for incident investigation, code security scanning, and penetration testing.

Install additional plugins using the same method as aws-core.

Agent Toolkit for AWS components

The Agent Toolkit for AWS includes four components that work together to help AI coding agents build, deploy, and manage applications on AWS.

- **the section called "AWS MCP Server"** — A managed server that gives agents access to AWS through the Model Context Protocol.
- **the section called "Skills"** — Curated packages of instructions and reference materials that help agents complete specific AWS tasks.
- **the section called "Plugins"** — Single-install packages that bundle the AWS MCP Server configuration and agent skills.
- **the section called "Rules files"** — Project-level configuration files that tell agents how to use AWS most effectively.

AWS MCP Server

The AWS MCP Server is a managed server that gives agents access to AWS through the Model Context Protocol (MCP). Agents can search AWS documentation and retrieve service information without authentication. To execute AWS API calls, run Python scripts in a sandboxed environment, or follow curated skills, agents authenticate through your existing IAM credentials.

All capabilities are available through a single endpoint with CloudWatch metrics and IAM-based access controls. CloudTrail logs all API calls for audit visibility.

Setting up the AWS MCP Server

If you already have an AWS account, skip to the section called "Set up the AWS MCP Server". If you are new to AWS, sign up for an AWS account and then continue with the setup process.

Set up the AWS MCP Server

AWS MCP Server supports two authentication methods. The following describes both options, followed by a decision guide to help you choose.

- **Option A: OAuth (simple)** — Connect to AWS MCP Server using OAuth. If you are a human user, you authenticate in your browser. If you are running an automated agent, it authenticates by requesting a token. Works for web clients (such as Claude.ai and ChatGPT.com) and most

IDE, terminal, or desktop-based clients. Use this option if you are new to AWS and use a single account, or if you interact through web clients.

- **Option B: SigV4 (advanced)** — Authenticate using the AWS CLI, then use the MCP Proxy for AWS to sign requests with SigV4 credentials. Use this option if you use terminal or IDE-based coding agents (such as Claude Code, Kiro, and Codex), or if you need to switch between AWS accounts frequently.

Choosing an authentication method

Use the following questions to determine which authentication method fits your use case:

Authentication method decision guide

Question	Use
Do you want to get started without installing `uvx`, installing the AWS CLI, or configuring local credentials?	OAuth
Does your client only support remote MCP servers (no local process)?	OAuth
Does your agent need access across multiple AWS accounts in the same session?	SigV4
Do you need read-only mode (hide write-capable tools from the agent entirely)?	SigV4
Does your organization restrict the `signin:AuthorizeOAuth2Access` and `signin:CreateOAuth2Token` permissions needed for browser-based OAuth login?	SigV4
Do you need to set a default AWS Region for your agentic session (without specifying it in every query)?	SigV4
Does your client not support the OAuth flow but can run a local MCP proxy?	SigV4

Topics

Step 1: (If applicable) Remove conflicting MCP servers

If you are currently using the AWS API MCP Server or AWS Knowledge MCP Server, we recommend switching to the AWS MCP Server. The AWS MCP Server is a managed remote MCP server that reduces setup and maintenance effort and offers enhanced security controls through IAM condition keys.

To switch, remove the older servers from your MCP client configuration to avoid tool conflicts that can confuse AI agents and reduce performance.

To remove existing AWS MCP servers:

1. Open your MCP client configuration file (for example, `~/.kiro/settings/mcp.json` for Kiro).
2. Remove any entries for these servers:
   - `aws-api-mcp-server`
   - `aws-knowledge-mcp-server`
3. Save the configuration file.
4. Restart your MCP client to apply the changes.

Step 2: Configure authentication and connect

AWS MCP Server supports the following AWS Regions:

- US East (N. Virginia) – `us-east-1`: `https://aws-mcp.us-east-1.api.aws/mcp`
- Europe (Frankfurt) – `eu-central-1`: `https://aws-mcp.eu-central-1.api.aws/mcp`

Option A: OAuth (simple)

With OAuth, you can connect directly to AWS MCP Server without installing or running local proxy software. Your MCP client handles the OAuth flow automatically.

Prerequisites

1. Grant OAuth permissions to your IAM role or user by attaching the managed policy:

```
aws iam attach-role-policy \
  --role-name MyRole \
  --policy-arn arn:aws:iam::aws:policy/AWSMCPSignInOAuthAccessPolicy
```

Configure your MCP client

The following clients fully support OAuth with the default endpoint URL:

Claude Code CLI

```
claude mcp add aws-mcp https://aws-mcp.us-east-1.api.aws/mcp --transport http
```

Claude Code for Web

Add the URL `https://aws-mcp.us-east-1.api.aws/mcp` in your web client MCP settings.

Kiro CLI (2.11 or later)

```
kiro-cli mcp add --name aws-mcp --url https://aws-mcp.us-east-1.api.aws/mcp
```

Devin Desktop

Add as a remote MCP server with URL: `https://aws-mcp.us-east-1.api.aws/mcp`

Devin CLI

```
devin mcp add aws-mcp https://aws-mcp.us-east-1.api.aws/mcp
```

The following clients support OAuth when you append `?oauth=initialize` to the endpoint URL:

Claude Desktop

Add as a remote MCP server with URL: `https://aws-mcp.us-east-1.api.aws/mcp?oauth=initialize`

Cursor

Add as a remote MCP server with URL: `https://aws-mcp.us-east-1.api.aws/mcp?oauth=initialize`

Kiro IDE

Add as a remote MCP server with URL: `https://aws-mcp.us-east-1.api.aws/mcp?oauth=initialize`

Gemini CLI

```
gemini mcp add aws-mcp https://aws-mcp.us-east-1.api.aws/mcp?oauth=initialize --transport http
```

Codex CLI and Codex Desktop

```
codex mcp add aws-mcp --url https://aws-mcp.us-east-1.api.aws/mcp?oauth=initialize
```

When you first invoke a tool, your MCP client opens a browser window to AWS Sign-in. Sign in with your existing credentials and authorize access. After you grant consent, access tokens remain valid for 1 hour. AWS Sign-in automatically refreshes them for up to 12 hours.

If your client is not listed above, use the default endpoint URL (`https://aws-mcp.us-east-1.api.aws/mcp`). Without `?oauth=initialize`, the client relies on MCP OAuth discovery to start the authorization flow automatically. If tool calls fail due to credential errors, append `?oauth=initialize` to the URL to instruct the server to explicitly trigger the OAuth flow.

For detailed OAuth setup, governance options, and token management, see the section called "OAuth authentication".

> **Note**
>
> Multi-profile switching for cross-account workflows is not supported with OAuth. If you need to work with multiple AWS accounts in a single session, use the SigV4 authentication option instead.

Option B: SigV4 (advanced)

Use the MCP Proxy for AWS to authenticate requests with SigV4. With this option, you can switch between multiple profiles for cross-account workflows.

Prerequisites

1. Install the AWS CLI (`2.32.0` or later) by following Installing the AWS CLI.
2. Sign in and configure credentials:

```
aws login
```

   The command auto-rotates credentials every 15 minutes for sessions up to 12 hours.
3. Verify your credentials:

```
aws sts get-caller-identity
```

4. Install uv (if not already installed):

```
# macOS and Linux
curl -LsSf https://astral.sh/uv/install.sh | sh

# Windows
powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 | iex"
```

For other credential methods (SSO, IAM access keys, cross-account roles), see Sign in with the AWS CLI.

Configure your MCP client

The endpoint Region determines which MCP server you connect to, while the `AWS_REGION` metadata parameter sets the default Region for AWS operations. Without `AWS_REGION`, operations default to `us-east-1`.

Kiro CLI or Kiro IDE

```
{
  "mcpServers": {
    "aws-mcp": {
```

```
      "command": "uvx",
      "timeout": 100000,
      "transport": "stdio",
      "args": [
        "mcp-proxy-for-aws-cli@latest",
        "https://aws-mcp.us-east-1.api.aws/mcp",
        "--metadata", "AWS_REGION=us-west-2"
      ]
    }
  }
}
```

Cursor IDE, Claude Desktop, or Devin Desktop

```
{
  "mcpServers": {
    "aws-mcp": {
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws-cli@latest",
        "https://aws-mcp.us-east-1.api.aws/mcp",
        "--metadata", "AWS_REGION=us-west-2"
      ]
    }
  }
}
```

Claude Code CLI

```
claude mcp add-json aws-mcp '{"type":"stdio","command":"uvx","args":["mcp-
proxy-for-aws-cli@latest","https://aws-mcp.us-east-1.api.aws/mcp","--
metadata","AWS_REGION=us-west-2"],"env":{}}'
```

Codex CLI

```
codex mcp add aws-mcp uvx mcp-proxy-for-aws-cli@latest https://aws-mcp.us-
east-1.api.aws/mcp --metadata AWS_REGION=us-west-2
```

Devin CLI

```
devin mcp add aws-mcp -- uvx mcp-proxy-for-aws-cli@latest https://aws-mcp.us-
east-1.api.aws/mcp --metadata AWS_REGION=us-west-2
```

> **Tip**
>
> To work with multiple AWS accounts in a single session, configure additional named profiles. See the section called "Multi-profile support" for setup instructions. Multi-profile switching is only available with SigV4 authentication.

Step 3: Test your connection

1. Start your MCP client (Kiro CLI, Cursor, Claude Desktop, or a similar client).
2. Wait for the MCP server to initialize (this might take a few minutes on the first connection).
3. Test the connection by asking your AI assistant:

   Example: What AWS Regions are available?
4. Verify that tools are loaded by running (in Kiro CLI):

   ```
   /tools
   ```

   Or to see installed MCP servers:

   ```
   /mcp
   ```

You should see tools like `aws___search_documentation` and `aws___retrieve_skill` listed. For more information about the tools, see Understanding the MCP Server tools.

Troubleshooting authentication errors

Authentication errors can prevent the MCP server from initializing. When this happens, your AI agent cannot use AWS MCP tools. If your AI agent is not using AWS MCP tools, an expired or missing credential is the most likely cause.

Use the following table to identify and resolve common authentication errors.

Common authentication errors

Error	Cause	Resolution
`ExpiredTokenException` :	Your temporary AWS credentials have expired.	Refresh your credentials based on your authentication method:

Error	Cause	Resolution
Your AWS session token has expired.	This is the most common authentication error. Short-lived session tokens (default 1 hour) typically expire during development.	• `aws login` users: Run `aws login` again to start a new session. Consider switching to this method if you experience frequent expiry, because it auto-rotates credentials every 15 minutes. • SSO users: Run `aws sso login --profile your-profile-name` to refresh your SSO session. • Assume-role users: Refresh the source profile credentials. The SDK then automatically re-assumes the role. After refreshing, restart your MCP client to re-initialize the server.
`UnrecognizedClientException`: The security token included in the request is not recognized.	Your credentials are invalid. This can happen when credentials have been revoked, are from a different AWS partition, are malformed, or belong to a deleted IAM user or role.	Verify your credentials are valid: 1. Run `aws sts get-caller-identity` to check if your credentials are recognized. 2. If the command fails, reconfigure your credentials using `aws login` or `aws configure sso`. 3. Ensure you are using credentials for the correct AWS partition (for example, `aws` vs. `aws-cn`).

Error	Cause	Resolution
`InvalidSignatureException` : The request signature we calculated does not match the signature you provided.	The SigV4 signature does not match. Common causes include credentials scoped to the wrong service or Region, clock skew on your machine, or a request body modified after signing.	Try the following steps: 1. Verify your system clock is accurate. Run `date` and compare with an authoritative time source. SigV4 requires your clock to be within 5 minutes of AWS servers. 2. Ensure your credentials are configured for the correct AWS Region and service. 3. Reconfigure your credentials and restart your MCP client.
400 error page after OAuth sign-in	Your IAM principal does not have the required OAuth permissions (`signin:AuthorizeOAuth2Access` and `signin:CreateOAuth2Token`).	Attach the `AWSMCPSignInOAuthAccessPolicy` managed policy to your IAM role or user: `aws iam attach-role-policy \ --role-name MyRole \ --policy-arn arn:aws:iam::aws:policy/AWSMCPSignInOAuthAccessPolicy` For more information, see AWS managed policies for AWS MCP Server.
No AWS credentials found.	AWS credentials are not configured on your machine, or the credential provider chain cannot locate them.	Configure your credentials by following the section called “Step 2: Configure authentication and connect”. We recommend using `aws login` for the simplest setup with automatic credential renewal.

> **Note**
>
> To learn more about how AWS IAM authorizes AWS MCP Server requests, including how to use IAM condition context keys to restrict agent actions, see Understanding IAM for managed AWS MCP servers on the AWS Security Blog.

Multi-profile support

When using the MCP Proxy for AWS with the AWS MCP Server, you can configure multiple AWS CLI profiles to switch between accounts or roles on a per-call basis. The proxy adds an `aws_profile` parameter to the server's auth-requiring tools, letting the agent route each request through a different set of credentials without restarting.

> **Note**
>
> This feature is specific to the AWS MCP Server. Profile switching is not available when proxying to other MCP servers.

> **Important**
>
> Multi-profile switching requires SigV4 authentication with the MCP Proxy for AWS. OAuth authentication does not support this feature. If you use OAuth, each session is bound to a single IAM role and switching accounts requires re-authentication.

How it works

1. You configure the proxy with multiple profiles at startup (via the `--profile` flag or `AWS_MCP_PROXY_PROFILES` environment variable).
2. The proxy adds an `aws_profile` parameter into the tool schema for `run_script`, `get_presigned_url`, and `get_tasks`.
3. When the agent makes a tool call:
   - Without `aws_profile`: the proxy signs with the default (first) profile.
   - With `aws_profile="dev"`: the proxy routes through a dedicated connection signed with the `dev` profile's credentials.

- With an invalid profile: the proxy rejects the call with an error listing allowed profiles.

4. The `aws_profile` parameter is stripped before forwarding to the backend — the AWS MCP Server never sees it.

Configuration

Configure multiple profiles using either the CLI flag or the environment variable.

CLI flag

The first profile is the default. Additional profiles are switchable:

```
mcp-proxy-for-aws-cli https://aws-mcp.us-east-1.api.aws/mcp --profile prod-readonly dev
 staging
```

Environment variable

Same behavior, useful for plugin integration where CLI args cannot be modified:

```
AWS_MCP_PROXY_PROFILES="prod-readonly dev staging"
```

> **Note**
>
> `AWS_MCP_PROXY_PROFILES` takes precedence over `--profile` and `AWS_PROFILE` when set.

Example MCP config

```
{
  "mcpServers": {
    "aws-mcp": {
      "command": "uvx",
      "args": ["mcp-proxy-for-aws-cli@latest", "https://aws-mcp.us-east-1.api.aws/
mcp"],
      "env": {
        "AWS_MCP_PROXY_PROFILES": "prod-readonly dev staging"
      }
```

```
    }
  }
}
```

Prerequisites

- AWS CLI profiles configured in `~/.aws/config` and `~/.aws/credentials` for each profile you want to use.
- `mcp-proxy-for-aws` version 1.6.0 or later.
- Valid IAM permissions for each profile. Each profile should have the minimum permissions required for the operations the agent will perform.

Security considerations

- **Explicit allowlist:** Only profiles declared at startup are available. The agent cannot discover or use other profiles in `~/.aws/config`.
- **Stateless routing:** Each call carries its own identity. No shared session state means parallel requests cannot interfere with each other.
- **Least privilege:** Configure profiles with the minimum permissions needed. Consider using a read-only profile as the default and requiring explicit selection of write-capable profiles.
- **Client-side gating:** For additional control (for example, requiring manual approval before using a production profile), configure client-side hooks or permission rules in your MCP client.

Example use cases

- **Cross-account cost comparison:** "Compare Lambda invocation costs between my dev and prod accounts."
- **Security audit:** "Check all S3 buckets across my three accounts for public access."
- **Troubleshooting:** "List failed ECS tasks in staging, then check the same service config in prod."
- **Resource inventory:** "Count EC2 instances across all my accounts."

Understanding the MCP Server tools

AWS MCP Server provides the following tools to help you complete AWS tasks through natural language interactions.

AWS Knowledge Tools

- `aws___retrieve_skill` - Retrieve domain-specific expertise for a particular AWS domain. Skills provide workflows, context, best practices, decision frameworks, and step-by-step procedures. When called with a skill name, returns the full skill content including reference materials. Use `aws___search_documentation` to discover available skills.
- `aws___search_documentation` - Search across all AWS documentation, including API references, best practices, service guides, and skills (formerly Agent SOPs). Use the topic filter to search skills exclusively, or see skills alongside general knowledge search results. Find relevant information from multiple AWS knowledge sources.
- `aws___read_documentation` - Retrieve and convert AWS documentation pages to markdown format for easy consumption by AI assistants.
- `aws___list_regions` - Retrieve a list of all AWS regions, including their identifiers and names.
- `aws___get_regional_availability` - Check AWS regional availability information for services, features, SDK APIs, and CloudFormation resources.

AWS API Tools

- `aws___run_script` - Execute Python code in a sandboxed environment with AWS API access. Use for tasks that involve listing resources and checking their properties, parallel API calls, multi-step workflows, cross-service checks, and retry logic.
- `aws___get_presigned_url` - Generate pre-signed Amazon S3 URLs for uploading or downloading files. Use this tool for direct Amazon S3 uploads and downloads, or when an AWS CLI command requires a local file path.
- `aws___get_tasks` - Poll the status of long-running tasks started by `aws___run_script`. Use when a previous tool call returns a task ID with a working status.

These tools work together to provide comprehensive AWS task completion: skills guide the workflow, knowledge tools provide current information and best practices, and API tools execute the actual AWS operations with proper authentication and authorization.

When multiple profiles are configured (via `--profile` or `AWS_MCP_PROXY_PROFILES`), the MCP Proxy for AWS injects an optional `aws_profile` parameter into the schema of `aws___run_script`, `aws___get_presigned_url`, and `aws___get_tasks`. This parameter lets the agent route individual requests through different AWS credential profiles. The parameter

is stripped by the proxy before forwarding to the server. See the section called “Multi-profile support” for configuration details.

Capabilities

In addition to the general-purpose AWS knowledge and API tools, AWS MCP Server provides capabilities. A capability groups a set of helper functions that work together to handle a specific kind of AWS task. This document describes the capabilities that AWS MCP Server provides and what they do.

> **Note**
>
> Your agent discovers the relevant capability for a task and invokes its helper functions automatically through AWS MCP Server with no manual setup required. As a best practice, describe the outcome you want (for example, "find out why this AWS Lambda function is failing"), and let your agent select the appropriate tools.

AWS MCP Server provides a serverless capability that helps you troubleshoot your running Lambda functions and their connected resources.

Serverless capability

The serverless capability helps coding and operational agents debug AWS serverless applications. It answers questions such as "Why is this Lambda function failing?", "Did anything change before the outage?", and "Is the issue in code, config, or an AWS dependency?" It provides a set of serverless-aware helper functions that return structured data your agent can use to diagnose and resolve issues.

The capability is scoped to the caller's own account and is read-only. It inspects the Lambda function and its connected resources, and can diagnose the following connected resource types:

- Amazon SQS
- Amazon SNS
- Amazon DynamoDB (including per-index Global Secondary Indexes)
- Amazon API Gateway (REST and HTTP)
- Amazon EventBridge

- AWS Step Functions

The serverless capability provides the following helper functions.

- `diagnose` – Returns a combined health check and rule-based diagnosis for a Lambda function and its connected resources in a single call. This includes current health status, metrics compared against a 7-day baseline to surface anomalies, detection of Lambda triggers and destinations, and root cause analysis correlated across the connected resources to identify the likely source of failure (for example, "DynamoDB throttling is causing Lambda invocation errors"). This gives your agent a starting point for resolution without requiring separate log, trace, and configuration inspection for every investigation.
- `search_logs` – Retrieves recent log evidence for a Lambda function from Amazon CloudWatch Logs. The response groups log messages into common exception types and representative error lines, producing a structured summary for agents to consume. This surfaces recurring exceptions and failure patterns without overwhelming the agent's context window.
- `get_live_config` – Returns the deployed configuration of a Lambda function and its connected resources, using consistent structure and field names across resource types. This covers runtime, memory, timeout, concurrency, event source mappings, and connected-resource settings. Use it to catch configuration drift (for example, a timeout that is too low, exhausted reserved concurrency, an SQS visibility timeout mismatch, or a missing dead-letter queue) or to confirm that a deployment took effect.
- `get_recent_changes` – Returns a timeline of recent deployments and configuration changes for a Lambda function and its connected resources, sourced from AWS CloudTrail, Lambda APIs, and AWS CloudFormation. Each entry shows who made the change, which settings changed, and their previous and new values. If the function was deployed as part of an AWS CloudFormation stack, the timeline also includes that stack's deployment events. Use it to answer "what changed before the Lambda function broke?"
- `get_trace_summary` – Summarizes recent AWS X-Ray traces to show where a Lambda function's latency and errors originate. X-Ray records the duration of each step in an invocation, including calls to downstream services. This helper function samples traces from a recent time window (the last 30 minutes by default) and identifies the downstream calls with the highest latency and error counts (for example, "DynamoDB Query is consuming 24 seconds of a 30-second Lambda invocation"). For this helper function to return results, X-Ray tracing must be enabled on the function.

Skills

Agent skills are curated packages of instructions, code scripts, and reference materials that help AI coding agents complete specific AWS tasks. Skills bridge the gap between what AI models know from training data and what they need to work effectively with AWS. This is especially important for newer services, complex multi-service workflows, and tasks where best practices matter.

Topics

What is a skill?

A skill is a directory containing a `SKILL.md` file and optional supporting files. The `SKILL.md` file includes a brief description and instructions that tell your agent how to complete a task. These instructions specify which steps to follow, which AWS APIs to call, which mistakes to avoid, and how to verify the result. Skills can also include reference files with deeper guidance on specific subtasks, code scripts for deterministic operations, and slash commands that let you invoke the skill directly.

Skills are lightweight and modular. Each skill focuses on a single task or domain, and your agent loads only the skills it needs for the current task. A skill typically consumes a few thousand tokens when loaded. This is far less than the equivalent documentation, because a skill contains only the information your agent needs to act, not background context it already has.

How agents discover and use skills

There are four ways agents get access to skills:

Bundled with a plugin

> Each plugin includes a curated set of skills that are available to your agent immediately after installation. Your agent can use these skills without any network calls or additional setup.

Installed locally

You can download individual skills from the Agent Toolkit for AWS repository on GitHub and add them to your agent's skills directory.

Installed with the AWS CLI

You can install skills with the AWS CLI through the interactive setup wizard (`aws configure agent-toolkit`) or with the individual `aws agent-toolkit` commands. The CLI detects supported agents on your system and installs skills into each agent's configuration directory. For more information, see the section called “AWS CLI”.

Discovered at runtime through the AWS MCP Server

Agents can search for and retrieve skills on demand through the AWS MCP Server, without any local installation. Your agent uses the `search_documentation` tool to find relevant skills and the `retrieve_skill` tool to load them into context.

Regardless of how a skill was installed, your agent uses it the same way:

1. Your agent reads the skill's description to determine if it is relevant to the current task.
2. If relevant, your agent loads the full instructions from `SKILL.md`.
3. Your agent follows the skill's procedures, loading reference files only as needed.
4. After the task is complete, your agent releases the skill content from context.

This progressive disclosure means that many available skills do not slow your agent down or consume unnecessary context. Your agent loads only the skills relevant to the current task.

Types of skills

The Agent Toolkit for AWS includes several types of skills:

Service decision guides

These skills help your agent choose the right AWS service for a use case. For example, a database decision guide helps your agent recommend Amazon DynamoDB, Amazon Aurora, or Amazon DSQL based on the workload requirements.

Step-by-step procedures

These skills provide tested workflows for common tasks like creating Amazon S3 Tables, setting up AWS Glue ETL pipelines, configuring IAM policies, and deploying serverless applications.

Troubleshooting guides

These skills provide diagnostic procedures for common errors, with steps to identify the cause and resolve the issue. For example, a CloudFormation deployment troubleshooting skill covers the top failure patterns and how to fix them.

SDK usage guides

These skills provide language-specific best practices for the AWS SDKs. They cover common mistakes that models consistently get wrong, like Amazon DynamoDB marshalling in JavaScript or pagination patterns in Python.

Available skills

Each plugin includes a curated set of skills covering the most common workflows for that domain. The full set of skills, including domain-specific skills for individual AWS services, is available on GitHub and discoverable at runtime through the AWS MCP Server.

To see what skills are available from within your agent, ask: *"What AWS skills do you have available?"* or *"Search for AWS skills related to databases."*

You can also browse and install skills from the command line:

```
npx skills add aws/agent-toolkit-for-aws/skills
```

Using skills with and without the AWS MCP Server

Skills work best with the AWS MCP Server, which provides authenticated API access, sandboxed script execution, and enterprise controls like CloudWatch metrics and IAM context keys. For production workflows, use the AWS MCP Server.

Skills also work without the AWS MCP Server. When your agent does not have access to the AWS MCP Server, it can run the same AWS operations using the AWS CLI directly. Each skill includes instructions that work with both approaches.

AWS CLI

The AWS CLI provides commands to set up AI coding agents with the Agent Toolkit for AWS. You can manage installed AWS skills from the command line. The CLI works with any AI coding agent that the toolkit detects on your system, including Kiro, Cursor, and Claude Code.

Use the CLI when you want to install skills and configure the AWS MCP Server in a single step.

The CLI provides two command groups:

- `aws configure agent-toolkit` - An interactive setup wizard that detects installed agents, installs default AWS skills, and configures the AWS MCP Server.
- `aws agent-toolkit` - A set of commands to install, update, remove, list, and search for individual AWS skills.

Topics

Prerequisites

- AWS CLI version `2.35.0` or later. To install or update the AWS CLI, see Installing the AWS CLI in the *AWS Command Line Interface User Guide*.
- One or more supported AI coding agents installed on your system. The CLI detects supported agents by the presence of their configuration directories (for example, `~/.kiro` for Kiro). Skills are installed globally in those configuration directories, not per project.

Setting up with the AWS CLI

The `aws configure agent-toolkit` command runs an interactive setup wizard. It detects installed AI coding agents, installs default AWS skills, and configures the AWS MCP Server connection.

```
aws configure agent-toolkit
```

After the wizard completes, your selected agents can use the AWS MCP Server and the installed skills. Restart your agent client to load the new configuration.

Managing skills with the AWS CLI

The `aws agent-toolkit` command group provides operations to install, update, remove, list, and search for individual skills. By default, install, update, and remove operations apply to all detected agents. Use the `--agent` option to target a specific tool.

The value for `--agent` is the stable agent identifier (for example, `kiro`). Run any command with `--help` to see valid values.

Topics

Install a skill

The `add-skill` command downloads and installs an AWS skill to detected AI coding agents. By default, it installs the latest version globally to all detected agents. Use `--agent` to target a specific tool, or `--skill-version` to pin a specific version.

Example 1: Install a skill

The following `add-skill` example downloads and installs the `aws-serverless` skill to all detected AI coding agents.

```
aws agent-toolkit add-skill \
    --skill-name aws-serverless
```

Example 2: Install a skill to a specific agent

The following `add-skill` example installs the `aws-cdk` skill only to Kiro.

```
aws agent-toolkit add-skill \
    --skill-name aws-cdk \
    --agent kiro
```

Example 3: Install a specific version of a skill

The following `add-skill` example installs version `v1` of the `aws-serverless` skill.

```
aws agent-toolkit add-skill \
    --skill-name aws-serverless \
    --skill-version v1
```

Update an installed skill

The `update-skill` command updates an installed AWS skill to the latest version. It compares the local version against the catalog and downloads a newer version if one is available. By default, the command updates the skill for all detected agents. Use `--agent` to update only for a specific tool.

Example 1: Update an installed skill

The following `update-skill` example updates the `aws-serverless` skill to the latest version across all agents.

```
aws agent-toolkit update-skill \
    --skill-name aws-serverless
```

Example 2: Update a skill for a specific agent

The following `update-skill` example updates the `aws-serverless` skill only for Kiro.

```
aws agent-toolkit update-skill \
    --skill-name aws-serverless \
    --agent kiro
```

Remove an installed skill

The `remove-skill` command removes a previously installed AWS skill from detected agents. By default, it removes the skill from all detected agents. Use `--agent` to remove the skill from a specific tool only.

Example 1: Remove an installed skill

The following `remove-skill` example removes the `aws-serverless` skill from all detected agents.

```
aws agent-toolkit remove-skill \
    --skill-name aws-serverless
```

Example 2: Remove a skill from a specific agent

The following `remove-skill` example removes the `aws-cdk` skill only from Kiro.

```
aws agent-toolkit remove-skill \
    --skill-name aws-cdk \
    --agent kiro
```

List installed skills

The `list-installed-skills` command lists AWS skills that you previously installed with the `aws agent-toolkit` commands. It shows the skill name, agent, and file path for each installation. By default, it lists skills from all detected agents. Use `--agent` to filter results to a specific tool.

Example 1: List installed skills

The following `list-installed-skills` example lists all AWS skills installed on detected agents.

```
aws agent-toolkit list-installed-skills
```

Example 2: List installed skills for a specific agent

The following `list-installed-skills` example lists skills installed only in Kiro.

```
aws agent-toolkit list-installed-skills \
    --agent kiro
```

List available skills

The `list-available-skills` command lists skills available in the remote catalog.

Example 1: List all available skills

The following `list-available-skills` example lists all skills available in the remote catalog.

```
aws agent-toolkit list-available-skills
```

Example 2: List available skills filtered by category

The following `list-available-skills` example lists only skills in the `aws-core` category.

```
aws agent-toolkit list-available-skills \
    --category-filter aws-core
```

Search for skills

The `search-skills` command searches the remote catalog for skills matching a query.

Example: Search for available skills

The following `search-skills` example searches for skills related to serverless development.

```
aws agent-toolkit search-skills \
    --search-query serverless
```

Get skill metadata

The `get-skill-metadata` command retrieves metadata for a skill. This includes the version, description, categories, and file list.

Example 1: Get metadata for a skill

The following `get-skill-metadata` example retrieves metadata for the `aws-serverless` skill.

```
aws agent-toolkit get-skill-metadata \
    --skill-name aws-serverless
```

Example 2: Get metadata for a specific skill version

The following `get-skill-metadata` example retrieves metadata for version `v1` of the `aws-serverless` skill.

```
aws agent-toolkit get-skill-metadata \
    --skill-name aws-serverless \
    --skill-version v1
```

Get a file from a skill

The `get-skill-file` command retrieves the contents of a single file from a skill. Use `aws agent-toolkit get-skill-metadata` to discover available file names for each skill. By default, the command retrieves the latest version. Use `--skill-version` to fetch a specific version.

Example 1: Fetch a file from a skill

The following `get-skill-file` example retrieves the `SKILL.md` file from the `aws-serverless` skill.

```
aws agent-toolkit get-skill-file \
    --skill-name aws-serverless \
    --file-path SKILL.md
```

Example 2: Fetch a specific version of a skill file

The following `get-skill-file` example retrieves a reference file from a specific version of the `aws-serverless` skill.

```
aws agent-toolkit get-skill-file \
```

```
    --skill-name aws-serverless \
    --file-path references/architecture.md \
    --skill-version v1
```

Plugins

Plugins for the Agent Toolkit for AWS are single-install packages that bundle the AWS MCP Server configuration and a curated set of agent skills. Instead of configuring MCP endpoints and installing skills individually, you install one plugin. Your agent is then ready to work with AWS.

Plugins work with Claude Code and Codex. They are open source and published at Agent Toolkit for AWS repository on the GitHub website. Kiro connects to the AWS MCP Server directly and does not require a plugin. For instructions on setting up Kiro, see the section called "Setting up the AWS MCP Server".

Topics

Available plugins

- **aws-core** - The primary plugin for the Agent Toolkit for AWS. Includes the AWS MCP Server configuration and default skills for common AWS workflows. Covered domains include service selection, infrastructure as code (CDK and CloudFormation), serverless, containers, databases, storage, observability, billing, SDK usage, and deployment. We recommend this plugin for all AWS developers.
- **aws-agents** - Provides skills for building AI agents on AWS using Amazon API Gateway, Amazon Bedrock AgentCore, and related services.
- **aws-data-analytics** - Provides skills for data lake, analytics, and ETL workflows using Amazon Simple Storage Service Tables, AWS Glue, Amazon Athena, and related services.
- **aws-agents-for-devsecops** - Provides skills for investigating incidents, reviewing code for release readiness, scanning code for vulnerabilities, and running penetration tests using AWS DevOps Agent and AWS Security Agent.

Installing plugins

Claude Code

In Claude Code:

```
/plugin install aws-core@claude-plugins-official
/reload-plugins
```

Codex

In your terminal:

```
codex plugin marketplace add aws/agent-toolkit-for-aws
```

Then launch Codex and run `/plugins` to browse and install the **aws-core** plugin.

Agents that do not support plugins

If your agent does not support plugins, you can configure the AWS MCP Server directly. See the section called “Setting up the AWS MCP Server” for instructions.

What a plugin contains

Each plugin contains the following:

- **Plugin manifest** (`.claude-plugin/plugin.json` or `.codex-plugin/plugin.json`) - Contains the plugin metadata, including name, description, and version.
- **Skills directory** (`skills/`) - Contains the agent skills that the plugin provides.
- **MCP configuration** (`.mcp.json`) - Defines the AWS MCP Server endpoint configuration.

Updating plugins

Your agent client updates plugins automatically when it refreshes its marketplace data. To update manually, run the update command in your agent client.

Rules files

Most AI coding agents support project-level configuration files, often called rules files. These files provide persistent instructions that your agent follows in every session. For example, you might have a rules file that tells your agent to always write TypeScript in strict mode, or to use a specific testing framework.

The Agent Toolkit for AWS includes a recommended rules file that tells your agent how to work with AWS. For example, the rules file instructs your agent to use the AWS MCP Server for API calls, to search for available skills before starting a task, and to prefer infrastructure-as-code over direct CLI commands.

Topics

Recommended AWS rules file

The Agent Toolkit for AWS includes a recommended rules file that covers using the AWS MCP Server, discovering skills, verifying against documentation, and following infrastructure-as-code best practices. You can find the latest version of this file in the Agent Toolkit for AWS repository on GitHub. Copy the content into the appropriate file for your agent.

Where to put the rules file

The file name and location depend on your agent:

Rules file locations by agent

Agent	Project rules	Location
Claude Code	`CLAUDE.md`	Project root
Codex	`AGENTS.md`	Project root
Cursor	`.cursor/rules/*.mdc`	`.cursor/rules/` directory
Kiro	`.kiro/steering/*.md`	`.kiro/steering/` directory

For Claude Code and Codex, add the recommended rules file content to your existing `CLAUDE.md` or `AGENTS.md` file, or create a new one in your project root. For Cursor, create a new `.mdc` file in the `.cursor/rules/` directory. For example, name it `.cursor/rules/aws.mdc`. Your agent still supports the legacy `.cursorrules` file, but it is deprecated.

Customization

The recommended rules file is a starting point. Customize it for your project. For example, you can specify which AWS Region to use, which VPC to deploy into, or which naming conventions to follow for resources.

The following example shows project-specific rules you might add:

```
# Project-specific AWS rules

- Default region is eu-west-1. Do not create resources in other regions
  unless explicitly asked.
- All infrastructure must be deployed into VPC vpc-0abc1234567890def in the
  shared-services account.
- Resource naming convention: {team}-{service}-{environment}-{purpose}
  (e.g., payments-lambda-prod-processor).
- Always tag resources with Team=payments, CostCenter=CC-4521, and
  Environment=(dev|staging|prod).
- Use Python 3.12 for all Lambda functions. Do not use Node.js unless
  the project already uses it.
- Never provision resources larger than t3.medium without confirmation.
```

Add these rules alongside the recommended AWS rules in your agent's rules file. Your agent follows both sets of instructions in every session.

Security in AWS MCP Server

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to AWS MCP Server, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

The Agent Toolkit for AWS has different security characteristics depending on how you use it:

- **AWS MCP Server** — A stateless proxy that executes AWS API calls on your behalf using your IAM credentials. Authentication and authorization for the operations it performs are governed by your IAM identity.
- **AWS CLI (`aws agent-toolkit`)** — Commands that discover and install AWS-vended agent skills over HTTPS. These commands are unauthenticated and do not send credentials. No customer data is stored.

The topic-specific pages below cover security for both usage paths.

This documentation helps you understand how to apply the shared responsibility model when using AWS MCP Server. The following topics show you how to configure AWS MCP Server to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your AWS MCP Server resources.

Topics

- Data protection in AWS MCP Server

- Identity and access management for AWS MCP Server
- OAuth 2.1 authentication for AWS MCP Server
- Compliance validation for AWS MCP Server
- Resilience in AWS MCP Server
- Data protection for the AWS CLI
- IAM for the AWS CLI

Data protection in AWS MCP Server

The AWS shared responsibility model applies to data protection in Agent Toolkit for AWS. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see Data Privacy FAQ. For information about data protection in Europe, see the General Data Protection Regulation (GDPR) Center.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with Agent Toolkit for AWS or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

> **Note**
>
> The AWS MCP Server doesn't support FIPS endpoints.

Encryption

AWS MCP Server is a stateless proxy that executes AWS API calls on your behalf. The service does not use any persistent storage services such as Amazon S3, DynamoDB, Amazon EBS, or Amazon SQS to store customer content. When file processing is required (for example, deploying an application package), customer content exists only transiently on ephemeral compute storage during active processing.

Encryption at rest

When AWS MCP Server temporarily processes customer files, the data resides only on ephemeral compute storage (Lambda `/tmp` storage and AgentCore Runtime ephemeral storage). This ephemeral storage is:

- Automatically encrypted at rest by the compute platform using AWS managed keys.
- Automatically destroyed when ephemeral storage is reclaimed (maximum ephemeral storage retention of 8 hours). Most operations complete in seconds.
- Not accessible through any external API or operator tooling.
- Fully isolated per customer through hardware-backed tenant isolation. For more information, see Security overview for Lambda in the *Lambda Developer Guide*.

AWS MCP Server does not currently offer customer managed KMS keys for encrypting ephemeral compute storage. While Lambda supports customer managed keys for `/tmp` storage, the service

uses AWS managed encryption across all compute platforms for consistency and operational simplicity. However, your source data in Amazon S3 remains protected by whatever encryption you have configured on your buckets, including customer managed keys. If your Amazon S3 objects are encrypted with a customer managed key, AWS MCP Server requires `kms:Decrypt` permission (provided through your credentials) to access those objects. Revoking this permission prevents the service from accessing your content.

Encryption in transit

All communication between your MCP client and AWS MCP Server is encrypted using TLS. All downstream AWS API calls made by the service on your behalf also use TLS encryption.

Key management

AWS MCP Server uses AWS managed keys to encrypt ephemeral compute storage. You do not need to manage or configure these keys.

For your source data stored in AWS services such as Amazon S3, you retain full control over your encryption keys. AWS MCP Server accesses your data using credentials derived from your own IAM session. These credentials are scoped to your authenticated session and respect all AWS KMS key policies you have configured.

File operations

AWS MCP Server runs in a cloud environment that cannot access your local file system directly. When an operation requires a file (for example, deploying a package or uploading an object to Amazon S3), the service uses pre-signed Amazon S3 URLs to stage files through your own Amazon S3 bucket.

The file operation workflow is:

1. The service generates a pre-signed Amazon S3 URL pointing to your Amazon S3 bucket.
2. You upload your file to your own bucket using the pre-signed URL.
3. The service references the staged file when calling the target AWS service.

When the target AWS service accepts Amazon S3 locations natively, the service passes the Amazon S3 reference directly and your file is never downloaded to the server. When the target service requires file content directly, the service temporarily downloads the file to ephemeral compute storage, executes the operation, and immediately deletes the temporary copy.

Billing considerations

Because file staging uses your own Amazon S3 bucket, standard Amazon S3 charges apply to your account:

- Amazon S3 PUT and GET request charges for each file upload and download.
- Amazon S3 storage charges for staged files that remain in your bucket.

The AWS MCP Server service itself does not incur additional storage charges for file operations. For current pricing, see Amazon S3 pricing.

Limits

File staging has the following limits:

- Pre-signed URLs expire after a maximum of 15 minutes (900 seconds).
- The maximum size for a single staged file that the service downloads for processing is 125 MB.
- The maximum size for a staged zip archive (used for directory uploads such as `deploy push --source` or `gamelift upload-build --build-root`) is 1 GB.
- The maximum total size across all staged files on the runtime is 4 GB. This budget is shared across all concurrent operations.

Internetwork traffic privacy

AWS MCP Server communicates with AWS services over the AWS network using TLS-encrypted connections. Your requests to the AWS MCP Server endpoint are encrypted in transit using TLS 1.2 or later.

Identity and access management for AWS MCP Server

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Agent Toolkit for AWS resources. IAM is an AWS service that you can use with no additional charge.

Topics

Audience

How you use AWS Identity and Access Management (IAM) differs based on your role:

- **Service user** - request permissions from your administrator if you cannot access features (see Troubleshooting AWS MCP Server identity and access)
- **Service administrator** - determine user access and submit permission requests (see How AWS MCP Server works with IAM)
- **IAM administrator** - write policies to manage access (see Identity-based policy examples for AWS MCP Server)

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be authenticated as the AWS account root user, an IAM user, or by assuming an IAM role.

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook credentials. For more information about signing in, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you

don't use the root user for everyday tasks. For tasks that require root user credentials, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users to use federation with an identity provider to access AWS services using temporary credentials.

A *federated identity* is a user from your enterprise directory, web identity provider, or Directory Service that accesses AWS services using credentials from an identity source. Federated identities assume roles that provide temporary credentials.

For centralized access management, we recommend AWS IAM Identity Center. For more information, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more information, see Require human users to use federation with an identity provider to access AWS using temporary credentials in the *IAM User Guide*.

An *IAM group* specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity with specific permissions that provides temporary credentials. You can assume a role by switching from a user to an IAM role (console) or by calling an AWS CLI or AWS API operation. For more information, see Methods to assume a role in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For

more information about JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Using policies, administrators specify who has access to what by defining which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. An IAM administrator creates IAM policies and adds them to roles, which users can then assume. IAM policies define permissions regardless of the method used to perform the operation.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you attach to an identity (user, group, or role). These policies control what actions identities can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be *inline policies* (embedded directly into a single identity) or *managed policies* (standalone policies attached to multiple identities). To learn how to choose between managed and inline policies, see Choose between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples include IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. You must specify a principal in a resource-based policy.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Other policy types

AWS supports additional policy types that can set the maximum permissions granted by more common policy types:

- **Permissions boundaries** – Set the maximum permissions that an identity-based policy can grant to an IAM entity. For more information, see Permissions boundaries for IAM entities in the *IAM User Guide*.

- **Service control policies (SCPs)** – Specify the maximum permissions for an organization or organizational unit in AWS Organizations. For more information, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – Set the maximum available permissions for resources in your accounts. For more information, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Advanced policies passed as a parameter when creating a temporary session for a role or federated user. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How AWS MCP Server works with IAM

AWS MCP Server uses a simplified authorization model that works like the AWS Command Line Interface (AWS CLI) and AWS SDKs. The server does not define its own IAM actions, resources, or service-specific condition keys. Instead, it authenticates your request using SigV4, adds standardized condition context keys, and forwards the request to the downstream AWS service. The downstream service performs the authorization check using your existing IAM policies. This means your AI agents work with your existing AWS credentials and service-level permissions, and you do not need to configure separate MCP-specific IAM actions.

Authorization flow

When an AI agent calls the AWS MCP Server, the following authorization flow occurs:

1. Your agent's request is authenticated with your AWS credentials using SigV4. The MCP Proxy for AWS handles this signing automatically between your host application and the server.
2. AWS MCP Server authenticates the request and adds the MCP condition context keys (`aws:ViaAWSMCPService` and `aws:CalledViaAWSMCP`).
3. AWS MCP Server forwards the request to the target AWS service.
4. The target AWS service authorizes the request using your existing IAM policies, which can reference the MCP condition context keys for fine-grained control.

MCP condition context keys

AWS MCP Server automatically adds the following global condition context keys to all requests it forwards to downstream AWS services. You can use these keys in IAM policies and service control policies (SCPs) to differentiate between requests made through an AWS managed MCP server and direct API calls.

`aws:ViaAWSMCPService`

A Boolean key set to `true` for any request that passes through an AWS managed MCP server. Use this key to allow or deny all actions initiated through any AWS managed MCP server.

Type: Boolean

`aws:CalledViaAWSMCP`

A single-valued string key containing the service principal of the specific AWS managed MCP server that initiated the request. Use this key to apply controls for a specific MCP server.

Type: String

Example values:

- `aws-mcp.amazonaws.com` – AWS MCP Server
- `eks-mcp.amazonaws.com` – Amazon EKS MCP Server
- `ecs-mcp.amazonaws.com` – Amazon ECS MCP Server

For more information about global condition context keys, see IAM condition context keys in the *IAM User Guide*.

OAuth condition keys for AWS Sign-in

When you use OAuth authentication with AWS MCP Server through AWS Sign-in, the following service-specific condition keys are available in the `signin:` namespace. Use these keys in IAM policies and SCPs to govern how OAuth is used within your organization.

`signin:OAuthClientId`

The ARN of the OAuth client making the request. Use this key to allow only approved OAuth clients to access AWS MCP Server.

Type: String (ARN)

Applies to: `signin:AuthorizeOAuth2Access`, `signin:CreateOAuth2Token`

`signin:OAuthRedirectUri`

The redirect URI used in the authorization request. Use this key to restrict token delivery to trusted redirect URIs such as localhost or corporate domains.

Type: String

Applies to: `signin:AuthorizeOAuth2Access`

`signin:OAuthGrantType`

The OAuth grant type being used. Use this key to control which OAuth authorization flows are permitted.

Type: String

Example values: `authorization_code`, `refresh_token`, `client_credentials`

Applies to: `signin:CreateOAuth2Token`

`signin:OAuthClientAuthentication`

The client authentication method used. Use this key to require specific authentication methods.

Type: String

Example values: none, `client_secret_basic`, `client_secret_post`

Applies to: `signin:CreateOAuth2Token`

OAuth access tokens carry the `aws:SignInSessionArn` global condition key. This key contains the unique ARN of the AWS Sign-in session.

For more information about OAuth condition keys, see OAuth condition keys reference in the *AWS Sign-in User Guide*. For OAuth authentication setup and governance examples, see the section called “OAuth authentication”.

Identity-based policies

Supports identity-based policies: Yes

Because AWS MCP Server forwards requests to downstream AWS services using your credentials, the IAM policies attached to your IAM user or role determine what actions the MCP server can

perform on your behalf. No additional IAM configuration is required to use AWS MCP Server beyond the permissions you already grant for direct API access.

You can use the MCP condition context keys in your existing policies to apply different permissions when actions are initiated through an MCP server. For examples, see Identity-based policy examples for AWS MCP Server.

Using temporary credentials with AWS MCP Server

Supports temporary credentials: Yes

AWS MCP Server works with temporary credentials obtained through AWS STS. When you authenticate with AWS MCP Server, you can use temporary credentials from IAM roles, federated identities, or assumed roles. The server forwards these credentials to downstream AWS services, which honor the same session policies and permission boundaries as direct API calls.

Deprecated MCP-specific IAM actions

During the preview period, AWS MCP Server required the following service-specific IAM actions:

- `aws-mcp:InvokeMcp`
- `aws-mcp:CallReadOnlyTool`
- `aws-mcp:CallReadWriteTool`

These actions are no longer required and have no effect. If you previously configured IAM permissions using these actions, we recommend that you remove them from your policies. If you used these actions in Deny statements to block access to AWS MCP Server, you must update your policies to use the `aws:ViaAWSMCPService` or `aws:CalledViaAWSMCP` condition context keys instead.

Identity-based policy examples for AWS MCP Server

The following examples show how to use IAM policies with the MCP condition context keys to control access through AWS MCP Server. Each example shows the policy statement to include within your IAM policy document.

Topics

- Restrict actions to a specific MCP server

Deny all actions through any AWS managed MCP server

The following SCP or IAM policy denies all actions when the request originates from any AWS managed MCP server. Use this to completely block MCP server access across an organization or for specific principals.

```
{
    "Sid": "DenyAllActionsViaMCP",
    "Effect": "Deny",
    "Action": "*",
    "Resource": "*",
    "Condition": {
        "Bool": {
            "aws:ViaAWSMCPService": "true"
        }
    }
}
```

Deny destructive actions through AWS MCP Server

The following policy allows read operations but denies destructive actions when the request comes through AWS MCP Server. This lets AI agents inspect resources without being able to delete them.

```
[
    {
        "Sid": "AllowS3ReadOperations",
        "Effect": "Allow",
        "Action": [
            "s3:GetObject",
            "s3:ListBucket"
        ],
        "Resource": "*"
    },
    {
        "Sid": "DenyDeleteWhenAccessedViaMCP",
        "Effect": "Deny",
        "Action": [
            "s3:DeleteObject",
            "s3:DeleteBucket"
```

```
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "aws:CalledViaAWSMCP": "aws-mcp.amazonaws.com"
            }
        }
    }
]
```

Restrict actions to a specific MCP server

The following policy denies all actions when the request comes specifically through AWS MCP Server, while allowing requests from other AWS managed MCP servers or direct API calls.

```
{
    "Sid": "DenyActionsViaAWSMCPServer",
    "Effect": "Deny",
    "Action": "*",
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "aws:CalledViaAWSMCP": "aws-mcp.amazonaws.com"
        }
    }
}
```

Troubleshooting AWS MCP Server identity and access

Use the following information to help you diagnose and fix common issues when working with AWS MCP Server and IAM.

Topics

- I get an access denied error when using AWS MCP Server
- My Deny policy using aws-mcp actions no longer blocks access

I get an access denied error when using AWS MCP Server

If you receive an `AccessDenied` error when AWS MCP Server calls a downstream AWS service on your behalf, check the following:

- Verify that your IAM role or user has the required permissions for the target AWS service action. AWS MCP Server uses your credentials, so you need the same permissions as you would for a direct API call.
- Check whether any SCPs or permission boundaries include Deny statements that use `aws:ViaAWSMCPService` or `aws:CalledViaAWSMCP` conditions that block MCP server access.
- If you previously used `aws-mcp:InvokeMcp` in Allow statements, note that these actions no longer have any effect. Your permissions for the downstream service are what matter.

My Deny policy using aws-mcp actions no longer blocks access

If you previously used Deny statements with `aws-mcp:InvokeMcp`, `aws-mcp:CallReadOnlyTool`, or `aws-mcp:CallReadWriteTool` to block access to AWS MCP Server, these actions no longer have any effect. Update your policies to use the condition context keys instead:

```
{
    "Effect": "Deny",
    "Action": "*",
    "Resource": "*",
    "Condition": {
        "Bool": {
            "aws:ViaAWSMCPService": "true"
        }
    }
}
```

OAuth 2.1 authentication for AWS MCP Server

AWS MCP Server supports OAuth 2.1 authorization through AWS Sign-in. OAuth-compatible MCP clients — including Claude Code, Cursor, Gemini CLI, Kiro, and GitHub Copilot — can connect directly to AWS MCP Server. No local proxy software is required. For an overview of how OAuth works with AWS Sign-in, see OAuth with AWS Sign-in overview and AWS MCP Server in the *AWS Sign-in User Guide*.

With OAuth, your MCP client authenticates using standard OAuth authorization flows while continuing to rely on existing IAM identities, permissions, and organizational controls. AWS Sign-in acts as the OAuth authorization server, issuing scoped access tokens that AWS MCP Server accepts as bearer tokens.

Prerequisites

Before using OAuth with AWS MCP Server, ensure the following:

- **IAM permissions** — The IAM principal must have `signin:AuthorizeOAuth2Access` and `signin:CreateOAuth2Token` actions allowed. You can use the AWS managed policy `AWSMCPSignInOAuthAccessPolicy`, or create a custom policy.
- **Supported MCP client** — An OAuth-compatible MCP client such as Claude Code, Cursor, Cline, Gemini CLI, Kiro, or GitHub Copilot.

Attach the managed policy

```
aws iam attach-role-policy \
  --role-name MyRole \
  --policy-arn arn:aws:iam::aws:policy/AWSMCPSignInOAuthAccessPolicy
```

Or use a custom IAM policy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "signin:AuthorizeOAuth2Access",
        "signin:CreateOAuth2Token"
      ],
      "Resource": "arn:aws:signin:*:*:service-principal/aws-mcp.amazonaws.com"
    }
  ]
}
```

Configure your MCP client for OAuth

Add the AWS MCP Server endpoint directly to your MCP client. Unlike the SigV4 proxy method, OAuth-compatible clients connect to the server URL without requiring `mcp-proxy-for-aws` or `uvx`.

Kiro CLI (2.11 or later)

```
kiro-cli mcp add --name aws-mcp --url https://aws-mcp.us-east-1.api.aws/mcp
```

Claude Code

```
claude mcp add aws-mcp https://aws-mcp.us-east-1.api.aws/mcp --transport http
```

Claude Desktop

Add as a remote MCP server with URL: `https://aws-mcp.us-east-1.api.aws/mcp?oauth=initialize`

Cursor

Add as a remote MCP server with URL: `https://aws-mcp.us-east-1.api.aws/mcp?oauth=initialize`

Codex

```
codex mcp add aws-mcp --url https://aws-mcp.us-east-1.api.aws/mcp?oauth=initialize
```

Gemini CLI

```
gemini mcp add aws-mcp https://aws-mcp.us-east-1.api.aws/mcp?oauth=initialize --
transport http
```

For other clients, see your MCP client documentation for instructions on adding a remote MCP server. Use the endpoint URL `https://aws-mcp.us-east-1.api.aws/mcp` (or the endpoint for your preferred Region).

Legacy MCP clients

Some MCP clients might not fully support the latest MCP specification for OAuth discovery. If your client does not automatically initiate the OAuth flow, append the `?oauth=initialize` query parameter to the endpoint URL:

```
https://aws-mcp.us-east-1.api.aws/mcp?oauth=initialize
```

This instructs the server to explicitly trigger the OAuth authorization flow for clients that do not handle it natively.

Authenticate with OAuth

When you first invoke an AWS MCP Server tool, your MCP client opens a browser to AWS Sign-in. Sign in with your existing AWS credentials, review the consent page showing the application requesting access, and authorize access.

After you grant consent, AWS Sign-in issues short-lived OAuth tokens. Your MCP client uses these tokens to interact with AWS MCP Server on your behalf. Access tokens remain valid for 1 hour. AWS Sign-in automatically refreshes them using refresh tokens (valid for up to 12 hours), so you do not need to sign in again during your session.

If you already have an active AWS Sign-in session, AWS Sign-in can reuse that session to complete the authorization flow without requiring you to sign in again.

OAuth authorization flows

AWS Sign-in supports two authorization flows for accessing AWS MCP Server:

Interactive user access (Authorization Code flow)

If you use an MCP client on your local machine, AWS uses the OAuth authorization code flow with Proof Key for Code Exchange (PKCE). This flow supports:

- Individual developers using IAM users or root users
- Enterprise users signing in through IAM Identity Center
- Organizations federating through third-party identity providers such as Okta or Ping

The flow works as follows:

1. The MCP client discovers the authorization server from the AWS MCP Server endpoint and redirects you to AWS Sign-in.
2. You authenticate with your AWS identity (IAM user, Identity Center, or federated identity).
3. You review and approve the consent page showing the permissions requested.

4. AWS Sign-in returns an authorization code to your MCP client.
5. The MCP client exchanges the authorization code for an access token (1-hour validity) and a refresh token.

Programmatic and agentic access (Client Credentials flow)

AI agents and automated applications can access AWS MCP Server without interactive browser sign-in by using the OAuth client credentials flow. The application authenticates using its existing AWS credentials (SigV4) to obtain a short-lived OAuth access token.

This flow is designed for:

- Headless terminals and developer desktops
- AWS-managed compute environments that already have credentials
- AI agents running automated workflows

Obtain an access token

Run the following AWS CLI command to request a token:

```
aws signin create-oauth2-token-with-iam \
    --grant-type client_credentials \
    --resource aws-mcp.amazonaws.com \
    --region us-east-1
```

The response includes an access token:

```
{
    "accessToken": "<token>",
    "tokenType": "Bearer",
    "expiresIn": 3597
}
```

Include the value of `accessToken` as an `Authorization: Bearer <token>` header in your MCP client configuration. Check your client's documentation for how to include this header.

AWS Sign-in does not issue a refresh token for the client credentials flow. Access tokens remain valid for 1 hour. For more information, see AWS MCP Server in the *AWS Sign-in User Guide*.

Access control for OAuth

OAuth authorization does not grant additional AWS permissions to the MCP client. The MCP client can only act within the permissions already granted to you. IAM policies, SCPs, RCPs, permission boundaries, and data perimeter controls still determine what actions are allowed.

OAuth access uses short-lived tokens. The same IAM permissions and organizational controls that apply to other forms of AWS access also govern OAuth access.

> **Important**
>
> Multi-profile switching for cross-account workflows is not supported with OAuth authentication. If you need to work with multiple AWS accounts in a single session, use SigV4 authentication with the MCP Proxy for AWS. See the section called “Multi-profile support” for setup instructions.

Governing OAuth access

AWS Sign-in provides condition keys that you can use to control how OAuth is used within your organization. For a complete reference, see OAuth condition keys reference in the *AWS Sign-in User Guide*.

`signin:OAuthClientId`

Allow only approved OAuth clients by restricting to specific client ARNs.

`signin:OAuthRedirectUri`

Restrict token delivery to trusted redirect URIs, such as localhost or corporate domains.

`signin:OAuthGrantType`

Control which OAuth authorization flows are permitted, such as allowing the authorization code flow while blocking the client credentials flow. Values: `authorization_code`, `refresh_token`, `client_credentials`.

`signin:OAuthClientAuthentication`

Require specific client authentication methods. Values: none, `client_secret_basic`, `client_secret_post`.

For example, the following policy restricts OAuth access to localhost redirects only:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "signin:AuthorizeOAuth2Access",
      "Resource": "arn:aws:signin:*:*:service-principal/aws-mcp.amazonaws.com",
      "Condition": {
        "StringLike": {
          "signin:OAuthRedirectUri": [
            "http://localhost:*/*",
            "http://127.0.0.1:*/*"
          ]
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": "signin:CreateOAuth2Token",
      "Resource": "arn:aws:signin:*:*:service-principal/aws-mcp.amazonaws.com"
    }
  ]
}
```

Token management

Token lifecycle

- **Access tokens** — Valid for 1 hour. Bound to the MCP client, MCP server, and user session. Contain encrypted IAM credentials (not exposed to the MCP client).
- **Refresh tokens** — Valid up to 12 hours. Single-use: each refresh returns a new access token and new refresh token. Re-use of a refresh token invalidates all tokens for that session.

Token revocation

AWS Sign-in automatically revokes refresh tokens in the following situations:

- An IAM policy attached to the principal changes
- The user's password is changed (IAM users, root)

- The Sign-in session is explicitly revoked

AWS Sign-in also provides token introspection (`signin:Introspect0Auth2Token`) and token revocation (`signin:Revoke0Auth2Token`) APIs. You can use these APIs to validate the status of individual OAuth tokens and revoke specific refresh tokens without affecting other active sessions.

For example, if a refresh token is accidentally exposed in a source code repository, administrators can revoke only the affected refresh token without requiring users to reauthorize unrelated applications.

Monitoring OAuth activity

AWS CloudTrail records OAuth-related activities, including authorization requests, token issuance, token revocation, and token introspection events. CloudTrail logs capture details such as the OAuth client, target AWS MCP Server, redirect URI, authorization flow, and associated sign-in session.

When you make API calls using OAuth access tokens, AWS includes the `aws:SignInSessionArn` context key. Use this context key to correlate API activity with the originating OAuth sign-in session.

The following example shows a CloudTrail log entry for an `Authorize0Auth2Access` event:

```
{
    "eventVersion": "1.11",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAEXAMPLE:testuser",
        "arn": "arn:aws:sts::111111111111:assumed-role/Admin/testuser",
        "accountId": "111111111111"
    },
    "eventTime": "2026-07-08T05:09:00Z",
    "eventSource": "signin.amazonaws.com",
    "eventName": "AuthorizeOAuth2Access",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "203.0.113.1",
    "requestParameters": {
        "resource": "https://aws-mcp.us-west-2.api.aws/mcp",
        "redirect_uri": "http://127.0.0.1:60432/oauth/callback",
        "code_challenge_method": "S256",
        "client_id": "arn:aws:signin:us-west-2::external-client/dcr/609544da-example"
    },
```

```
    "additionalEventData": {
        "success": "true"
    },
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111111111111"
}
```

The following example shows a CloudTrail log entry for a `CreateOAuth2Token` event:

```
{
    "eventVersion": "1.11",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAEXAMPLE:testuser",
        "arn": "arn:aws:sts::111111111111:assumed-role/Admin/testuser",
        "accountId": "111111111111"
    },
    "eventTime": "2026-07-08T05:10:04Z",
    "eventSource": "signin.amazonaws.com",
    "eventName": "CreateOAuth2Token",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "203.0.113.2",
    "requestParameters": {
        "resource": "https://aws-mcp.us-west-2.api.aws/mcp",
        "client_id": "arn:aws:signin:us-west-2::external-client/dcr/609544da-example"
    },
    "additionalEventData": {
        "signInSessionArn": "arn:aws:signin:us-west-2:111111111111:session/example-
uuid",
        "success": "true"
    },
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111111111111"
}
```

Compliance validation for AWS MCP Server

Our new AWS sign-up experience is not designed for regulated workloads. If you're using our new AWS sign-up experience, but you want to use AWS for regulated workloads, you can sign up for AWS (advanced) or activate advanced features for your AWS environment.

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. For more information about your compliance responsibility when using AWS services, see AWS Security Documentation.

Resilience in AWS MCP Server

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

In addition to the AWS global infrastructure, Agent Toolkit for AWS offers several features to help support your data resiliency and backup needs.

Data protection for the AWS CLI

This topic describes data protection when you use the AWS CLI to discover and install AWS-vended agent skills (the `aws configure agent-toolkit` and `aws agent-toolkit` commands). These commands communicate with an unauthenticated, read-only endpoint over HTTPS. For data protection information about the AWS MCP Server (the authenticated component that executes AWS API calls on your behalf), see the section called “Data protection”.

No customer data

The AWS CLI does not send customer data when fetching or searching for skills. The endpoint has no concept of customer identity and persists no per-customer state.

What the AWS CLI sends

When you run `aws configure agent-toolkit` or an `aws agent-toolkit` command, the AWS CLI sends only the following data:

- Skill identifiers (for example, via `--skill-name` or via `--skill-version`) or in case of the `aws agent-toolkit search-skills` command, a search query that you provide with `--search-query` on the command line.
- Standard HTTP request metadata, such as the `User-Agent` header and your client's source IP address.

The AWS CLI does not send your AWS credentials, account information, or IAM principal when fetching skills. For more information, see the section called “IAM for the AWS CLI”.

Trust model for skills

Skills fetched by the AWS CLI are AWS-vended content. Treat them the same as AWS-published guidance: they describe operations that an AI coding agent can perform on your behalf using *your* IAM credentials. The skill content itself does not carry AWS permissions — the agent uses your existing credentials to execute any operation it derives from a skill, so IAM remains the authoritative authorization control. To constrain what an agent can do, scope down the IAM role you use with the AWS MCP Server to the minimum permissions required for the task. For more information, see the section called “Identity and access management”.

Search query privacy

When you use commands such as `aws agent-toolkit search-skills`, the natural-language search query you provide (for example, "deploy a Lambda with environment variables") is sent to AWS over TLS and may appear in service operational logs.

> **Important**
>
> Do not include confidential or sensitive information in search queries. As with tags and other free-form text fields you submit to AWS services, treat search queries as potentially observable by AWS for operational purposes.

IAM for the AWS CLI

The `aws configure agent-toolkit` and `aws agent-toolkit` commands do not use IAM. These commands fetch AWS-vended skills from a public, read-only catalog over HTTPS. The AWS CLI does not sign requests or send credentials when using these commands.

You do not need to grant any IAM permissions to discover, install, update, remove, or search for skills with the AWS CLI. No IAM policies, roles, or identity configuration is required.

For information about IAM permissions required by the AWS MCP Server (the authenticated component that executes AWS API calls on your behalf), see the section called “Identity and access management”.

Monitoring AWS MCP Server

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS MCP Server and your other AWS solutions. AWS provides the following monitoring tools to watch AWS MCP Server, report when something is wrong, and take automatic actions when appropriate:

- *Amazon CloudWatch* monitors your AWS resources and the applications you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. AWS MCP Server automatically publishes metrics to CloudWatch at no additional cost. For more information, see AWS MCP Server CloudWatch metrics.
- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the AWS CloudTrail User Guide.

AWS MCP Server CloudWatch metrics

AWS MCP Server automatically publishes metrics to Amazon CloudWatch at no additional cost. You can use these metrics to monitor usage patterns, track success rates, identify errors, and set up alarms for your AWS MCP Server operations.

Metrics namespace

All AWS MCP Server metrics are published under the `AWS-MCP` namespace in CloudWatch. Metrics are organized by tool name, allowing you to monitor which MCP tools you use most frequently (such as `aws___run_script` or `aws___list_regions`) and track their success rates.

Available metrics

The following metrics are available for AWS MCP Server. All metrics use standard resolution (1-minute granularity).

Metric	Description	Unit
`Invocation`	The number of times a tool was called, regardless of the response status.	Count

Metric	Description	Unit
`Success`	The number of successful requests that returned a 200 response.	Count
`UserError`	The number of requests that failed with a 4XX client error (excluding throttles).	Count
`SystemError`	The number of requests that failed with a 5XX server error.	Count
`Throttle`	The number of requests that were throttled with a 429 response.	Count

Metric dimensions

Metrics include the following dimensions to help you filter and analyze your data:

Tool Name

The name of the specific MCP tool that was invoked. For example, `aws___run_script`, `aws___list_regions`, or `aws___retrieve_skill`. For a complete list of available tools, see Understanding the MCP Server tools.

Using metrics

You can use AWS MCP Server metrics to:

Monitor usage patterns

Track which tools you use most frequently to understand your interaction patterns with AWS services.

Identify errors

Monitor `UserError` and `SystemError` metrics to quickly detect and troubleshoot issues such as permission problems or service disruptions.

Track success rates

Calculate success rates by comparing `Success` counts to `Invocation` counts to ensure your operations are completing successfully.

Set up alarms

Create CloudWatch alarms to notify you when error rates exceed thresholds or when usage patterns change unexpectedly.

Optimize costs

Monitor invocation counts to identify inefficient usage patterns that might be increasing your AWS costs.

For more information about working with CloudWatch metrics, see Using Amazon CloudWatch metrics in the *CloudWatch User Guide*.

Example use cases

The following examples show how you can use AWS MCP Server metrics:

Calculate success rate for API calls

Filter metrics by `Tool Name = aws___run_script` and compare `Success` to `Invocation` to calculate your API call success rate. Set up an alarm to notify you if the success rate drops below 95%.

Detect permission issues

Monitor `UserError` metrics for specific tools. A spike in user errors often indicates IAM permission issues or incorrect API parameters.

Track tool usage trends

Compare `Invocation` counts across different tools over time to understand which AWS services and operations you interact with most frequently.

Monitor system health

Set up alarms for `SystemError` metrics to be notified of service disruptions or infrastructure issues that might affect your operations.

Usage metrics

AWS MCP Server publishes usage metrics to the `AWS/Usage` namespace in CloudWatch. These metrics help you monitor your resource consumption relative to your account quotas. You can use these metrics with the Service Quotas console to measure utilization and create alarms as you approach a quota.

All usage metrics include the following dimensions: `Type`, `Resource`, `Service`, and `Class`. The `Service` dimension is always `AWS MCP` and the `Class` dimension is always `None`.

Metric name	Type	Resource	Description	Unit
`CallCount`	API	Request	The number of requests made to the AWS MCP Server in this account in the current Region.	Count
`ResourceCount`	Resource	ConcurrentConnection	The number of concurrent connections per AWS account in the current Region.	Count
`ResourceCount`	Resource	AccountSessionCount	The number of concurrent active sessions per AWS account in the current Region.	Count
`ResourceCount`	Resource	UserSessionCount	The number of concurrent active sessions per user in this AWS account in the current Region.	Count

Note

Requests that are throttled before reaching the AWS MCP Server are not reflected in the `CallCount` metric. As a result, the metric may not reliably report values above the throttling limit.

For more information about monitoring your usage and setting up alarms, see AWS usage metrics in the *CloudWatch User Guide* and Service Quotas and Amazon CloudWatch in the *Service Quotas User Guide*.

Logging AWS MCP Server API calls using AWS CloudTrail

AWS MCP Server is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in AWS MCP Server. CloudTrail captures all API calls for AWS MCP Server as events. The calls captured include calls from the AWS MCP Server console and code calls to the AWS MCP Server API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for AWS MCP Server. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to AWS MCP Server, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

AWS MCP Server information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in AWS MCP Server, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing events with CloudTrail Event history.

For an ongoing record of events in your AWS account, including events for AWS MCP Server, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for creating a trail
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple regions and Receiving CloudTrail log files from multiple accounts

All AWS MCP Server actions for authenticated tools are logged by CloudTrail and are documented in the Agent Toolkit for AWS API Reference. For example, calls to the `CallReadWriteTool` action generate entries in the CloudTrail log files.

When you use OAuth authentication, CloudTrail also records the following AWS Sign-in events:

- `AuthorizeOAuth2Access` — Recorded when a user authorizes an OAuth flow for an MCP client.
- `CreateOAuth2Token` — Recorded when AWS Sign-in issues or refreshes OAuth tokens.

These events include the OAuth client ID, target AWS MCP Server resource, redirect URI, and associated sign-in session ARN. API calls made using OAuth access tokens include the `aws:SignInSessionArn` context. Use this context to correlate API activity with the originating OAuth sign-in session. For examples of CloudTrail entries for OAuth events, see Monitoring OAuth activity.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity element.

Understanding AWS MCP Server log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

> **Important**
>
> Tool names in CloudTrail logs may not match exactly the tools shown in your MCP client. For example:

- MCP client shows: `aws___retrieve_skill`
- CloudTrail logs show: `retrieve_skill`

This occurs because CloudTrail logs the tool name without the namespace prefix used by MCP clients.

The following example shows a CloudTrail log entry that demonstrates the `CallReadWriteTool` action.

```
{
  "eventVersion": "1.08",
  "eventCategory": "Data",
  "eventType": "AwsMcpEvent",
  "userIdentity": {
      ...
  },
  "eventTime": "...",
  "eventSource": "aws-mcp.amazonaws.com",
  "eventName": "CallReadWriteTool",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "...",
  "delegatedViaAWS": "...",
  "requestParameters": {
    "method": "tools/call",
    "params": {
        // Exact copy of MCP request params
    },
    "id": "request-id"
  },
  "responseElements": {
    "content": [
      {
        "type": "text",
        "text": "example"
      }
    ],
    "isError": false
  },
  "requestID": "12345678-1234-1234-1234-123456789012",
  "eventID": "87654321-4321-4321-4321-210987654321",
```

```
  "readOnly": true,
  "recipientAccountId": "123456789012",
  "resources": [
    {
      "type": "AWS::S3::Bucket",
      "ARN": "arn:aws:s3:::example-bucket-1",
      "accountId": "123456789012"
    }
  ],
  "mcpEventDetails": {
    "sessionId": "sess_xyz789_YXJuOmF3czppYW06OjEyMzQ1Njc4OTAxMjpkZXZlbG9wZXI=",
    "mcpProtocolVersion": "2024-11-05",
    "serverVersion": "1.0.0",
    "mcpServerName": "aws-mcp.us-east-1.api.aws",
    "executionTimeMs": 250,
    ...
  }
}
```

Quotas for AWS MCP Server

Your AWS account has default quotas, formerly referred to as limits, for each AWS service. Unless otherwise noted, each quota is Region-specific. You can request increases for some quotas, and other quotas cannot be increased.

To view the quotas for AWS MCP Server, open the Service Quotas console. In the navigation pane, choose **AWS services** and select **AWS MCP Server**.

To request a quota increase, see Requesting a quota increase in the *Service Quotas User Guide*. If the quota is not yet available in Service Quotas, use the limit increase form.

Topics

Connection and session quotas

The following quotas apply to connections and sessions for AWS MCP Server.

Quota	Default value	Adjustable	Description
Concurrent connections per account per Region	100	No	The maximum number of concurrent MCP connections per AWS account in a single Region. This limit applies to authenticated requests.
Concurrent active sessions per account per Region	180	Yes	The maximum number of concurrent active sessions per AWS account in a single Region.

Quota	Default value	Adjustable	Description
Concurrent active sessions per user per Region	90	Yes	The maximum number of concurrent active sessions per IAM user or role in a single Region.

> **Note**
>
> The per-account and per-user quotas in the preceding table apply to authenticated requests. Unauthenticated requests are not subject to a per-account connection quota and have lower connection limits.

Throttling quotas

The following quotas apply to request rates for AWS MCP Server. Requests that exceed these limits are throttled with a 429 response.

Request rate limits for AWS MCP Server vary based on whether your request is authenticated. Authenticating your requests gives you more capacity:

- **Authenticated requests** – When you authenticate with your AWS credentials, your requests receive a per-AWS-account request rate. You can authenticate using either SigV4 through the MCP Proxy for AWS or OAuth 2.1 through AWS Sign-in. Authenticated requests also have access to the full set of AWS MCP Server tools, including the tools that run AWS API calls and execute scripts. To get more capacity, authenticate your requests.
- **Unauthenticated requests** – Requests that you send without AWS credentials share a lower rate limit. This limit applies per source IP address rather than per account. Unauthenticated access is limited to the read-only AWS knowledge tools, such as documentation search, documentation retrieval, and regional availability. Tools that run AWS API calls or execute scripts require authentication.

For more information about authenticating with AWS MCP Server, see the section called “OAuth authentication”.

Quota	Default value	Adjustable	Description
Authenticated requests per account per Region	10 per second (sustained)	No	The maximum number of authenticated requests per second to the AWS MCP Server per AWS account in a single Region. AWS MCP Server does not count unauthenticated requests against this per-account quota; it limits them separately per source IP address.
Unauthenticated requests per source IP address per Region	5 per second (sustained)	No	The maximum number of unauthenticated requests per second from a single source IP address in a single Region. This limit is lower than the authenticated per-account rate and applies per source IP address rather than per account. To get more capacity, authenticate your requests.

Session limits

The following limits apply to individual sessions with AWS MCP Server. These limits are not adjustable.

Limit	Value	Adjustable	Description
Maximum ephemeral storage retention	8 hours	No	The maximum duration that ephemeral compute storage is retained for a session. After this time, ephemeral storage is reclaimed.

Monitoring your quotas

AWS MCP Server publishes usage metrics to the `AWS/Usage` namespace in CloudWatch. You can use these metrics with the Service Quotas console to measure your utilization and create alarms as you approach a quota. The following usage metrics are available:

- `CallCount` (Resource: `Request`) – The number of requests made to the AWS MCP Server
- `ResourceCount` (Resource: `ConcurrentConnection`) – The number of concurrent connections
- `ResourceCount` (Resource: `AccountSessionCount`) – The number of concurrent active sessions per account
- `ResourceCount` (Resource: `UserSessionCount`) – The number of concurrent active sessions per user

For more information about these metrics, see Usage metrics in the AWS MCP Server CloudWatch metrics section.

> **Note**
>
> Requests that are throttled before reaching the AWS MCP Server are not reflected in the `CallCount` metric. For more information about this limitation, see Usage metrics.

For more information about monitoring your usage and setting up alarms, see AWS usage metrics in the *CloudWatch User Guide* and Service Quotas and Amazon CloudWatch in the *Service Quotas User Guide*.

Document history for the Agent Toolkit for AWS User Guide

The following table describes the documentation releases for Agent Toolkit for AWS.

Change	Description	Date
Serverless capability	Added documentation for AWS MCP Server capabilities, starting with the serverless capability, which provides read-only observability and troubleshooting for AWS Lambda functions and their connected resources.	August 25, 2026
OAuth 2.1 authentication support	Added OAuth 2.1 authentication through AWS Sign-in. OAuth-compatible MCP clients can now connect directly to AWS MCP Server without local proxy software. Added new managed policy `AWSMCPSignInOAuthAccessPolicy` and OAuth condition keys for governance.	July 8, 2026
AWS CLI support	The Agent Toolkit for AWS is now available in the AWS CLI. Added documentation for the `aws configure agent-toolkit` setup wizard and the `aws agent-toolkit` commands for installing, updating, removing, listing,	June 8, 2026

	and searching AWS skills from the AWS CLI.	
Multi-profile support	You can now configure multiple AWS profiles and switch between them on each tool call.	May 30, 2026
General availability of the Agent Toolkit for AWS	The Agent Toolkit for AWS is now generally available, with agent skills, plugins, and rules files. The user guide has been restructured to cover all toolkit components.	April 30, 2026
Deployment SOPs updates	Updated Deployment SOPs documentation with `deploy-supabase-app` guidance, and overall improved guidance.	February 18, 2026
Deployment SOPs	Added content describing the Deployment SOPs now available in Agent Toolkit for AWS	January 26, 2026
Initial release	Initial release of the Agent Toolkit for AWS User Guide	November 30, 2025