User Guide

AWS MCP Server (Preview)

AWS MCP Server (Preview): User Guide

Table of Contents

What is the AWS MCP Server (Preview)?

> **Important**
>
> The AWS MCP Server (Preview) is currently in preview release and is subject to change.

The AWS MCP Server (Preview) is a managed remote Model Context Protocol (MCP) server that provides AI assistants and agents with secure, authenticated access to AWS services through natural language interactions. You can use the AWS MCP Server (Preview) to perform complex, multi-step AWS tasks by combining real-time access to AWS documentation, syntactically correct API calls, and pre-built workflows called Agent SOPs that follow AWS best practices.

With the AWS MCP Server (Preview), you can ask AI assistants to provision infrastructure, troubleshoot issues, configure services, and manage AWS resources without needing to know specific API syntax or remember complex procedures. The server handles authentication through standard AWS Identity and Access Management (IAM) controls and provides comprehensive audit logging through AWS CloudTrail.

The AWS MCP Server (Preview) consolidates capabilities from existing MCP servers (AWS Knowledge MCP and AWS API MCP) into a single, unified interface that reduces configuration complexity while improving AI agent effectiveness across multi-service AWS workflows.

Topics

What can I do with the AWS MCP Server (Preview)?

You can use the AWS MCP Server (Preview) to do the following:

- **Execute multi-step AWS workflows** – Use Agent SOPs to perform complex tasks like setting up production VPCs, deploying serverless applications, or configuring monitoring across multiple AWS services with step-by-step guidance that follows AWS Well-Architected principles.

- **Get real-time AWS knowledge** – Search and retrieve up-to-date AWS documentation, API references, best practices, and regional availability information to inform your AI assistant's responses and decisions.
- **Make authenticated AWS API calls** – Execute most of the 15,000+ AWS APIs with SigV4 through your existing IAM roles and policies, with automatic syntax validation and error handling.
- **Troubleshoot AWS issues** – Analyze CloudWatch logs and CloudTrail events, investigate permission problems, debug application failures, and diagnose performance issues using guided workflows and access to comprehensive AWS knowledge sources.
- **Provision and configure infrastructure** – Create and configure AWS resources like VPCs, databases, compute instances, and storage with automated workflows that implement security best practices and proper resource tagging.
- **Manage costs** – Set up billing alerts, analyze resource usage, and understand resource costs and billing using pre-built procedures that follow AWS best practices.

How the AWS MCP Server (Preview) works

The AWS MCP Server (Preview) operates as a remote service that your MCP-compatible client connects to over HTTPS. When you ask your AI assistant to perform an AWS task, the server uses three integrated capabilities to complete your request:

- **Agent SOPs provide structured guidance** – The server searches its library of pre-built Agent SOPs to find workflows relevant to your task. These scripts contain step-by-step instructions that guide the AI through complex procedures while following AWS best practices and security guidelines.
- **Knowledge tools provide current information** – When the AI needs clarification or encounters unfamiliar concepts, it can search AWS documentation, retrieve API references, check regional availability, and access the latest AWS announcements to make more informed decisions.
- **API tools execute authenticated actions** – The server translates natural language requests into properly formatted AWS API calls, handles authentication using your IAM credentials, and executes the commands while providing detailed feedback about results and any errors.

For example, when you ask to "Create a full-stack TODO React web application on AWS", the agent finds the relevant Agent SOP. The SOP guides the agent through the entire process: setting up authentication using Amazon Cognito, provisioning APIs with AWS AppSync, configuring

compute resources, and creating an Amazon DynamoDB database while following AWS security best practices.

Authentication and authorization use your existing AWS IAM roles and policies, so you maintain full control over what resources and actions are available. All API calls are logged through AWS CloudTrail for audit visibility.

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

Pricing

With the AWS MCP Server (Preview), you pay only for the AWS resources you use and any applicable data transfer costs. The MCP server itself has no additional charges. 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.

Setting up your AWS MCP Server (Preview)

This section outlines how you can set up your AWS MCP Server (Preview).

Topics

Prerequisites

Before you begin, you must ensure that you have set up an AWS account.

Sign up for an AWS account

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.
2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Set up your AWS MCP Server (Preview)

To set up AWS MCP Server (Preview), use the steps in the following sections.

Topics

Step 1: (If applicable) Remove conflicting MCP servers

If you currently have AWS API MCP Server or AWS Knowledge MCP Server installed, we recommend removing them before setting up the AWS MCP Server (Preview) 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.
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 AWS credentials

Before connecting to AWS MCP Server (Preview), you need to configure AWS credentials on your local machine. The server uses these credentials to authenticate your requests.

You can use the SigV4 via Proxy to authenticate the AWS MCP Server (Preview). SigV4 via Proxy uses your available AWS credentials and requires the MCP Proxy for AWS.

> **Note**
>
> If your authentication step worked previously but you have encountered an authentication error, you might need to refresh your credentials and try again.

1. Install the AWS CLI by following the instructions at Installing the AWS CLI.
2. Configure your AWS credentials using one of these methods:

For users with AWS Management Console credentials (Recommended)

From the AWS CLI, run the following command:

```
aws login
```

> **Note**
>
> AWS Management Console credentials means that you have a username and password that allows you to sign in to the console. To use this method, you need the AWS CLI version `2.32.0` or later.

For SSO users

```
aws configure sso
```

Follow the prompts to set up your SSO configuration.

For IAM users

```
aws configure
```

Enter your Access Key ID, Secret Access Key, default region, and output format.

3. Test your configuration:

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

4. Install uv (if not already installed)

On 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"
```

Step 3: Configure your MCP client

Set your default AWS Region by adding the `--metadata` parameter with `AWS_REGION`. Without this setting, all AWS operations default to `us-east-1`.

> **Note**
>
> Replace `us-west-2` with your preferred default AWS Region.

Region behavior:

- Without `--metadata` and `AWS_REGION`: Operations default to `us-east-1`
- With `--metadata` and `AWS_REGION`: Operations use your specified Region
- In queries: You can override by specifying a Region (example: "list my EC2 instances in eu-west-1")

Amazon Kiro CLI

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

Cursor IDE

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

Claude Desktop

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

```
    }
  }
}
```

Step 4: Configure IAM permissions

If you're not using an administrator role, you must add specific permissions for AWS MCP Server (Preview) access.

> **Note**
>
> Skip this step if you're using an administrator role.

To configure IAM permissions

1. Open the IAM console at https://console.aws.amazon.com/iam/
2. Choose the user or role you configured in Step 2
3. Add this policy to grant AWS MCP Server (Preview) access:

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "aws-mcp:InvokeMcp",
                "aws-mcp:CallReadOnlyTool",
                "aws-mcp:CallReadWriteTool"
            ],
            "Resource": "*"
        }
    ]
}
```

Step 5: Test your connection

1. Start your MCP client (Kiro CLI, Cursor, Claude Desktop, etc.).

2. Wait for the MCP server to initialize (this may take a few minutes on 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 `retrieve_agent_sop` listed. For more information about the tools, see Understanding the MCP Server tools.

Understanding the MCP Server tools

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

AWS Knowledge Tools

- `aws___retrieve_agent_sop` - Retrieve the complete workflow for a specific Agent SOP. When called with an SOP name, returns the full step-by-step instructions. When called without arguments, provides guidance on using `aws___search_documentation` to discover available SOPs.
- `aws___search_documentation` - Search across all AWS documentation, including API references, best practices, service guides, and Agent SOPs. Use the topic filter to search Agent SOPs exclusively, or see SOPs 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___recommend` - Get content recommendations for AWS documentation pages based on related topics and commonly viewed content.
- `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___call_aws` - Execute authenticated AWS API calls with proper syntax validation and error handling. Supports most of the 15,000+ AWS APIs with automatic credential management.

  > **Note**
  >
  > APIs that require filesystem access or stream responses are not reliably supported.

- `aws___suggest_aws_commands` - Get descriptions and syntax help for relevant AWS APIs, including newly released APIs that may not be in the AI model's training data.

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

Agent SOPs

Agent SOPs are pre-built, tested workflows that guide AI assistants through complex multi-step AWS tasks. These automated procedures eliminate the guesswork from common AWS operations by providing step-by-step instructions that follow AWS best practices and security guidelines.

- Proven workflows that have been tested in real AWS environments
- Security considerations included in procedures
- AWS Well-Architected principles applied consistently
- Error handling guidance for common issues
- Success validation criteria to ensure tasks complete correctly

For example, when you ask to "create a production-ready VPC," the **Create Production VPC Multi-AZ** Agent SOP guides your AI assistant through each step. It creates subnets across multiple availability zones, configures route tables, sets up NAT gateways, and applies proper security groups—all following AWS networking best practices.

Available Agent SOPs

The AWS MCP Server (Preview) includes Agent SOPs for common AWS tasks. Here are some examples:

- **Deployment** — Creates, prepares and deploys production-ready infrastructure as code for existing web applications
- **Infrastructure setup** — Create production-ready VPCs with multi-AZ subnets and NAT gateways
- **Security configuration** — Apply comprehensive security controls and audit logging to S3 buckets
- **Database management** — Create Aurora database clusters with managed credentials and best practices
- **Monitoring setup** — Configure SNS notifications for CloudWatch alarms and monitoring alerts
- **Application management** — Build and deploy full-stack web and mobile applications with AWS Amplify's framework and hosting capabilities

You can see which Agent SOPs are available by asking your AI assistant: `What Agent SOPs do you have available?`

If you're unsure how to complete a task, ask the agent to develop a plan with the current tools available.

Deployment SOPs

The AWS MCP Server (Preview) includes Standard Operating Procedures (SOPs) that deploy applications to AWS. These SOPs analyze your application, generate Infrastructure as Code (IaC) using the AWS Cloud Development Kit (CDK), and deploy it through AWS CloudFormation.

Deployment SOPs support single-page applications, static site generators, Supabase-backed applications (such as Lovable.dev and Bolt.new), and static websites. These SOPs can deploy applications with minimal prompting. For complex applications, your coding agent may require additional information or iterations to complete the deployment.

Your coding agent is guided by AWS security best practice recommendations from the SOPs, providing a secure starting point from where you can review and customize for your requirements.

Quick start

1. Install the AWS MCP Server (Preview). For instructions, see Setting up your AWS MCP Server (Preview).
2. Log in to the AWS CLI. For instructions, see Configuring the AWS CLI.
3. Prompt your coding agent: `Deploy my app to AWS`

Available deployment types

Frontend applications

Deploys applications built with modern frontend frameworks to Amazon S3 and Amazon CloudFront. Generates CDK infrastructure code and deploys it through AWS CloudFormation, providing a shareable preview URL.

Supported application types: React, Vue, Angular, SvelteKit, Next.js (static export), Nuxt 2/3, Gatsby, Hugo, Jekyll, Docusaurus, Astro, Eleventy. Other frameworks may require you to provide additional guidance to your coding agent, or perform manual updates after the deployment.

For more information, see Frontend applications.

Supabase applications

Deploys applications built with Supabase to your AWS account. Your database and authentication remain in Supabase, while Edge Functions migrate to AWS Lambda and Amazon API Gateway. Stores secrets in AWS Secrets Manager. Generates CDK infrastructure code and deploys through CloudFormation, providing a shareable preview URL.

Supported application types: Applications with environment-based Supabase configuration (`supabase/config.toml`), such as Lovable.dev and Bolt.new.

For more information, see Supabase applications.

Set up CodePipeline

Creates a CI/CD pipeline using AWS CodePipeline that automatically builds, tests, and deploys your application when changes are pushed to your source repository.

Supported application types: All applications deployed using Deployment SOPs.

For more information, see Set up CodePipeline.

How the SOPs work

The SOPs provide step-by-step instructions that your coding agent follows. Your coding agent inspects the application, generates CDK infrastructure code, and deploys using CloudFormation. Application code is modified to support deployment to AWS, which includes changing how secrets are obtained, how edge functions are wrapped for AWS Lambda, and, in some cases, frontend build configurations. The SOPs instruct your coding agent to avoid modifying application code unless required for deployment.

Your coding agent may ask your permission to use a tool or request additional information such as application secret keys. Where possible, your coding agent derives information from available source code.

The SOPs generate documentation in your repository to track deployment progress and provide context for future deployments.

In some situations, you may need to prompt your coding agent to fix inconsistencies, particularly with larger applications or older models with smaller context windows.

Prerequisites

AI model requirements

Testing showed best results with the following models:

- Anthropic Claude Opus 4.6 (200k, 1M)
- Anthropic Claude Opus 4.5 (200k)
- Anthropic Claude Sonnet 4.5
- OpenAI GPT-5.2-Codex
- OpenAI GPT-5.3-Codex
- Google Gemini 3 Pro

Tooling prerequisites

Before you begin, ensure that you have an AWS account with appropriate permissions. For instructions, see Setting up your AWS MCP Server (Preview).

Additional prerequisites vary depending on your application. The SOP guides your coding agent to verify these automatically:

- AWS MCP Server (Preview) — configured in your AI coding assistant (such as Kiro or Cursor). For setup instructions see Setting up your AWS MCP Server (Preview)
- Git CLI — Installed and configured
- AWS CLI — Configured with valid credentials. For setup instructions see Set up the AWS CLI
- AWS CDK CLI — Version 2.x configured. For setup instructions see Getting started with the AWS CDK
- Package manager — npm, yarn, pnpm, or bun as required by your project

Security features

Note

The AWS shared responsibility model applies to data protection when using Deployment SOPs in AWS MCP Server (Preview). Always review generated infrastructure code before

deploying. Your coding agent may not apply all recommended security defaults. For more information, see Data protection.

Deployment SOPs prompt your coding agent to implement the following security best practices:

- *Private Amazon S3 buckets* — Blocks all public access to stored content
- *Encryption at rest* — Enables Amazon S3 managed encryption for all stored content
- *HTTPS enforcement* — Requires TLS 1.2 or higher with automatic HTTPS redirect
- *Origin Access Control (OAC)* — Configures Amazon CloudFront to access Amazon S3 through the AWS internal network
- *AWS IAM least privilege* — Applies minimal required permissions for each service

When you combine it with the CodePipeline SOP, you have access to additional quality controls that include:

- *Security scanning* — Detects exposed secrets in your codebase during each build
- *Quality gates* — Runs available unit tests and static code analysis before deployment

Limitations

The Deployment SOPs work using local code agent capabilities, and depend upon the LLM you select. Large applications, such as those with over 25 APIs functions, may have reliability issues. If that happens, prompt your coding agent to test the application or API and fix the problems it finds.

Pricing

With Deployment SOPs, you pay only for the AWS resources you use and any applicable data transfer costs. The Deployment SOPs have no additional charges. 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.

Frontend applications

This SOP analyzes your frontend application and generates AWS CDK infrastructure code. The SOP then deploys the infrastructure to AWS. After deployment, the SOP provides a shareable URL for your website.

For prerequisites and security information, see AWS Deployment SOPs.

Supported application types

- Single-page applications (SPAs): React, Vue, Angular, SvelteKit
- Static site generators (SSGs): Next.js (static export), Nuxt 2/3, Gatsby, Hugo, Jekyll, Docusaurus, Astro, Eleventy
- Static websites

> **Note**
>
> Applications that require server-side rendering (SSR) are not supported by this SOP. If your application uses SSR, consider deploying with AWS Lambda, Amazon ECS, or AWS App Runner.

Example prompt

To start a deployment, prompt your coding agent: `Deploy my application`.

Steps your coding agent takes

Your coding agent commits changes after each significant step to a new `deploy-to-aws` branch.

1. Scans the project to detect the framework, build configuration, and output directory
2. Validates prerequisites (AWS credentials, package manager, CDK CLI)
3. Creates a new branch (`deploy-to-aws`)
4. Generates CDK infrastructure code for Amazon S3 and Amazon CloudFront
5. Builds your application and deploys infrastructure through AWS CloudFormation
6. Validates the deployment and provides a URL for your application
7. Records deployment details, and follow-up instructions, in your repository in `AGENTS.md` and `DEPLOYMENT.md`

How it works

Your coding agent analyzes your application to determine the framework, build output directory, and routing strategy. Based on this analysis, the agent generates CDK infrastructure code using the CloudFrontToS3 AWS Solutions Construct.

The generated infrastructure creates an Amazon S3 bucket to store your compiled application. It also creates an Amazon CloudFront distribution to serve the application globally. CloudFront is configured with Origin Access Control (OAC) so that Amazon S3 is accessed only through the AWS internal network, keeping the bucket private.

Your coding agent determines the correct CloudFront routing configuration based on your framework:

- Single-page applications use CloudFront error responses to redirect navigation requests to `index.html`
- Static site generators use CloudFront Functions to rewrite URLs. Depending on your framework's trailing slash configuration, requests are rewritten to either `/path/index.html` or `/path.html`

A personal preview stack is provisioned using CloudFormation with the naming pattern `{AppName}Frontend-preview-{username}`. Preview environments use non-production defaults. These defaults include `DESTROY` removal policies and short log retention. Production environments use `RETAIN` removal policies and longer retention periods.

The SOP prompts your coding agent to apply security best practices. These practices include private S3 buckets, Content Security Policy (CSP) headers, HTTPS enforcement, and managed security response headers. Always review the generated configuration before deploying to production environments.

For production environments with CI/CD, see Set up CodePipeline.

Troubleshooting

Application type not supported

Verify that all prerequisites are met. If your application meets the prerequisites but is reported as unsupported, prompt your coding agent to attempt the deployment. Minor adjustments may be sufficient.

For any other troubleshooting issues, you can contact AWS Support or post your question on re:Post and tag it to the AWS MCP Server (Preview) to ask the community.

Supabase applications

This SOP deploys applications that use Supabase (such as Lovable.dev and Bolt.new) to AWS. The SOP migrates Supabase Edge Functions to AWS Lambda and Amazon API Gateway. It stores secrets in AWS Secrets Manager and hosts the frontend on Amazon S3 and Amazon CloudFront. After deployment, the SOP provides a shareable URL for your application.

For prerequisites and security information, see AWS Deployment SOPs.

> **Important**
>
> This SOP keeps your database, authentication, and storage in Supabase. You can use an existing project or create a new one. Hosting, Edge Functions, and secrets management migrate to AWS. To fully migrate to AWS, you must manually move your Supabase data.

Supported application types

- Applications with environment-based Supabase configuration (`supabase/config.toml`):
  - Single-page applications (SPAs): React, Vue, Angular, SvelteKit
  - Static site generators (SSGs): Next.js (static export), Nuxt 2/3, Gatsby, Hugo, Jekyll, Docusaurus, Astro, Eleventy

Example prompt

To start a deployment, prompt your coding agent: `Deploy my application`.

Steps your coding agent takes

Your coding agent commits changes after each significant step to a new `deploy-to-aws` branch.

1. Scans the project to detect the framework, build configuration, and Supabase setup
2. Validates prerequisites (AWS credentials, Supabase CLI, package manager, CDK CLI)
3. Creates a new branch (`deploy-to-aws`)
4. Analyzes Supabase Edge Functions, database configuration, and required secrets

5. Configures the Supabase project — uses your existing project or creates a new one under your existing subscription
6. Migrates Supabase Edge Functions (Deno, TypeScript) to AWS Lambda (Node.js, TypeScript)
7. Converts AI or LLM functions to Amazon Bedrock, if detected. This uses `InvokeModel` or `InvokeModelWithResponseStream` as needed
8. Updates all Edge Function references in your application code to use new API endpoints
9. Stores application secrets in AWS Secrets Manager
10. Generates CDK infrastructure code for Lambda, API Gateway, Amazon S3, and Amazon CloudFront
11. Deploys infrastructure through AWS CloudFormation
12. Validates each migrated function through the deployed API
13. Provides a URL for your application
14. Records deployment details in your repository

How it works

Your coding agent analyzes your application to identify Supabase Edge Functions, database configuration, and required secrets. The SOP can create a new Supabase project if needed. The new project is created under your existing subscription, and the SOP pushes database migrations to it.

Each Supabase Edge Function is migrated from Deno/TypeScript to Node.js/TypeScript. The migrated function is deployed as an AWS Lambda function fronted by Amazon API Gateway. The SOP uses NodejsFunction with esbuild to bundle each function. If your application uses AI or LLM features, those functions are converted to use Amazon Bedrock. Application secrets, including Supabase credentials, are stored in AWS Secrets Manager. Lambda functions access these secrets at runtime.

For the frontend, the SOP generates the same Amazon S3 and Amazon CloudFront infrastructure as a frontend deployment. The SOP adds an additional CloudFront behavior that proxies `/api/*` requests to API Gateway. This routing approach avoids cross-origin issues and presents a single domain to your users. All Supabase Edge Function references in your application code are updated to use the new `/api/*` endpoints.

After deployment, the SOP updates Supabase authentication redirect URLs to include the CloudFront domain, so authentication flows work correctly with the new hosting.

The SOP prompts your coding agent to apply security best practices to all generated resources. Always review the generated configuration before deploying to production environments.

For production environments with CI/CD, see Set up CodePipeline.

Services used

Amazon CloudFront, Amazon S3, AWS Lambda, Amazon API Gateway, AWS Secrets Manager, AWS CloudFormation, AWS IAM, and optionally Amazon Bedrock.

Troubleshooting

Application type not supported

Verify that all prerequisites are met. If your application meets the prerequisites but is reported as unsupported, prompt your coding agent to attempt the deployment. Minor adjustments may be sufficient.

For any other troubleshooting issues, you can contact AWS Support or post your question on re:Post and tag it to the AWS MCP Server (Preview) to ask the community.

Set up CodePipeline

This SOP creates a CI/CD pipeline using AWS CodePipeline. The pipeline automatically builds, tests, and deploys your application when changes are pushed to a source repository branch.

For prerequisites and security information, see AWS Deployment SOPs.

Requirements

Your application must already be configured as a CDK application with existing infrastructure code. This SOP works best after deploying with Frontend applications or Supabase applications.

> **Important**
>
> This SOP requires you to manually approve an AWS CodeConnections resource in your web browser. You need permissions to install and configure the connection in your repository or organization.

Example prompt

To set up a pipeline, prompt your coding agent with the following: `Set up a pipeline for my application`.

Steps your coding agent takes

Your coding agent commits changes after each significant step to the `deploy-to-aws` branch.

1. Scans the project to detect existing CDK infrastructure, stacks, and application configuration
2. Identifies available quality checks (linting, unit tests) and verifies they pass locally
3. Presents a detection summary and asks you to confirm the configuration
4. Creates an AWS CodeConnections resource to connect AWS to your source repository
5. Creates production secrets in AWS Secrets Manager, if your application uses Lambda functions
6. Generates CDK infrastructure code for the pipeline
7. Deploys the pipeline stack through AWS CloudFormation
8. Prompts you to authorize the connection in the AWS console
9. Verifies the pipeline triggers and runs successfully
10. Records pipeline configuration and deployment details in your repository

Manual steps

During Step 8, you must complete authorization in the AWS console:

1. Open the AWS CodeConnections console
2. Find the pending connection for your application
3. Choose **Update pending connection**
4. Authorize and install the connector for your repository

How it works

Your coding agent verifies your application has existing CDK infrastructure code. The agent then generates a pipeline stack using the CDK Pipelines module (`aws-cdk-lib/pipelines`). The pipeline is self-mutating. When you push changes to pipeline infrastructure code, the pipeline automatically updates itself.

The pipeline uses AWS CodeConnections to authenticate with your source repository. When changes are pushed to the configured branch, the pipeline executes the following stages:

1. *Source* — Pulls source code from your repository through the CodeConnections resource
2. *Build (Synth)* — Installs dependencies, runs quality checks, builds the application, and synthesizes CloudFormation templates using CDK
3. *Update pipeline* — Self-mutation stage that updates the pipeline if its own infrastructure code changed
4. *Assets* — Publishes file and Docker image assets required by the stacks
5. *Deploy* — Deploys your application stacks to a production environment

The pipeline initially triggers on the `deploy-to-aws` branch. You can reconfigure the pipeline to trigger on `main` or another branch. To reconfigure, update the `branchName` context variable in the CDK configuration.

Quality checks are included only if they pass locally during setup. End-to-end tests are not included in the pipeline. The pipeline uses Secretlint to scan for exposed secrets in your codebase during each build. As part of the AWS Shared Responsibility Model, you should rotate exposed secrets immediately.

If your application includes Lambda functions, the SOP creates a separate production secret in AWS Secrets Manager (`{AppName}/prod/secrets`) and deploys both Lambda and frontend stacks through the pipeline.

The SOP prompts your coding agent to apply security best practices. Always review the generated pipeline configuration before deploying.

Troubleshooting

For troubleshooting issues, you can contact AWS Support or post your question on re:Post and tag it to the AWS MCP Server (Preview) to ask the community.

Security in AWS MCP Server (Preview)

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 (Preview), 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.

This documentation helps you understand how to apply the shared responsibility model when using AWS MCP Server (Preview). The following topics show you how to configure AWS MCP Server (Preview) 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 (Preview) resources.

Topics

Data protection in AWS MCP Server (Preview)

The AWS shared responsibility model applies to data protection in AWS MCP Server (Preview). 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 the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

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 AWS MCP Server (Preview) 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 (Preview) doesn't support FIPS endpoints.

Identity and access management for AWS MCP Server (Preview)

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 AWS MCP Server (Preview) 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 (Preview) identity and access)
- **Service administrator** - determine user access and submit permission requests (see How AWS MCP Server (Preview) works with IAM)
- **IAM administrator** - write policies to manage access (see Identity-based policy examples for AWS MCP Server (Preview))

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 (Preview) works with IAM

Before you use IAM to manage access to AWS MCP Server (Preview), learn what IAM features are available to use with AWS MCP Server (Preview).

IAM feature	AWS MCP Server (Preview) support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes

IAM feature	AWS MCP Server (Preview) support
Policy condition keys	Yes
ACLs	No
ABAC (tags in policies)	Partial
Temporary credentials	Yes
Principal permissions	Yes
Service roles	Yes
Service-linked roles	No

To get a high-level view of how AWS MCP Server (Preview) and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for AWS MCP Server (Preview)

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles 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*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for AWS MCP Server (Preview)

To view examples of AWS MCP Server (Preview) identity-based policies, see Identity-based policy examples for AWS MCP Server (Preview).

Resource-based policies within AWS MCP Server (Preview)

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are 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. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for AWS MCP Server (Preview)

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Include actions in a policy to grant permissions to perform the associated operation.

To see a list of AWS MCP Server (Preview) actions, see Actions Defined by AWS MCP Server (Preview) in the *Service Authorization Reference*.

Policy actions in AWS MCP Server (Preview) use the following prefix before the action:

```
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      ":action1",
      ":action2"
         ]
```

To view examples of AWS MCP Server (Preview) identity-based policies, see Identity-based policy examples for AWS MCP Server (Preview).

Policy resources for AWS MCP Server (Preview)

Supports policy resources: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. As a best practice, specify a resource using its Amazon Resource Name (ARN). For actions that don't support resource-level permissions, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of AWS MCP Server (Preview) resource types and their ARNs, see Resources Defined by AWS MCP Server (Preview) in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions Defined by AWS MCP Server (Preview) .

To view examples of AWS MCP Server (Preview) identity-based policies, see Identity-based policy examples for AWS MCP Server (Preview).

Policy condition keys for AWS MCP Server (Preview)

Supports service-specific policy condition keys: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element specifies when statements execute based on defined criteria. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of AWS MCP Server (Preview) condition keys, see Condition Keys for AWS MCP Server (Preview) in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions Defined by AWS MCP Server (Preview) .

To view examples of AWS MCP Server (Preview) identity-based policies, see Identity-based policy examples for AWS MCP Server (Preview).

ACLs in AWS MCP Server (Preview)

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with AWS MCP Server (Preview)

Supports ABAC (tags in policies): Partial

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes called tags. You can attach tags to IAM entities and AWS resources, then design ABAC policies to allow operations when the principal's tag matches the tag on the resource.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with AWS MCP Server (Preview)

Supports temporary credentials: Yes

Temporary credentials provide short-term access to AWS resources and are automatically created when you use federation or switch roles. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM and AWS services that work with IAM in the *IAM User Guide*.

Cross-service principal permissions for AWS MCP Server (Preview)

Supports forward access sessions (FAS): Yes

Forward access sessions (FAS) use the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. For policy details when making FAS requests, see Forward access sessions.

Service roles for AWS MCP Server (Preview)

Supports service roles: Yes

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Warning**
>
> Changing the permissions for a service role might break AWS MCP Server (Preview) functionality. Edit service roles only when AWS MCP Server (Preview) provides guidance to do so.

Service-linked roles for AWS MCP Server (Preview)

Supports service-linked roles: No

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see AWS services that work with IAM. Find a service in the table that includes a Yes in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Identity-based policy examples for AWS MCP Server (Preview)

By default, users and roles don't have permission to create or modify AWS MCP Server (Preview) resources. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by AWS MCP Server (Preview), including the format of the ARNs for each of the resource types, see Actions, Resources, and Condition Keys for AWS MCP Server (Preview) in the *Service Authorization Reference*.

Topics

Policy best practices

Identity-based policies determine whether someone can create, access, or delete AWS MCP Server (Preview) resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and

functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.

- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the AWS MCP Server (Preview) console

To access the AWS MCP Server (Preview) console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the AWS MCP Server (Preview) resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the AWS MCP Server (Preview) console, also attach the AWS MCP Server (Preview) `ConsoleAccess` or `ReadOnly` AWS managed policy to the entities. For more information, see Adding permissions to a user in the *IAM User Guide*.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
```

```
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS managed policies for AWS MCP Server (Preview)

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see AWS managed policies in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when

a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ReadOnlyAccess** AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see AWS managed policies for job functions in the *IAM User Guide*.

AWS managed policy: AWSMcpServiceActionsFullAccess

You can attach the `AWSMcpServiceActionsFullAccess` policy to your IAM identities. This policy grants full access to all MCP service actions. This policy does not grant access to the actions taken by the MCP, only the MCP actions themselves.

To view the permissions for this policy, see AWSMcpServiceActionsFullAccess in the *AWS Managed Policy Reference*.

AWS MCP Server (Preview) updates to AWS managed policies

View details about updates to AWS managed policies for AWS MCP Server (Preview) since this service began tracking these changes.

Change	Description	Date
AWSMcpServiceActionsFullAccess – New policy	AWS MCP Server (Preview) added a new policy to grant full access to all MCP service actions.	November 30, 2025
AWS MCP Server (Preview) started tracking changes	AWS MCP Server (Preview) started tracking changes for its AWS managed policies.	November 30, 2025

Troubleshooting AWS MCP Server (Preview) identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with AWS MCP Server (Preview) and IAM.

Topics

I am not authorized to perform an action in AWS MCP Server (Preview)

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional *`:GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to
 perform: :GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the *`:GetWidget`* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to AWS MCP Server (Preview).

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in AWS MCP Server (Preview). However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my AWS MCP Server (Preview) resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether AWS MCP Server (Preview) supports these features, see How AWS MCP Server (Preview) works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Compliance validation for AWS MCP Server (Preview)

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 (Preview)

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, AWS MCP Server (Preview) offers several features to help support your data resiliency and backup needs.

Monitoring AWS MCP Server (Preview)

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS MCP Server (Preview) and your other AWS solutions. AWS provides the following monitoring tools to watch AWS MCP Server (Preview), 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 (Preview) automatically publishes metrics to CloudWatch at no additional cost. For more information, see AWS MCP Server (Preview) 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 (Preview) CloudWatch metrics

AWS MCP Server (Preview) 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 (Preview) 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___call_aws` or `aws___list_regions`) and track their success rates.

Available metrics

The following metrics are available for AWS MCP Server (Preview). 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
`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___call_aws`, `aws___list_regions`, or `aws___retrieve_agent_sop`. For a complete list of available tools, see Understanding the MCP Server tools.

Using metrics

You can use AWS MCP Server (Preview) 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 (Preview) metrics:

Calculate success rate for API calls

: Filter metrics by `Tool Name = aws___call_aws` 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.

Logging AWS MCP Server (Preview) API calls using AWS CloudTrail

AWS MCP Server (Preview) 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 (Preview). CloudTrail captures all API calls for AWS MCP Server (Preview) as events. The calls captured include calls from the AWS MCP Server (Preview) console and code calls to the AWS MCP Server (Preview) 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 (Preview). 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 (Preview), 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 (Preview) information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in AWS MCP Server (Preview), 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 (Preview), 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 (Preview) actions are logged by CloudTrail and are documented in the AWS MCP Server (Preview) API Reference. For example, calls to the `ACTION_1`, `ACTION_2` and `ACTION_3` actions generate entries in the CloudTrail log files.

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 (Preview) 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: `retrieve_agent_sop`
- CloudTrail logs show: `retrieve_agent_scripts`

This occurs because CloudTrail logs the internal tool names used by the server, while MCP clients may display user-friendly names.

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

```
{
  "eventVersion": "1.08",
  "eventCategory": "Data",
  "eventType": "AwsMcpEvent",
  "userIdentity": {
      ...
  },
  "eventTime": "...",
  "eventSource": "aws-mcp.us-east-1.api.aws",
  "eventName": "CallTool",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "...",
```

```
  "delegatedViaAWS": "...",
  "requestParameters": {
    "method": "call_aws",
    "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,
    ...
  }
}
```

Document history for the AWS MCP Server (Preview) User Guide

The following table describes the documentation releases for AWS MCP Server (Preview).

Change	Description	Date
Deployment SOPs updates	Updated Deployment SOPs documentation with `deploy-supabase-app` guidance, and overall improved guidance.	February 18, 2026
Initial release	Initial release of the AWS MCP Server (Preview) User Guide	November 30, 2025
Deployment SOPs	Added content describing the Deployment SOPs now available in AWS MCP Server (Preview)	January 26, 2025