User Guide

AWS DevOps Agent

AWS DevOps Agent: User Guide

Table of Contents

About AWS DevOps Agent

AWS DevOps Agent is a frontier agent that resolves and proactively prevents incidents, continuously improving reliability and performance.

AWS DevOps Agent investigates incidents and identifies operational improvements as an experienced DevOps engineer.

The agent works by:

- Learning your resources and their relationships.
- Working with your observability tools, runbooks, code repositories, and CI/CD pipelines.
- Correlating telemetry, code, and deployment data to understand relationships between your application resources.
- Supporting applications in multicloud and hybrid environments.

Key features

AWS DevOps Agent provides comprehensive incident response and prevention capabilities through the following features:

Always-on, autonomous incident response

AWS DevOps Agent autonomously investigates issues the moment they occur:

- **Automated incident investigation** – Begins investigating immediately when an alert or support ticket comes in
- **Interactive investigation chat** – Initiate and guide investigations using natural language in the Dev Op Agent Space web app
- **Detailed mitigation plans** – Provides specific actions to resolve incidents, validate success, and revert changes if needed
- **Automated incident coordination** – Routes observations, findings, and mitigation steps through your preferred communication channels like Slack and ServiceNow
- **AWS Support integration** – Create AWS Support cases directly from an investigation with immediate context provided to AWS Support experts

Prevent future incidents

AWS DevOps Agent analyzes patterns across historical incidents to help you move from reactive firefighting to proactive operational improvement:

- **Targeted recommendations** – Delivers specific, actionable improvements that strengthen four key areas: observability (monitoring, alerting, logging), infrastructure optimization (autoscaling, capacity tuning), and deployment pipeline enhancement (testing, validation).
- **Continuous learning** – Refines recommendations based on your team's feedback

Get more from your DevOps tools

AWS DevOps Agent integrates with your existing tools without changing your workflows:

- **Application resource mapping** – Builds a topology graph of your application resources and their relationships
- **Built-in integrations** – Works with popular observability tools (Amazon CloudWatch, Dynatrace, Datadog, New Relic, and Splunk), code repositories, and CI/CD pipelines (GitHub Actions and repositories, GitLab workflows and repositories)
- **Custom tool integration** – Extend capabilities by connecting to your own Model Context Protocol (MCP) servers for additional tools

How AWS DevOps Agent works

AWS DevOps Agent operates through a dual-console architecture. Administrators use the AWS Management Console to create and manage Agent Spaces, configure integrations, and set up access controls. Operations teams use the AWS DevOps Agent web app for day-to-day incident response and investigation activities. The web app is where operators can interact with agent investigations, browse cross-account application topology, and learn about preventative improvements to observability, code, pipelines, and infrastructure architectures. To learn more, see the section called “Preventing future incidents”.

The service is organized around Agent Spaces, which are logical containers that define what AWS DevOps Agent can access and investigate. Each Agent Space contains your AWS account configurations, third-party tool integrations, and access permissions. To learn more, see the section called “What are DevOps Agent Spaces?”.

AWS DevOps Agent automatically builds an application topology that maps your resources and their relationships. This topology helps the service understand your application architecture during investigations. To learn more, see the section called "What is a DevOps Agent Topology?".

Benefits

- **Reduce mean time to resolution (MTTR)** – Autonomous investigation starts immediately, accelerating incident resolution from hours to minutes
- **Prevent recurring incidents** – Targeted recommendations address root causes and strengthen system resilience
- **Improve operational efficiency** – Free your team from repetitive investigation tasks to focus on innovation
- **Work within existing workflows** – Integrates with your existing tools and processes without disruption

What is a DevOps Agent Web App?

AWS DevOps Agent uses a dual-console architecture that separates administrative functions from day-to-day operational activities. This design enables administrators to configure the service while operations teams focus on incident response and prevention.

Consoles

AWS DevOps Agent provides two distinct interfaces:

- **AWS Management Console** – Administrators use the AWS Management Console to set up and manage AWS DevOps Agent. In this console, you can the section called "Creating an Agent Space" connect AWS services and third-party tools, and manage access permissions for your organization.
- **DevOps Agent web app -** Operations teams use DevOps Agent Space web apps for daily incident response activities. This standalone application provides an interface where on-call engineers can launch investigations, interact with the agent through natural language chat, view application topologies, and review incident prevention recommendations.

Web app capabilities

The DevOps Agent web app provides the following primary capabilities:

- **Incident Response –** The page is where you create and track incident investigations as well as generate mitigation plans to resolve incidents.
- **Incident Prevention –** Found in the Prevention tab, this is where you will find recommendations to improve your observability posture, delivery processes, and infrastructure architecture to prevent future incidents.
- **DevOps Center –** The DevOps Center tab provides an interactive visual representation of the account resources and their relationships across all of the resources in the connected accounts. You can view the topology with different levels of detail.
- **Runbooks** – Natural language explanations of your company's in-house processes, high-level application architectures, and desired agent behaviors

Authentication

AWS DevOps Agent supports flexible authentication methods to accommodate different organizational requirements:

- **IAM Identity Center integration (User access)** – Organizations can use AWS Identity Center (IAM Identity Center) to centrally manage user access to the DevOps Agent Space web apps. IAM Identity Center can federate with external identity providers through standard OIDC and SAML protocols, including providers like Okta, Ping Identity, and Microsoft Entra ID. This method supports multi-factor authentication from your identity provider.
- **IAM authentication link (Admin access)** – An alternative method provides direct access to the web app from the AWS Management Console using your existing console session. This option is useful This option is useful before implementing full Identity Center integration, but sessions are limited to 10 minutes.

What are DevOps Agent Spaces?

A DevOps Agent Space is a logical container that defines the tools and infrastructure that AWS DevOps Agent has access to. Each Agent Space operates independently with its own AWS account access, third-party integrations, and user permissions.

An Agent Space represents the boundary of what AWS DevOps Agent can access and investigate during incident response. When you create an Agent Space, you define which AWS accounts the agent can access, which external tools it can connect to, and which users in your organization can interact with the agent.

Each Agent Space functions as an independent deployment of AWS DevOps Agent. You configure the Agent Space through the AWS Management Console, while your operations teams use the Agent Space's web app to conduct investigations and review recommendations within that space.

How Agent Spaces are isolated

Agent Spaces maintain isolation to ensure security and prevent unintended access across different environments or teams:

- **AWS account isolation** – Each Agent Space uses dedicated IAM roles that grant access only to specific AWS accounts and resources. The agent cannot access AWS resources outside of those explicitly configured for the Agent Space.
- **User access isolation** – You control which users or groups can access each Agent Space. This allows you to align access permissions with your organizational structure, ensuring teams only interact with their designated Agent Spaces.
- **Data isolation** – Investigation data, incident history, and recommendations are maintained separately within each Agent Space. Information from one Agent Space is not visible or accessible from another Agent Space.

Agent Space Web App

Each Agent Space has a dedicated web app that is accessible outside of the AWS Management Console. See the section called "What is a DevOps Agent Web App?" to learn more about the web app.

When to use multiple Agent Spaces

Consider creating multiple Agent Spaces to support different organizational needs:

- **Team separation** – Create dedicated Agent Spaces for different application teams or business units to maintain clear ownership boundaries in the Agent Space.
- **Environment isolation** – Separate production and non-production environments into different Agent Spaces to prevent accidental cross-environment access.

- **Service boundaries** – Align Agent Spaces with specific services or application boundaries to keep investigations focused and relevant.
- **Compliance requirements** – Configure separate Agent Spaces with different access controls or data residency settings to meet regulatory requirements.

> **Note**
>
> When creating multiple Agent Spaces, you can use a dedicated AWS account as the primary account for an Agent Space and connect distinct application accounts as secondary accounts. This approach allows you to maintain granular access controls while ensuring that each Agent Space can access only the resources specific to its intended scope, even when using automatic role creation.

What is a DevOps Agent Topology?

AWS DevOps Agent's automatically discovers and visualizes the resources and relationships within your applications and uses the resulting topology to understand your infrastructure during incident investigations and when making preventative recommendations.

How topology graphs are created

AWS DevOps Agent builds topology graphs through several automated processes:

- **Resource discovery** – The agent automatically scans your AWS accounts to identify resources like compute instances, storage services, networking components, and databases that are part of your applications.
- **Relationship detection** – The agent analyzes configuration data, CloudFormation stacks, and resource tags to determine how resources relate to one another.
- **Code and deployment mapping** – When connected to CI/CD pipelines, the agent links infrastructure resources back to their deployment processes and changed application and infrastructure code.
- **Observability behavior mapping** – Data from observability systems such as Amazon CloudWatch Application Signals and Dynatrace are used to identify observed behaviors that indicate relationships between resources.

Key capabilities

Resource mapping provides several capabilities that enhance incident investigation and prevention:

- **Interactive visualization** – Explore your application topology through an interactive graph in the Operator Web App. You can zoom and navigate the topology to understand complex relationships between resources.
- **Contextual investigation** – During incident investigations, AWS DevOps Agent is assisted by the resource topology to identify affected components, understand blast radius, and trace the impact path through your systems.
- **Root cause analysis** – The detailed understanding of resource relationships helps pinpoint where issues originate, even in complex distributed systems with many interdependencies.
- **Impact assessment** – When analyzing incidents, the agent can better determine which downstream services might be affected by identifying dependency chains in the topology.
- **Preventative recommendations** – The agent uses topology insights to make targeted recommendations for resilience improvements, suggesting changes that will have the most significant impact on system stability.

Topology views

The topology visualization in DevOps Center page in the Operator Web App offers multiple levels of detail:

- **System view** – Shows high-level account and region boundaries
- **Container view** – Displays deployment stacks like CloudFormation stacks that contain related resources
- **Resource view** – Shows the complete view with all resources and their relationships

Resource discovery

Resources are discovered through two methods:

- **CloudFormation stacks** – The agent will list all of the CloudFormation stacks and their resources in the primary AWS account as well as an connected secondary accounts. This is supported for any infrastructure-as-code tooling that uses CloudFormation for deployment, including Cloud Development Kit (CDK).

- **Resource Tags** – For resources not deployed from CloudFormation, you can specify a list of AWS Tag keys and value pairs to include in the resource topology. This is useful to identify application boundaries for applications deployed through the AWS Management Console, the AWS service APIs, or other infrastructure-as-code frameworks.

> **Note**
>
> The target AWS account must have Resource Explorer enabled to discover tagged resources.

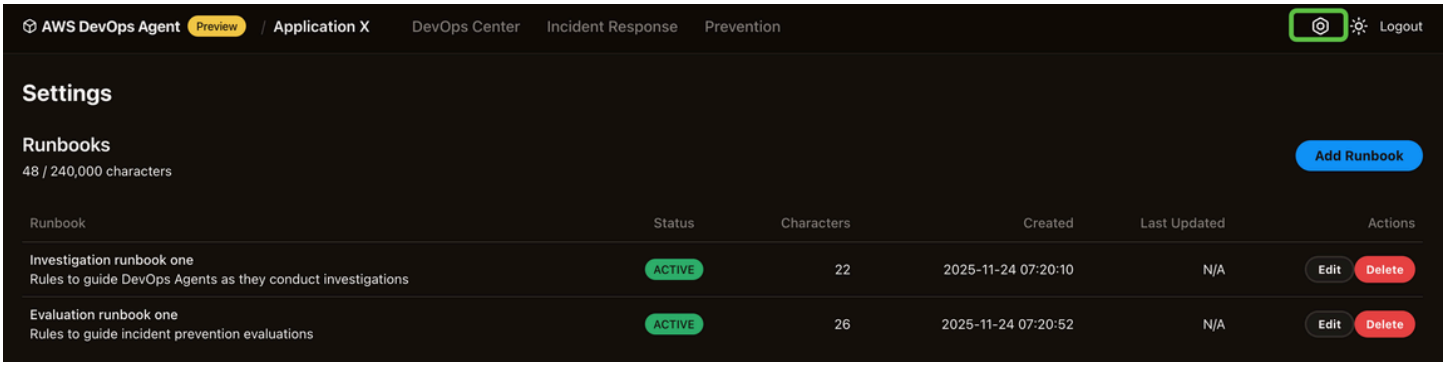
Investigation scope beyond topology

While the application topology provides important context during investigations, AWS DevOps Agent is not limited to investigating only the resources shown in the topology. The agent may use additional data sources, such as AWS service APIs or connected observability tools, to investigate resources that are not in the application topology.

To limit the resources the agent has access to, restrict the policy for the role assigned to the agent to access cross-account resources. For more information, see the section called “Limiting Agent Access in an AWS Account”.

DevOps Agent Runbooks

You can configure runbooks to guide the AWS DevOps Agent as it performs incident response investigations and incident prevention evaluations. Click on the settings icon in the top right of your DevOps Agent web app and enter one or more runbooks.



Investigation and evaluation performance can be further enhanced by using AWS DevOps Agent Runbooks to provide investigation hints and guidance for non-standard configurations across all

connection types. You can also use Investigation chat/steering to test runbook concepts by steering live investigations or restarting completed ones with additional hints.

Please note that the title, description, and status of runbooks matter. Your DevOps Agent will use titles and descriptions to decide which runbook to use. Include a summary of the runbook content in the description field and use a title that reflects the purpose of the runbook. You can also control which runbooks are available to your DevOps Agent via the status field. If you want to skip some runbooks from being used for any reason, you can modify the status of runbook by clicking the " **Edit** " button and changing its status to " **Inactive**." To make runbooks available again to your DevOps Agent, set the " **Status** " back to " **Active**."

Example use cases

If your environment uses Dynatrace for alarms and metrics, Splunk for logs, and CloudWatch for serverless logs, document this in your runbook to accelerate issue resolution.

If your environment uses Atlassian MCP server for managing knowledge, you can hint the agent to read the knowledge using runbook. Here is an example:

```
In all investigations, try to use as many documentation tools as you can to gain
 context and background information about the task at hand. Retrieving this information
 saves on investigation time.

# Atlassian Confluence Wiki Pages
Additional runbook and investigation information may be found by using the Atlassian
 MCP tools. These pages contain very helpful information on how to triage issues, and
 may contain past root causes and investigation attempts.

## Using the search functionality
In general, you can use the `atlassian_search` tool to look up keywords that may appear
 inside of a Confluence page. This tool will only return a fragment of information. To
 retrieve the entire page, you must use the corresponding get page tools.

## Using the getConfluencePage functionality
If you need to read the contents of an entire page, use the
 `atlassian_getConfluencePage` tool. This tool reads the entire page contents, instead
 of just a fragment.
```

Public preview pricing and limits

Pricing

During public preview you will not be charged for usage of AWS DevOps Agent. Note, however, that queries and API calls made to other AWS and non-AWS services may generate charges from those services. For example, if they charge for metric or log queries made by your DevOps Agent during an incident resolution investigation.

Limits and quotas

During public preview, on a per account basis, your usage of AWS DevOps Agent will be limited to:

- 10 Agent Spaces
- 20 DevOps Agent incident resolution hours
- 10 DevOps Agent incident prevention hours
- 1,000 chat messages per month

The number of concurrent incident resolution investigation tasks allowed is **three**, and the number of concurrent incident prevention evaluation tasks allowed is **one**.

Getting started with AWS DevOps Agent

In this getting started guide, you'll create a basic Agent Space, configure minimal permissions, and conduct your first AI-powered investigation.

Topics:

Creating an Agent Space

An Agent Space defines the tools and infrastructure that AWS DevOps Agent has access to.
This guide walks you through creating an Agent Space, configuring primary account access, and enabling the DevOps Agent Web App. See “What is an Agent Space” to learn more about the Agent Space concept.

Creating an Agent Space

Access the AWS DevOps Agent console

1. Sign in to the AWS Management Console
2. Navigate to the AWS DevOps Agent console

Name the Agent Space

1. Click **Create Agent Space +**

In the **Agent Space details** section, provide:

1. In the **Name** field, enter a name for your Agent Space

2. (Optional) In the **Description** field, add details about the Agent Space's purpose

Configuring primary account access

In the **Give this Agent Space AWS resource access** section, you will set up an IAM role to grant the Agent Space access to the primary AWS account. The primary account is the AWS account where you create your Agent Space. AWS DevOps Agent requires an IAM role to discover and access AWS resources in this account during investigations.

Choose a role configuration method. Select one of the following options:

Option 1: Auto-create a new AWS DevOps Agent role (recommended)

This option automatically creates a role with appropriate permissions for AWS DevOps Agent to investigate resources in your account.

> **Note**
>
> You must have IAM permissions to create new roles to use this option.

1. Select **Auto-create a new AWS DevOps Agent role**
2. (Optional) Update the Agent Space role name to be created

Option 2: Assign an existing role

Use this option when another administrator has previously created a role specifically for AWS DevOps Agent.

1. Select **Assign an existing role**
2. From the dropdown menu, select an existing role that has appropriate permissions

Option 3: Create a new AWS DevOps Agent role using a policy template

Use this option when you need to limit the services and resources the agent can access in the primary account.

1. Select **Create a new AWS DevOps Agent role using a policy template**
2. Follow the instructions to create the new role's trust policy and inline policy.

Enabling the Agent Space Web App

The Web App is where personnel interact with AWS DevOps Agent for incident investigations and reviewing recommendations. See AWS DevOps Agent Console Architecture[link] to learn more. When enabled, users can access the Agent Space Web App through an IAM authentication link from the AWS Management Console.

Select one of the following options:

Option 1: Auto-create a new AWS DevOps Agent role (recommended)

This option automatically creates a role with appropriate permissions for accessing the DevOps Agent Web App.

> **Note**
>
> You must have IAM permissions to create new roles to use this option.

1. Select **Auto-create a new AWS DevOps Agent role**
2. Review the permissions that will be granted to the role

Option 2: Assign an existing role

Use this option when another administrator has previously created an operator role.

1. Select **Assign an existing role**
2. From the dropdown menu, select an existing role that has appropriate permissions

Option 3: Create a new AWS DevOps Agent role using a policy template

Use this option when you need to customize permissions for web app access.

1. Select **Create a new AWS DevOps Agent role using a policy template**
2. Follow the instructions to create the new role's trust policy and inline policy.

Once all sections are filled out, click **Submit**

Verifying your Agent Space setup

Once configured, the “Configure Web App” button should become “Admin access”. Clicking should open the Web App and authenticate successfully.

Next steps

After setting up your Agent Space, consider these next steps:

- Add secondary accounts if your applications span multiple AWS accounts
- Configure third-party integrations like observability tools or ticketing systems
- Set up AWS Identity Center authentication for production environments
- Explore your application resource mapping to help AWS DevOps Agent understand your infrastructure

CLI onboarding guide

Overview

AWS DevOps Agent helps you monitor and manage your AWS infrastructure. This guide walks you through setting up AWS DevOps Agent in the **us-east-1** region.

> **Note**
>
> AWS DevOps Agent is in preview. The instructions on this page may change before general availability (GA)

Prerequisites

- AWS CLI installed and configured
- Authenticated to your AWS monitoring account
- AWS DevOps Agent is available in us-east-1

Setup AWS CLI for DevOps Agent

Download the Service Model

Download the AWS DevOps Agent model file:

```
# Download from: https://d1co8nkiwcta1g.cloudfront.net/devopsagent.json
# Save as: devopsagent.json
```

Patch AWS CLI

Add the DevOps Agent service to your AWS CLI:

```
aws configure add-model --service-model "file://${PWD}/devopsagent.json" --service-name
 devopsagent
```

Test the installation:

```
aws devopsagent help
```

IAM Roles Setup

1. Create DevOps Agent Space Role

Create the AWS Identity and Access Management (IAM) trust policy:

```
cat > devops-agentspace-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "aidevops.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "<ACCOUNT_ID>"
```

```
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:aidevops:us-east-1:<ACCOUNT_ID>:agentspace/*"
        }
      }
    }
  ]
}
EOF
```

Create the IAM role:

```
aws iam create-role \
  --region us-east-1 \
  --role-name DevOpsAgentRole-AgentSpace \
  --assume-role-policy-document file://devops-agentspace-trust-policy.json

# Save the role ARN
aws iam get-role --role-name DevOpsAgentRole-AgentSpace --query 'Role.Arn' --output
 text
```

Attach the AWS managed policy:

```
aws iam attach-role-policy \
  --role-name DevOpsAgentRole-AgentSpace \
  --policy-arn arn:aws:iam::aws:policy/AIOpsAssistantPolicy
```

Create and attach additional inline policy:

```
cat > devops-agentspace-inline-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAwsSupportActions",
      "Effect": "Allow",
      "Action": [
        "support:CreateCase",
        "support:DescribeCases"
      ],
      "Resource": [
        "*"
```

```
      ]
    },
    {
      "Sid": "AllowExpandedAIOpsAssistantPolicy",
      "Effect": "Allow",
      "Action": [
        "aidevops:GetKnowledgeItem",
        "aidevops:ListKnowledgeItems",
        "eks:AccessKubernetesApi",
        "synthetics:GetCanaryRuns",
        "route53:GetHealthCheckStatus",
        "resource-explorer-2:Search"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
EOF

aws iam put-role-policy \
  --role-name DevOpsAgentRole-AgentSpace \
  --policy-name AllowExpandedAIOpsAssistantPolicy \
  --policy-document file://devops-agentspace-inline-policy.json
```

2. Create Operator App IAM Role

Create the IAM trust policy:

```
cat > devops-operator-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "aidevops.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "<ACCOUNT_ID>"
```

```
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:aidevops:us-east-1:<ACCOUNT_ID>:agentspace/*"
        }
      }
    }
  ]
}
EOF
```

Create the IAM role:

```
aws iam create-role \
  --role-name DevOpsAgentRole-WebappAdmin \
  --assume-role-policy-document file://devops-operator-trust-policy.json \
  --region us-east-1

# Save the role ARN
aws iam get-role --role-name DevOpsAgentRole-WebappAdmin --query 'Role.Arn' --output
 text
```

Create and attach the operator app inline policy:

```
cat > devops-operator-inline-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowBasicOperatorActions",
      "Effect": "Allow",
      "Action": [
        "aidevops:GetAgentSpace",
        "aidevops:GetAssociation",
        "aidevops:ListAssociations",
        "aidevops:CreateBacklogTask",
        "aidevops:GetBacklogTask",
        "aidevops:UpdateBacklogTask",
        "aidevops:ListBacklogTasks",
        "aidevops:ListChildExecutions",
        "aidevops:ListJournalRecords",
        "aidevops:DiscoverTopology",
        "aidevops:InvokeAgent",
        "aidevops:ListGoals",
```

```
        "aidevops:ListRecommendations",
        "aidevops:ListExecutions",
        "aidevops:GetRecommendation",
        "aidevops:UpdateRecommendation",
        "aidevops:CreateKnowledgeItem",
        "aidevops:ListKnowledgeItems",
        "aidevops:GetKnowledgeItem",
        "aidevops:UpdateKnowledgeItem",
        "aidevops:ListPendingMessages",
        "aidevops:InitiateChatForCase",
        "aidevops:EndChatForCase",
        "aidevops:DescribeSupportLevel",
        "aidevops:SendChatMessage"
      ],
      "Resource": "arn:aws:aidevops:us-east-1:<ACCOUNT_ID>:agentspace/*"
    },
    {
      "Sid": "AllowSupportOperatorActions",
      "Effect": "Allow",
      "Action": [
        "support:DescribeCases",
        "support:InitiateChatForCase",
        "support:DescribeSupportLevel"
      ],
      "Resource": "*"
    }
  ]
}
EOF

aws iam put-role-policy \
  --role-name DevOpsAgentRole-WebappAdmin \
  --policy-name AIDevOpsBasicOperatorActionsPolicy \
  --policy-document file://devops-operator-inline-policy.json
```

Onboarding Steps

1. Create an Agent Space

```
aws devopsagent create-agent-space \
  --name "MyAgentSpace" \
  --description "AgentSpace for monitoring my application" \
  --endpoint-url "https://api.prod.cp.aidevops.us-east-1.api.aws" \
```

```
  --region us-east-1
```

Save the `agentSpaceId` from the response.

To list your agent spaces later:

```
aws devopsagent list-agent-spaces \
  --endpoint-url "https://api.prod.cp.aidevops.us-east-1.api.aws" \
  --region us-east-1
```

2. Associate AWS Account

Associate your AWS account to enable topology discovery. This is the primary source or monitoring account, the account where the agentspace exists.

```
aws devopsagent associate-service \
  --agent-space-id <AGENT_SPACE_ID> \
  --service-id aws \
  --configuration '{
    "aws": {
      "assumableRoleArn": "arn:aws:iam::<ACCOUNT_ID>:role/DevOpsAgentRole-AgentSpace",
      "accountId": "<ACCOUNT_ID>",
      "accountType": "monitor",
      "resources": [
      ]
    }
  }' \
  --endpoint-url "https://api.prod.cp.aidevops.us-east-1.api.aws" \
  --region us-east-1
```

3. Enable Operator App

Authentication flows can use IAM, IDC. Enable the Operator App for your AgentSpace:

```
aws devopsagent enable-operator-app \
  --agent-space-id <AGENT_SPACE_ID> \
  --auth-flow iam \
  --operator-app-role-arn "arn:aws:iam::<ACCOUNT_ID>:role/DevOpsAgentRole-WebappAdmin"
 \
  --endpoint-url "https://api.prod.cp.aidevops.us-east-1.api.aws" \
  --region us-east-1
```

Note

If you have previously created an Operator App role for another AgentSpace in your account, you can reuse that role ARN.

4. (Optional) Associate Additional Source Accounts

For additional accounts that AWS DevOps Agent should monitor, you need to create an IAM cross-account role.

Create Cross-Account Role in External Account

Switch to the external account and create the trust policy, the `MONITORING_ACCOUNT_ID` is the main account hosting the agentspace setup in step 2. This allows the monitoring account to assume a role in the secondary source account(s).

```
cat > devops-cross-account-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::<MONITORING_ACCOUNT_ID>:role/DevOpsAgentRole-AgentSpace"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "sts:ExternalId": "arn:aws:aidevops:us-
east-1:<MONITORING_ACCOUNT_ID>:agentspace/<AGENT_SPACE_ID>"
        }
      }
    }
  ]
}
EOF
```

Create the cross-account IAM role:

```
aws iam create-role \
```

```
  --role-name DevOpsAgentCrossAccountRole \
  --assume-role-policy-document file://devops-cross-account-trust-policy.json

# Save the role ARN
aws iam get-role --role-name DevOpsAgentCrossAccountRole --query 'Role.Arn' --output
 text
```

Attach the AWS managed policy:

```
aws iam attach-role-policy \
  --role-name DevOpsAgentCrossAccountRole \
  --policy-arn arn:aws:iam::aws:policy/AIOpsAssistantPolicy
```

Attach the additional inline policy (json created in step 2):

```
aws iam put-role-policy \
  --role-name DevOpsAgentCrossAccountRole  \
  --policy-name AIDevOpsAdditionalPermissions \
  --policy-document file://devops-agentspace-inline-policy.json
```

Update Monitoring Account Role

Switch back to your monitoring account and add cross-account permissions:

```
cat > devops-cross-account-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "arn:aws:iam::<EXTERNAL_ACCOUNT_ID>:role/DevOpsAgentCrossAccountRole"
    }
  ]
}
EOF

aws iam put-role-policy \
  --role-name DevOpsAgentRole-AgentSpace \
  --policy-name DevOpsAgentCrossAccountAccess \
  --policy-document file://devops-cross-account-policy.json
```

Associate the External Account

```
aws devopsagent associate-service \
  --agent-space-id <AGENT_SPACE_ID> \
  --service-id aws \
  --configuration '{
    "sourceAws": {
      "accountId": "<EXTERNAL_ACCOUNT_ID>",
      "accountType": "source",
      "assumableRoleArn": "arn:aws:iam::<EXTERNAL_ACCOUNT_ID>:role/
DevOpsAgentCrossAccountRole",
      "resources": []
    }
  }' \
  --endpoint-url "https://api.prod.cp.aidevops.us-east-1.api.aws" \
  --region us-east-1
```

5. (Optional) Associate GitHub

> **Note**
>
> GitHub must first be registered through the AWS DevOps Agent Console UI via OAuth flow before it can be associated via CLI.

the section called “Connecting to CI/CD pipelines”

List registered services:

```
aws devopsagent list-services \
  --endpoint-url "https://api.prod.cp.aidevops.us-east-1.api.aws" \
  --region us-east-1
```

Save the `serviceId` for serviceType:“github”

Search for accessible GitHub repositories:

```
aws devopsagent search-service-accessible-resource \
  --service-id <serviceId> \
  --endpoint-url "https://api.prod.cp.aidevops.us-east-1.api.aws" \
  --region us-east-1
```

Save the `name`, `id`, and extract the `owner` from the `fullName`. The `ownerType` will either be user or organization depending on the type of repo.

After registering GitHub in the UI, associate GitHub repositories:

```
aws devopsagent associate-service \
  --agent-space-id <AGENT_SPACE_ID> \
  --service-id github \
  --configuration '{
    "github": {
      "repoName": "<GITHUB_REPO_NAME>",
      "repoId": "<GITHUB_REPO_ID>",
      "owner": "<GITHUB_OWNER>",
      "ownerType": "organization"
    }
  }' \
  --endpoint-url "https://api.prod.cp.aidevops.us-east-1.api.aws" \
  --region us-east-1
```

6. (Optional) Register and Associate ServiceNow

First, register the ServiceNow service with OAuth credentials:

```
aws devopsagent register-service \
  --service servicenow \
  --service-details  '{
    "servicenow": {
      "instanceUrl": "<SERVICENOW_INSTANCE_URL>",
      "authorizationConfig": {
        "oAuthClientCredentials": {
            "clientName": "<SERVICENOW_CLIENT_NAME>",
            "clientId": "<SERVICENOW_CLIENT_ID>",
            "clientSecret": "<SERVICENOW_CLIENT_SECRET>"
        }
      }
    }
  }' \
  --endpoint-url "https://api.prod.cp.aidevops.us-east-1.api.aws" \
  --region us-east-1
```

Save the returned `<SERVICE_ID>`, then associate ServiceNow:

```
aws devopsagent associate-service \
```

```
  --agent-space-id <AGENT_SPACE_ID> \
  --service-id <SERVICE_ID> \
  --configuration '{
    "servicenow": {
      "instanceUrl": "<SERVICENOW_INSTANCE_URL>"
    }
  }' \
  --endpoint-url "https://api.prod.cp.aidevops.us-east-1.api.aws" \
  --region us-east-1
```

7. (Optional) Register and Associate Dynatrace

First, register the Dynatrace service with OAuth credentials:

```
aws devopsagent register-service \
  --service dynatrace \
  --service-details '{
  "dynatrace": {
    "accountUrn": "<DYNATRACE_ACCOUNT_URN>",
    "authorizationConfig": {
        "oAuthClientCredentials": {
            "clientName": "<DYNATRACE_CLIENT_NAME>",
            "clientId": "<DYNATRACE_CLIENT_ID>",
            "clientSecret": "<DYNATRACE_CLIENT_SECRET>"
        }
      }
    }
  }' \
  --endpoint-url "https://api.prod.cp.aidevops.us-east-1.api.aws" \
  --region us-east-1
```

Save the returned <SERVICE_ID>, then associate Dynatrace (resources are optional), the environment is which specific Dynatrace environment to associate with:

```
aws devopsagent associate-service \
  --agent-space-id <AGENT_SPACE_ID> \
  --service-id <SERVICE_ID> \
  --configuration '{
    "dynatrace": {
      "envId": "<DYNATRACE_ENVIRONMENT_ID>",
      "resources": [
        "<DYNATRACE_RESOURCE_1>",
```

```
        "<DYNATRACE_RESOURCE_2>"
      ]
    }
  }' \
  --endpoint-url "https://api.prod.cp.aidevops.us-east-1.api.aws" \
  --region us-east-1
```

The response will include webhook information for integration, you can trigger an investigation from Dynatrace using this webhook. For details see, the section called “Connecting Dynatrace”

7. (Optional) Register and Associate Splunk

First, register the Splunk service with OAuth credentials:

The endpoint will look something like:

`"endpoint": "[https://<XXX>.api.scs.splunk.com/<XXX>/mcp/v1/](https://partner-nfr-us-east-1.api.scs.splunk.com/partner-nfr-us-east-1/mcp/v1/)",`

```
aws devopsagent register-service \
  --service mcpserversplunk \
  --service-details '{
  "mcpserversplunk": {
    "name": "<SPLUNK_NAME>",
    "endpoint": "<SPLUNK_ENDPOINT>",
    "authorizationConfig": {
        "bearerToken": {
            "tokenName": "<SPLUNK_TOKEN_NAME>",
            "tokenValue": "<SPLUNK_TOKEN_VALUE>"
        }
      }
    }
  }' \
  --endpoint-url "https://api.prod.cp.aidevops.us-east-1.api.aws" \
  --region us-east-1
```

Save the returned <SERVICE_ID>, then associate Splunk:

```
aws devopsagent associate-service \
  --agent-space-id <AGENT_SPACE_ID> \
  --service-id <SERVICE_ID> \
  --configuration '{
```

```
    "mcpserversplunk":  {
      "name": "<SPLUNK_NAME>",
      "endpoint": "<SPLUNK_ENDPOINT>"
    }
  }' \
  --endpoint-url "https://api.prod.cp.aidevops.us-east-1.api.aws" \
  --region us-east-1
```

The response will include webhook information for integration, you can trigger an investigation from Splunk using this webhook. For details see, the section called “Connecting Splunk”

8. (Optional) Register and Associate New relic

First, register the New relic service with apiKey credentials:

Region: Either "US" or "EU"

Optional fields: `applicationIds`, `entityGuids`, `alertPolicyIds`

```
aws devopsagent register-service \
  --service mcpservernewrelic \
  --service-details '{
    "mcpservernewrelic": {
      "authorizationConfig": {
        "apiKey": {
          "apiKey": "<YOUR_NEW_RELIC_API_KEY>",
          "accountId": "<YOUR_ACCOUNT_ID>",
          "region": "US",
          "applicationIds": ["<APP_ID_1>", "<APP_ID_2>"],
          "entityGuids": ["<ENTITY_GUID_1>"],
          "alertPolicyIds": ["<POLICY_ID_1>"]
        }
      }
    }
  }' \
  --endpoint-url "https://api.prod.cp.aidevops.us-east-1.api.aws" \
  --region us-east-1
```

Save the returned `<SERVICE_ID>`, then associate New relic:

```
aws devopsagent associate-service \
  --agent-space-id <AGENT_SPACE_ID> \
```

```
  --service-id <SERVICE_ID> \
  --configuration '{
    "mcpservernewrelic":  {
      "accountId": "<YOUR_ACCOUNT_ID>",
      "endpoint": "https://mcp.newrelic.com/mcp/"
    }
  }' \
  --endpoint-url "https://api.prod.cp.aidevops.us-east-1.api.aws" \
  --region us-east-1
```

The response will include webhook information for integration, you can trigger an investigation from New relic using this webhook. For details see, the section called “Connecting New Relic”

9. (Optional) Register and Associate Datadog

Datadog must first be registered through the AWS DevOps Agent Console UI via OAuth flow before it can be associated via CLI. For details, see the section called “Connecting DataDog”

List registered services

```
aws devopsagent list-services \
  --endpoint-url "https://api.prod.cp.aidevops.us-east-1.api.aws" \
  --region us-east-1
```

Note the serviceId for serviceType:“`mcpserverdatasog`”

Save the returned <SERVICE_ID>, then associate Datadog:

```
aws devopsagent associate-service \
  --agent-space-id <AGENT_SPACE_ID> \
  --service-id <SERVICE_ID> \
  --configuration '{
    "mcpserverdatadog": {
      "name": "Datadog-MCP-Server",
      "endpoint": "<DATADOG_MCP_ENDPOINT>"
    }
  }' \
  --endpoint-url "https://api.prod.cp.aidevops.us-east-1.api.aws" \
  --region us-east-1
```

The response will include webhook information for integration, you can trigger an investigation from Datadog using this webhook. For details see, the section called “Connecting DataDog”

10. (Optional) Deleting an agent space

Deleting agent spaces

```
aws devopsagent delete-agent-space \
  --agent-space-id <AGENT_SPACE_ID> \
  --endpoint-url "https://api.prod.cp.aidevops.us-east-1.api.aws" \
  --region us-east-1
```

Verification

Verify your setup:

```
# List your AgentSpaces
aws devopsagent list-agent-spaces \
  --endpoint-url "https://api.prod.cp.aidevops.us-east-1.api.aws" \
  --region us-east-1

# Get details of a specific AgentSpace
aws devopsagent get-agent-space \
  --agent-space-id <AGENT_SPACE_ID> \
  --endpoint-url "https://api.prod.cp.aidevops.us-east-1.api.aws" \
  --region us-east-1

# List associations for an AgentSpace
aws devopsagent list-associations \
  --agent-space-id <AGENT_SPACE_ID> \
  --endpoint-url "https://api.prod.cp.aidevops.us-east-1.api.aws" \
  --region us-east-1
```

Notes

- Replace <AGENT_SPACE_ID>, <ACCOUNT_ID>, <STACK_NAME>, <TEAM_ID>, etc. with your actual values
- All commands must be run in us-east-1 region
- When onboarding accounts, we recommend providing CloudFormation stacks to expedite resource indexing
- Alternatively, you can use tag key:value pairs
- If you want to onboard all stacks in an account, leave the resources list empty

Creating a test environment

This guide provides hands-on tests to validate AWS DevOps Agent's incident response functionality using sample architecture. Use this supplement if you want to test DevOps Agent before connecting your production systems.

Prerequisites

- AWS account with administrative access
- AWS DevOps Agent Space created with and configured using the Auto create DevOps Agent role flow

Cost and safety overview

Cost protection

- **EC2 test**: FREE (AWS Free Tier) or ~$0.02 for 2 hours
- **Lambda test**: FREE (1M requests/month free tier)
- **CloudWatch**: FREE (10 alarms, basic metrics included)
- **Expected estimated total cost**: $0.00 - $0.05 for complete testing

Safety features in these tests

- **Auto-termination**: Built-in automatic shutdown
- **Free Tier eligible**: Uses smallest instance types
- **Limited scope**: Minimal, isolated test resources
- **Easy cleanup**: Simple console steps to remove everything
- **No production impact**: Completely separate test environment

Set up your AWS account for testing

> **Important**
>
> Infrastructure resources need to be deployed in the AWS account where you created your DevOps Agent Space's primary cloud account. The specific region does not matter.

1. Log into AWS Console: https://console.aws.amazon.com
2. Ensure you're working in the same AWS account where your DevOps Agent Space is located
3. You can use any region for your testing resources

> **Note**
>
> The 1:1 mapping between your DevOps Agent's primary account and the test environment resources you are creating simplifies the test setup. You can easily extend your DevOps Agent Space to include secondary accounts and enable cross-account investigations.

Choose your test

You can run either test independently or both together:

Test option A: EC2 CPU capacity test

Purpose: Validate AWS DevOps Agent's ability to detect and investigate EC2 performance issues

Estimated time: 5 minutes setup + 10 minutes automatic execution

Difficulty: Fully automated (no manual steps required)

Test option B: Lambda error rate test

Purpose: Validate AWS DevOps Agent's ability to detect and investigate Lambda function errors

Estimated time: 10 minutes setup + 2 minutes to trigger

Difficulty: Very easy

Test option A: EC2 CPU capacity test

Step 1: Deploy CloudFormation stack for EC2 test

We'll use CloudFormation to create our test resources, which allows AWS DevOps Agent to properly track and investigate them.

1. **Navigate to CloudFormation**:
   a. In AWS Console, search for "CloudFormation" and click **CloudFormation**
   b. Click **Create stack** > **With new resources (standard)**
2. **Upload template**:
   a. Create a new local file called`AWS-DevOpsAgent-ec2-test.yaml`
   b. Copy and paste this CloudFormation template into the file:
      i.

```
AWSTemplateFormatVersion: '2010-09-09'
    Description: 'AWS DevOps Agent EC2 CPU Test Stack'
    Parameters:
      MyIP:
        Type: String
        Description: Your current IP address for SSH access (find at https://
whatismyipaddress.com)
        Default: '0.0.0.0/0'
    Resources:
      # Security Group for SSH access
      TestSecurityGroup:
        Type: AWS::EC2::SecurityGroup
        Properties:
          GroupName: AWS-DevOpsAgent-test-sg
          GroupDescription: AWS DevOps Agent beta testing security group
          SecurityGroupIngress:
            - IpProtocol: tcp
              FromPort: 22
              ToPort: 22
              CidrIp: !Ref MyIP
              Description: SSH access from your IP
          Tags:
            - Key: Name
              Value: AWS-DevOpsAgent-Test-SG
            - Key: Purpose
              Value: AWS-DevOpsAgent-Testing
      # Key Pair for SSH access
```

```
      TestKeyPair:
        Type: AWS::EC2::KeyPair
        Properties:
          KeyName: AWS-DevOpsAgent-test-key
          KeyType: rsa
          Tags:
            - Key: Name
              Value: AWS-DevOpsAgent-Test-Key
            - Key: Purpose
              Value: AWS-DevOpsAgent-Testing
      # EC2 Instance for CPU testing
      TestInstance:
        Type: AWS::EC2::Instance
        Properties:
          InstanceType: t3.micro
          ImageId: '{{resolve:ssm:/aws/service/ami-amazon-linux-latest/al2023-
ami-kernel-6.1-x86_64}}'
          KeyName: !Ref TestKeyPair
          SecurityGroupIds:
            - !Ref TestSecurityGroup
          UserData:
            Fn::Base64: !Sub |
              #!/bin/bash
              yum update -y
              yum install -y htop

              # Create the CPU stress test script
              cat > /home/ec2-user/cpu-stress-test.sh << 'EOF'
              #!/bin/bash
              echo "Starting AWS DevOpsAgent CPU Stress Test"
              echo "Time: $(date)"
              echo "Instance: $(curl -s http://169.254.169.254/latest/meta-data/
instance-id)"
              echo ""

              # Get number of CPU cores
              CORES=$(nproc)
              echo "CPU Cores: $CORES"
              echo ""

              echo "Starting stress test (5 minutes)..."
              echo "This will generate >70% CPU usage to trigger CloudWatch
 alarm"
              echo ""
```

```
              # Create CPU load using yes command
              echo "Starting CPU load processes..."
              for i in $(seq 1 $CORES); do
                  (yes > /dev/null) &
                  CPU_PID=$!
                  echo "Started CPU load process $i (PID: $CPU_PID)"
                  echo $CPU_PID >> /tmp/cpu_test_pids
              done

              # Auto-cleanup after 5 minutes
              (sleep 300 && echo "Stopping CPU load processes..." && kill
 $(cat /tmp/cpu_test_pids 2>/dev/null) 2>/dev/null && rm -f /tmp/cpu_test_pids)
 &

              echo ""
              echo "CPU load processes started for 5 minutes"
              echo "Check CloudWatch for alarm trigger in 3-5 minutes"
              EOF

              chmod +x /home/ec2-user/cpu-stress-test.sh
              chown ec2-user:ec2-user /home/ec2-user/cpu-stress-test.sh

              # Create auto-shutdown script (safety mechanism)
              cat > /home/ec2-user/auto-shutdown.sh << 'SHUTDOWN_EOF'
              #!/bin/bash
              echo "Auto-shutdown scheduled for 2 hours from now: $(date)"
              sleep 7200
              echo "Auto-shutdown executing at: $(date)"
              sudo shutdown -h now
              SHUTDOWN_EOF

              chmod +x /home/ec2-user/auto-shutdown.sh
              nohup /home/ec2-user/auto-shutdown.sh > /home/ec2-user/auto-
shutdown.log 2>&1 &

              echo "AWS DevOpsAgent test setup completed at $(date)" > /home/
ec2-user/setup-complete.txt
          Tags:
            - Key: Name
              Value: AWS-DevOpsAgent-Test-Instance
            - Key: Purpose
              Value: AWS-DevOpsAgent-Testing
      # CloudWatch Alarm for CPU utilization
```

```
      CPUAlarm:
        Type: AWS::CloudWatch::Alarm
        Properties:
          AlarmName: AWS-DevOpsAgent-EC2-CPU-Test
          AlarmDescription: AWS-DevOpsAgent beta test - EC2 CPU utilization
 alarm
          MetricName: CPUUtilization
          Namespace: AWS/EC2
          Statistic: Average
          Period: 60
          EvaluationPeriods: 1
          Threshold: 70
          ComparisonOperator: GreaterThanThreshold
          Dimensions:
            - Name: InstanceId
              Value: !Ref TestInstance
          TreatMissingData: notBreaching
    Outputs:
      InstanceId:
        Description: EC2 Instance ID for testing
        Value: !Ref TestInstance

      SecurityGroupId:
        Description: Security Group ID
        Value: !Ref TestSecurityGroup

      AlarmName:
        Description: CloudWatch Alarm Name
        Value: !Ref CPUAlarm

      SSHCommand:
        Description: SSH command to connect to instance
        Value: !Sub 'ssh -i "AWS-DevOpsAgent-test-key.pem" ec2-user@
${TestInstance.PublicDnsName}'
```

c. In the CloudFormation console, select **Upload a template file**

d. Click **Choose file**

e. Select the `AWS-DevOpsAgent-ec2-test.yaml` file

f. Click **Next**

3. **Configure stack**:

   a. **Stack name**:`AWS-DevOpsAgent-EC2-Test`

b. **Parameters**:

i. **MyIP**: Leave as default `0.0.0.0/0` (you can secure this later if needed)

c. Click **Next**

4. **Configure stack options**:

a. Leave defaults, click **Next**

5. **Review and create**:

a. Check **I acknowledge that AWS CloudFormation might create IAM resources**

b. Click **Submit**

6. **Wait for completion**:

a. Stack creation takes 3-5 minutes

b. Status will change from`CREATE_IN_PROGRESS`to`CREATE_COMPLETE`

c. **Important**: Your EC2 instance is now part of a CloudFormation stack that AWS DevOpsAgent can track!

Optional: Secure SSH access (only if you plan to connect to the instance)

Skip this step if you just want to run the automated test

1. **Navigate to EC2 Security Groups**:

a. In AWS Console, go to **EC2** → **Security Groups**

b. Find`AWS-DevOpsAgent-test-sg`

2. **Update SSH rule**:

a. Select the security group → **Inbound rules** tab → **Edit inbound rules**

b. Find the SSH rule (port 22)

c. Change source from`0.0.0.0/0`to your IP:`[YOUR_IP]/32`

d. Get your IP from https://whatismyipaddress.com

e. Click **Save rules**

Step 2: Wait for automatic test execution

1. **Automatic test execution**:

- The CPU stress test will **automatically start 5 minutes** after instance launch

- No manual intervention required - just wait, the test runs completely in the background

2. **Monitor the test**:
   - Instance boots and prepares the test automatically
   - The script will run for 5 minutes and generate >70% CPU usage
   - CloudWatch alarm should trigger within 8-10 minutes total (5 min delay + 3-5 min for alarm)
3. **Optional: Manual re-run** (for additional testing):
   - Connect to your instance: EC2 console →`AWS-DevOpsAgent-Test-Instance`→ **Connect** → **Session Manager**
   - Run the stress test again:`./cpu-stress-test.sh`
   - Perfect for testing AWS DevOpsAgent's response multiple times

Test option B: Lambda error rate test

Step 1: Deploy CloudFormation stack for Lambda test

1. **Navigate to CloudFormation**:
   a. In AWS Console, go to **CloudFormation**
   b. Click **Create stack** → **With new resources (standard)**
2. **Upload template**:
   a. Create a new local file called`AWS-DevOpsAgent-lambda-test.yaml`
   b. Copy and paste this CloudFormation template into the file:
      i.

```
AWSTemplateFormatVersion: '2010-09-09'
    Description: 'AWS DevOpsAgent Lambda Error Test Stack'
    Resources:
      # IAM Role for Lambda function
      LambdaExecutionRole:
        Type: AWS::IAM::Role
        Properties:
          RoleName: AWS-DevOpsAgentLambdaTestRole
          AssumeRolePolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Effect: Allow
                Principal:
                  Service: lambda.amazonaws.com
```

```
                Action: sts:AssumeRole
          ManagedPolicyArns:
            - arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
          Tags:
            - Key: Name
              Value: AWS-DevOpsAgent-Lambda-Test-Role
            - Key: Purpose
              Value: AWS-DevOpsAgent-Testing
      # Lambda function that generates errors
      TestLambdaFunction:
        Type: AWS::Lambda::Function
        Properties:
          FunctionName: AWS-DevOpsAgent-test-lambda
          Runtime: python3.12
          Handler: index.lambda_handler
          Role: !GetAtt LambdaExecutionRole.Arn
          Code:
            ZipFile: |
              import json
              import random
              import time
              from datetime import datetime
              def lambda_handler(event, context):
                  print(f"AWS DevOpsAgent Test Lambda - {datetime.now()}")
                  print(f"Event: {json.dumps(event)}")

                  # Intentionally generate errors for testing
                  error_scenarios = [
                      "Simulated database connection timeout",
                      "Test API rate limit exceeded",
                      "Intentional validation error for AWS DevOpsAgent testing"
                  ]

                  # Always throw an error for testing purposes
                  error_message = random.choice(error_scenarios)
                  print(f"Generating test error: {error_message}")

                  # This will create a Lambda error that CloudWatch will detect
                  raise Exception(f"AWS DevOpsAgent Test Error:
{error_message}")
          Description: AWS DevOpsAgent beta test function - intentionally
generates errors
          Timeout: 30
          Tags:
```

```
            - Key: Name
              Value: AWS-DevOpsAgent-Test-Lambda
            - Key: Purpose
              Value: AWS-DevOpsAgent-Testing
      # CloudWatch Alarm for Lambda errors
      LambdaErrorAlarm:
        Type: AWS::CloudWatch::Alarm
        Properties:
          AlarmName: AWS-DevOpsAgent-Lambda-Error-Test
          AlarmDescription: AWS-DevOpsAgent beta test - Lambda error rate alarm
          MetricName: Errors
          Namespace: AWS/Lambda
          Statistic: Sum
          Period: 60
          EvaluationPeriods: 1
          Threshold: 0
          ComparisonOperator: GreaterThanThreshold
          Dimensions:
            - Name: FunctionName
              Value: !Ref TestLambdaFunction
          TreatMissingData: notBreaching
    Outputs:
      LambdaFunctionName:
        Description: Lambda Function Name for testing
        Value: !Ref TestLambdaFunction

      LambdaFunctionArn:
        Description: Lambda Function ARN
        Value: !GetAtt TestLambdaFunction.Arn

      AlarmName:
        Description: CloudWatch Alarm Name
        Value: !Ref LambdaErrorAlarm

      TestCommand:
        Description: AWS CLI command to test the function
        Value: !Sub 'aws lambda invoke --function-name ${TestLambdaFunction} --
payload "{\"test\":\"AWS DevOpsAgent validation\"}" response.json'
```

c. In the CloudFormation console, select **Upload a template file**

d. Click **Choose file**

e. Select the `AWS-DevOpsAgent-lambda-test.yaml` file

f. Click **Next**

3. **Configure stack**:
   a. **Stack name**:`AWS-DevOpsAgent-Lambda-Test`
   b. Click **Next**
4. **Configure stack options**:
   a. Leave defaults, click **Next**
5. **Review and create**:
   a. Check **I acknowledge that AWS CloudFormation might create IAM resources**
   b. Click **Submit**
6. **Wait for completion**:
   a. Stack creation takes 2-3 minutes
   b. Status will change to`CREATE_COMPLETE`

Step 2: Trigger Lambda errors

1. **Navigate to Lambda console**:
   a. Go to **AWS Lambda** console
   b. Find your function`AWS-DevOpsAgent-test-lambda`
2. **Test the function**:
   a. Click **Test** tab
   b. Click **Create new event**
   c. **Event name**:`AWS-DevOpsAgent-test-event`
   d. Use this JSON payload:
      i.
```
{
    "test": "AWS DevOpsAgent validation",
    "timestamp": "2024-01-01T00:00:00Z"
    }
```
   e. Click **Save**
3. **Generate errors**:
   a. Click **Test** button 3 times (wait 10 seconds between each)
   b. Each test will generate an intentional error
   c. **CloudWatch alarm** should trigger within 2-3 minutes

d. **AWS DevOpsAgent** should now be able to detect the alarm with an **Investigation** in the **Operator app** which you will set up next.

Validate AWS DevOps Agent detection

Step 1: Sanity check CloudWatch alarms (optional)

This step is for ensuring that the above tests are now in an alarm state.

For EC2 Test:

- In CloudWatch console, go to **Alarms**
- **Wait 3-5 minutes** after starting the stress test
- Your alarm should show **In alarm** state
- **If still "OK"**: Wait another 2-3 minutes (CloudWatch metrics can be delayed)

For Lambda Test:

- Check`AWS-DevOpsAgent-Lambda-Error-Test`alarm
- Should show **In alarm** within 2-3 minutes of running tests

Step 2: Start a AWS DevOps Agent Investigation

1. Open your **AWS DevOps Agent AgentSpace**
2. Click **Admin access**. This will open the DevOps Agent Space web app in a new window
3. Click the **Start Investigation** button on the right side of the screen
4. Complete the following form:
   a. **Investigation details:** Describe the investigation you'd like to run. Include any details you can about the investigation goals, areas to explore, or relevant information.
   b. **Investigation starting point**: Describe the information you'd like to start the investigation from. You can mention an alarm, metric, log snippet, or anything else to give DevOps Agent a starting point to work from. In this case, provide a summary of the alarms you just created.
   c. **Date and time of incident** (ISO 8601 preferred): YYYY-MM-DDTHH:MMZ
   d. **Name your investigation:** example: `Oncall_investigation_1:2025-10-27`
   e. **AWS Account ID** for the incident

f. **Region** where the incident occurred

g. **Priority** - AWS DevOpsAgent allows for two concurrent investigations. The Priority allows for you to define the order of execution of your investigations.

5. Click Investigate to launch the investigation.
6. Click on your Investigation listed in the dashboard. You will be taken to the Investigation Details screen where you can view the granular steps that DevOps Agent is taking.

Expected Results

EC2 test results:

- Detects EC2 CPU alarm
- Identifies root cause: "CPU stress testing workload"
- Shows timeline: Stress test → CPU spike → Alarm
- Provides recommendations for monitoring and scaling

Lambda test results:

- Detects Lambda error rate spike
- Identifies root cause: "Intentional test exceptions"
- Shows timeline: Function invocations → Errors → Alarm
- Provides recommendations for error handling and monitoring

Cleanup instructions

Cleanup test A (EC2 test)

Automatic cleanup

- Instance will auto-terminate after 2 hours (built into CloudFormation template)

Manual cleanup (immediate)

1. **Delete CloudFormation Stack**:
   a. Go to CloudFormation console

b. Select`AWS-DevOpsAgent-EC2-Teststack`

c. Click **Delete**

d. Confirm deletion

e. **This will automatically delete all resources**: EC2 instance, security group, key pair, and CloudWatch alarm

Cleanup test B (Lambda test)

1. **Delete CloudFormation Stack**:

   a. Go to CloudFormation console

   b. Select`AWS-DevOpsAgent-Lambda-Teststack`

   c. Click **Delete**

   d. Confirm deletion

   e. **This will automatically delete all resources**: Lambda function, IAM role, and CloudWatch alarm

Troubleshooting

Common issues

"Can't connect to EC2 instance"

- **Check Security Group**: Ensure SSH (port 22) is open to your IP
- **Check Key Permissions**: Run`chmod 400 AWS-DevOpsAgent-test-key.pem`
- **Verify Public IP**: Instance must have public IP assigned
- **Wait for Instance**: Ensure instance is in "Running" state

"Alarm not triggering"

- **Wait for Metrics**: CloudWatch metrics can take 2-5 minutes to appear
- **Check CPU Load**: SSH to instance and run`top`to verify CPU >70%
- **Verify Stress Test**: Run`ps aux | grep yes`to see if load processes are running
- **Extended Wait**: Sometimes takes up to 7-8 minutes for first alarm trigger

Test validation

Your AWS DevOp Agent testing is successful when:

Technical validation

- **Investigation Accuracy**: The results of the EC2 test should correctly indicate that the alarm was triggered due to CPU load. The result of the Lambda test should indicate that this was an intentional failure.
- **Timeline Accuracy**: Correct sequence of events shown
- **Recommendation Quality**: Actionable suggestions provided

Getting started with AWS DevOps Agent using AWS CDK

Overview

This guide shows you how to use AWS Cloud Development Kit (CDK) to create and deploy AWS DevOps Agent resources, including the agent space, IAM roles, and AWS account associations. Using CDK provides infrastructure as code benefits such as version control, repeatability, and automated deployment.

The CDK approach automates the manual steps described in the CLI onboarding guide by creating all required resources through CloudFormation.

> **Note**
>
> AWS DevOps Agent is in preview. The instructions on this page may change before general availability (GA).

Prerequisites

- AWS CLI installed and configured with appropriate credentials
- Node.js (version 18 or later)
- AWS CDK CLI installed globally: `npm install -g aws-cdk`
- AWS DevOps Agent is available in us-east-1

What gets created

The CDK stack creates the following resources using CloudFormation:

IAM Roles

- **DevOpsAgentRole-AgentSpace**: Main role for the agent space with:
- Trust policy for `aidevops.amazonaws.com` service
- `AIOpsAssistantPolicy` managed policy
- Additional inline policies for support and expanded permissions
- **DevOpsAgentRole-WebappAdmin**: Operator app role with:
- Trust policy for `aidevops.amazonaws.com` service
- Inline policies for basic operator actions and support

DevOps Agent Resources

- **Agent Space**: Created using `AWS::DevOpsAgent::AgentSpace` CloudFormation resource
- **AWS Association**: Created using `AWS::DevOpsAgent::Association` CloudFormation resource

Setup

1. Clone the sample repository

```
git clone https://github.com/aws-samples/sample-aws-devops-agent-cdk.git
cd sample-aws-devops-agent-cdk
```

2. Install dependencies

```
npm install
```

3. Bootstrap your AWS environment

If you haven't bootstrapped CDK in your AWS account and region before:

```
cdk bootstrap --region us-east-1
```

4. Review the configuration

The CDK stack is pre-configured with sensible defaults. You can modify the following in `lib/sample-aws-devops-agent-cdk-stack.ts`:

- Agent space name (default: "MyAgentSpace")
- IAM role names
- Policy configurations

Deployment

1. Build the TypeScript code

```
npm run build
```

2. Preview the changes

Review what resources will be created:

```
cdk diff --region us-east-1
```

3. Deploy the stack

```
cdk deploy --region us-east-1
```

The deployment will create all necessary resources and output important values:

- `AgentSpaceId`: The ID of the created agent space
- `AgentSpaceRoleArn`: The ARN of the agent space role
- `OperatorRoleArn`: The ARN of the operator role
- `AssociationId`: The ID of the AWS association

4. Enable the operator app

After deployment, run the provided script to enable the operator app:

```
./scripts/enable-operator-app.sh
```

This script uses the stack outputs to automatically configure the operator app with the correct role ARN and agent space ID.

Verification

Verify your setup using the AWS CLI:

```
# Get details of your AgentSpace (replace <AGENT_SPACE_ID> with the output value)
aws devopsagent get-agent-space \
  --agent-space-id <AGENT_SPACE_ID> \
  --endpoint-url "https://api.prod.cp.aidevops.us-east-1.api.aws" \
  --region us-east-1

# List associations
aws devopsagent list-associations \
  --agent-space-id <AGENT_SPACE_ID> \
  --endpoint-url "https://api.prod.cp.aidevops.us-east-1.api.aws" \
  --region us-east-1
```

Adding additional associations

After the initial deployment, you can extend your setup by adding associations for:

- Additional AWS accounts (cross-account monitoring)
- GitHub repositories
- ServiceNow instances
- Dynatrace environments
- Splunk instances
- New Relic accounts
- Datadog instances

Use the CLI commands from the CLI onboarding guide to add these associations to your CDK-created agent space.

Customization

Modifying IAM policies

To add custom permissions to the agent space role, modify the inline policy in the CDK stack:

```
agentSpaceRole.addToPolicy(new PolicyStatement({
  effect: Effect.ALLOW,
  actions: ['your-custom-action:*'],
  resources: ['*']
}));
```

Adding multiple agent spaces

To create multiple agent spaces, instantiate additional `AgentSpace` constructs in your stack:

```
const secondAgentSpace = new CfnAgentSpace(this, 'SecondAgentSpace', {
  name: 'SecondAgentSpace',
  description: 'Second agent space for different environment'
});
```

Cross-account deployment

To deploy the stack in a different account, ensure your CDK deployment role has the necessary permissions and specify the account in your CDK app:

```
new SampleAwsDevopsAgentCdkStack(app, 'SampleAwsDevopsAgentCdkStack', {
  env: {
    account: 'TARGET_ACCOUNT_ID',
    region: 'us-east-1'
  }
});
```

Troubleshooting

Common deployment issues

CloudFormation resource not found *Ensure you're deploying in the us-east-1 region* Verify your AWS CLI is configured with appropriate permissions

IAM role creation failed *Check that your deployment role has IAM permissions* Verify the trust policy conditions match your account ID

IAM propagation delays : The deployment script includes retry logic for IAM propagation. If deploying manually, wait a few minutes between role creation and usage.

Agent space creation failed *Ensure the DevOps Agent service is available in your region* Check that the IAM role was created successfully before the agent space

Updating the stack

To update your deployment with changes:

```
npm run build
cdk diff --region us-east-1
cdk deploy --region us-east-1
```

Cleanup

To remove all resources created by the stack:

```
cdk destroy --region us-east-1
```

Warning

This will permanently delete your agent space and all associated data. Ensure you have backed up any important information before proceeding.

Security considerations

- The CDK stack creates IAM roles with specific trust policies that only allow the DevOps Agent service to assume them
- All policies follow the principle of least privilege
- The agent space role includes conditions that restrict access to your specific AWS account and agent space
- Review and customize the IAM policies based on your organization's security requirements

Next steps

After successfully deploying your AWS DevOps Agent using CDK:

1. **Explore capabilities**: Learn about the full range of DevOps Agent features in the user guide
2. **Automate further**: Consider integrating the CDK deployment into your CI/CD pipelines

Additional resources

- AWS DevOps Agent User Guide
- Sample CDK repository

Getting started with AWS DevOps Agent using Terraform

AWS DevOps Agent helps you monitor and manage your AWS infrastructure using AI-powered insights. This guide shows you how to use Terraform to automate the setup and deployment of AWS DevOps Agent resources, providing Infrastructure as Code for your DevOps monitoring solution.

Overview

This Terraform configuration replicates the AWS DevOps Agent CLI onboarding setup, automating the creation of Agent Spaces, IAM roles, and account associations. Using Terraform provides several advantages:

- **Infrastructure as Code** – Version control your DevOps Agent configuration
- **Reproducible deployments** – Consistent setup across environments
- **Automated provisioning** – Reduce manual configuration errors
- **Cross-account management** – Easily manage multiple AWS accounts

Note

AWS DevOps Agent is in preview. The instructions on this page may change before general availability (GA).

Prerequisites

Before you begin, ensure you have:

- Terraform >= 1.0 installed
- AWS CLI configured with appropriate permissions
- AWS account with administrative access

- AWS DevOps Agent is only available in the `us-east-1` region

Required IAM permissions

Your AWS credentials must have permissions to create: *IAM roles and policies* DevOps Agent resources (Agent Spaces, associations) * Cross-account trust relationships

Architecture

The Terraform configuration creates the following resources:

IAM Resources

- **DevOpsAgentRole-AgentSpace** – IAM role for the Agent Space with monitoring permissions
- **DevOpsAgentRole-WebappAdmin** – IAM role for the Operator App interface
- Associated policies and trust relationships for secure access

DevOps Agent Resources

- **Agent Space** – The main container for your DevOps Agent configuration
- **AWS Account Association** – Links your AWS account for monitoring
- **Operator App** – (Optional) Enables the web-based operator interface
- **External Account Associations** – (Optional) For cross-account monitoring

Getting started

Step 1: Clone the repository

```
git clone https://github.com/aws-samples/sample-aws-devops-agent-terraform.git
cd sample-aws-devops-agent-terraform
```

Step 2: Configure variables

Copy the example variables file and customize it for your environment:

```
cp terraform.tfvars.example terraform.tfvars
```

Edit `terraform.tfvars` with your specific configuration:

```
agent_space_name = "MyCompanyAgentSpace"
agent_space_description = "DevOps monitoring for production workloads"
enable_operator_app = true
auth_flow = "iam"
# external_account_ids = ["123456789012"]  # Optional: for cross-account monitoring
```

Step 3: Deploy with automation (recommended)

Use the provided deployment script for a streamlined setup:

```
./deploy.sh
```

This script automatically: *Checks prerequisites (Terraform, AWS CLI, credentials)* Creates `terraform.tfvars` from example if needed *Initializes, validates, plans, and applies Terraform* Handles IAM propagation delays with retry logic

Step 4: Complete the setup

Run the post-deployment script to finalize configuration:

```
./post-deploy.sh
```

This script: *Configures AWS DevOps Agent CLI if needed* Optionally enables the Operator App * Provides verification commands

Manual deployment

If you prefer manual control over the deployment process:

Step 1: Initialize Terraform

```
terraform init
```

Step 2: Review the plan

```
terraform plan
```

Step 3: Apply the configuration

```
terraform apply
```

Type `yes` when prompted to confirm the deployment.

Configuration options

Input variables

Variable Description Default Required 1`aws_region` AWS region (must be us-east-1) `us-east-1` Yes 2`agent_space_name` Name for the Agent Space `MyAgentSpace` No 3`agent_space_description` Description for the Agent Space `AgentSpace for monitoring my application` No 4`enable_operator_app` Enable the operator web app TRUE No 5`auth_flow` Authentication flow (iam/idc) `iam` No 6`external_account_ids` External AWS accounts to monitor `[]` No 7`tags` Tags for all resources See variables.tf No

Output values

After deployment, Terraform provides these useful outputs:

- `agent_space_id` – The ID of your Agent Space
- `agent_space_arn` – The ARN of your Agent Space
- `devops_agentspace_role_arn` – ARN of the Agent Space IAM role
- `devops_operator_role_arn` – ARN of the Operator App IAM role
- `manual_setup_instructions` – Next steps and verification commands

Cross-account monitoring

To monitor resources across multiple AWS accounts, you need to set up cross-account roles.

Automated setup (recommended)

1. **Deploy the main infrastructure first**: `bash ./deploy.sh ./post-deploy.sh`
2. **Generate cross-account role templates**: `bash ./setup-cross-account-roles.sh`

This script extracts necessary values from your Terraform deployment and generates step-by-step commands for each external account.

1. **Add external account IDs**: Edit `terraform.tfvars` and add: `hcl external_account_ids = ["123456789012", "234567890123"]`
2. **Apply the updated configuration**: `bash terraform apply`

Manual cross-account setup

For each external AWS account you want to monitor:

1. **Create the trust policy**:

```
bash cat > trust-policy.json << EOF { "Version": "2012-10-17" ,
"Statement": [ { "Effect": "Allow", "Principal": { "AWS":
"arn:aws:iam::MONITORING_ACCOUNT_ID:role/DevOpsAgentRole-
AgentSpace" }, "Action": "sts:AssumeRole", "Condition":
{ "StringEquals": { "sts:ExternalId": "arn:aws:aidevops:us-
east-1:MONITORING_ACCOUNT_ID:agentspace/AGENT_SPACE_ID" } } } ] } EOF
```

1. **Create the cross-account role**:

```
bash aws iam create-role \ --role-name DevOpsAgentCrossAccountRole \ --
assume-role-policy-document file://trust-policy.json aws iam attach-
role-policy \ --role-name DevOpsAgentCrossAccountRole \ --policy-arn
arn:aws:iam::aws:policy/AIOpsAssistantPolicy
```

1. **Update your Terraform configuration** to include the external account ID in the `external_account_ids` variable.

Verification

After deployment, verify your setup using the AWS CLI:

List Agent Spaces

```
aws devopsagent list-agent-spaces \
  --endpoint-url "https://api.prod.cp.aidevops.us-east-1.api.aws" \
  --region us-east-1
```

Get Agent Space details

```
aws devopsagent get-agent-space \
  --agent-space-id <AGENT_SPACE_ID> \
  --endpoint-url "https://api.prod.cp.aidevops.us-east-1.api.aws" \
  --region us-east-1
```

List associations

```
aws devopsagent list-associations \
  --agent-space-id <AGENT_SPACE_ID> \
  --endpoint-url "https://api.prod.cp.aidevops.us-east-1.api.aws" \
  --region us-east-1
```

Accessing AWS DevOps Agent

After successful deployment, you can access AWS DevOps Agent through:

1. **AWS Management Console** – Visit https://console.aws.amazon.com/devopsagent/
2. **AWS CLI** – Use the AWS CLI with the DevOps Agent service model
3. **Operator App** – If enabled, access through the AWS console for interactive investigations

Troubleshooting

Common issues

Region error : Ensure you're deploying to the `us-east-1` region. AWS DevOps Agent is currently only available in this region.

Permission errors : Verify your AWS credentials have the necessary IAM permissions to create roles and policies.

Role trust issues : Check that trust policies include the correct account IDs and external IDs.

IAM propagation delays : The deployment script includes retry logic for IAM propagation. If deploying manually, wait a few minutes between role creation and usage.

Getting help

If you encounter issues:

1. Check the Terraform output for error messages
2. Verify your AWS credentials and permissions
3. Ensure you're using the correct region (`us-east-1`)
4. Review the AWS DevOps Agent documentation for service-specific requirements

Clean up

To remove all resources created by this Terraform configuration:

```
./cleanup.sh
```

Or manually:

```
terraform destroy
```

Important

This will permanently delete your Agent Space and all associated configurations. Ensure you have backups of any important data before proceeding.

Next steps

After setting up AWS DevOps Agent with Terraform:

1. **Configure integrations** – Connect your observability tools, code repositories, and CI/CD pipelines
2. **Set up notifications** – Configure Slack, ServiceNow, or other communication channels
3. **Review topology** – Examine the automatically generated application topology
4. **Test investigations** – Create test incidents to verify the agent's response capabilities

For more information about using AWS DevOps Agent, see the AWS DevOps Agent User Guide.

DevOps Agent Incident Response

All investigations that AWS DevOps Agent performs are listed in the Incident Prevention tab of your DevOps Agent web apps. Click on an investigation from the list to view an agent activity timeline, root cause analyses, and generated mitigation plans. You can also chat with your DevOps Agent to understand what it has done and steer its investigation plan. And, if you ever want to bring in an AWS support expert to help with investigation, you can do that too!

Investigation timeline

AWS DevOps Agent will summarize and share its actions and findings in the **Investigation Timeline** tab. Use this view to understand which hypotheses the agent investigated and view its findings. The following is a list of the types of updates you will find in the investigation timeline:

- **Planning:** Agent defines approach, creates plans, or sets up analysis tasks.
- **Fetching data**: Agent gathers data, discovers resources, or fetches information (for example, tool use, results, and activities).
- **Observations:** Agent analyzes data and identifies patterns or insights from signals.
- **Findings:** Agent draws conclusions from the analysis.
- **Root cause:** The main finding that explains why issues occurred.
- **User request:** User expands or modifies the investigation.
- **Update:** Agent response that shows thinking.

Root cause

Once your DevOps Agent completes its investigation and determines a likely root cause, it will publish a root cause summary under the Root Cause tab, containing an overview of the investigated incident, root cause, key observations supporting the root cause analysis, and any investigation gaps it encountered. Investigation gaps are a list of things the agent wanted to introspect during its investigation but was unable to do so because it lacked necessary connectivity or permissions. Use reported investigations gaps, which will be consolidated and displayed in the Investigation Gaps table in AWS console view of the related DevOps Agent Space.

Note

If you use chat to restart and refocus and investigation, your agent will create a new root cause version. You can switch between versions from the version picker in the Root Cause tab.

Mitigation plans

Once your DevOps Agent has determined a likely root cause it will offer to generate a mitigation plan. Click on the “Generate mitigation plan” and the agent will generate a plan to mitigate the investigated incident. Mitigation plans will include the following steps:

1. Prepare
2. Pre-Validate
3. Apply
4. Post-Validate steps.

Each step may contain one or more suggested actions, and each suggested action may contain commands that you can use to inform changes in infrastructure-as-code templates that you may use to make configuration changes. Where appropriate, your DevOps Agent will also provide “agent-ready specs” that you can use with coding agents you may use (e.g. Kiro).

Example mitigation scenarios:

- **System changes:** If an incident is caused by Amazon DynamoDB getting throttled due to high latency from inefficient use, AWS DevOps Agent may recommend rolling back the change as an immediate mitigation.
- **System changes:** If an incident is caused by Amazon SNS subscription errors due to filter policy mismatch, AWS DevOps Agent may recommend changing the filter policy as an immediate mitigation.
- **Input anomalies:** If an incident is caused by AWS Lambda throttling on notifications due to high traffic exceeding limits, AWS DevOps Agent may recommend increasing concurrency limits as an immediate mitigation.

- **Input Anomalies:** If an incident is caused by Amazon SNS message publish failures due to message size issues, AWS DevOps Agent may recommend adding validation to Amazon SNS message publishing as an immediate mitigation.
- **Resource Limits**: If an incident is caused by API throttling due to exceeded rate limits, AWS DevOps Agent may recommend raising rate/burst limits as an immediate mitigation.
- **Resource Limits:** If an incident is caused by Amazon DynamoDB throttling due to exceeded write capacity, AWS DevOps Agent may recommend increasing write capacity as an immediate mitigation.
- **Component Failures:** If an incident is caused by cold start latency due to performance degradation, AWS DevOps Agent may recommend increasing provisioned concurrency as an immediate mitigation.
- **Dependency Issues:** If an incident is caused by Amazon S3 access denied due to restrictive bucket policy, AWS DevOps Agent may recommend updating the bucket policy as an immediate mitigation.
- **Dependency Issues:** If an incident is caused by AWS SQS permission failure due to policy denies, AWS DevOps Agent may recommend restoring AWS SQS permissions as an immediate mitigation.

Chat and investigation steering

The investigation details page includes a chat component. You can use chat to ask your DevOps Agent follow up questions about the investigation. You can also steer the agent's investigation plan by asking it to restart and refocus its investigation.

Example chat scenarios:

- Steering investigation - "Actually, the errors in the application logs are a red herring. Can the investigation focus on the service logs and the faults logged instead"
- Resume Stopped Investigation - "Can we investigate this issue further? I see that the logs you checked earlier are not related to the problem"
- User defined time limits - "Only check the logs for last 2 hours, not the full day"
- User defined resource boundaries - "Check only the ECS cluster and its configuration to know if that caused the alarm"
- Recommendations - Next steps suggestion - "The metrics show a spike in errors at 3 PM"

- Context Preservation - "Check the logs again for the 2nd alarm you were looking at"
- Investigation History - "I have fixed the bug that caused the faults in the earlier investigation. Can you re-check if the faults are still happening?"

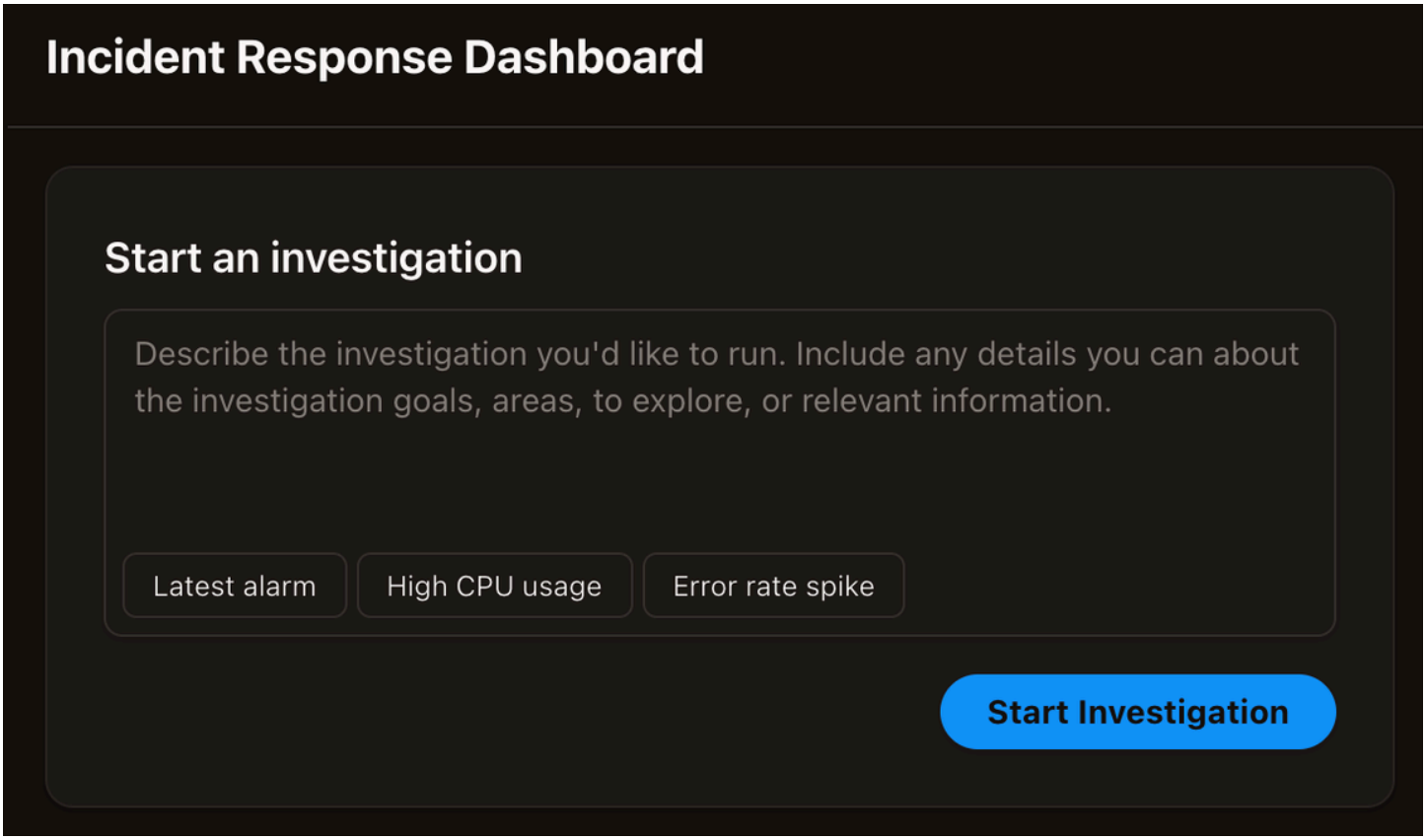
Asking for human support

If needed, you can create an AWS Support case directly from an investigation, giving AWS Support experts immediate context for faster resolution. Creating a support case from the investigation details page will automatically open a “chat with AWS support” component in your DevOps Agent Space web app.

Starting Investigations

Incident response investigations can be started in one of three ways.

- **Built-in integrations** - You can connect a DevOps Agent Space to ticketing systems like ServiceNow using built-in integrations. Once connected, DevOps Agent incident response investigations will be automatically triggered from support tickets, and your DevOps Agent will provide updates of its key findings, root cause analyses, and mitigation plans into the originating ticket.
- **Webhooks** - You can use webhooks to send events to AWS DevOps Agent. For example you can use webhooks to trigger incident response investigations from PagerDuty tickets or Grafana alarms.
- **Manually** - You can manually start incident response investigations from the Incident Response tab of the any DevOps Agent Space web app. You can either enter free form text that describes the incident you want your DevOps Agent to investigation, and it will create an investigation plan, collect finds, determine a root cause, and offer to generate a mitigation plan. You can also choose from several pre-configured starting points to quickly begin your Investigation: Latest alarm to investigate your most recent triggered alarm and analyze the underlying metrics and logs to determine the root cause, High CPU usage to investigate high CPU utilization metrics across your compute resources and identify which processes or services are consuming excessive resources, or Error rate spike to investigate the recent increase in application error rates by analyzing metrics, application logs, and identifying the source of failures.



Once you click “Start Investigation” you’ll be asked to provide some additional (optional) details to help the agent focus its work. Provide any additional details you may have, and click Start Investigation. You will then be taken to the investigation details page where you can see your DevOps Agent in action!

Ask for human support

AWS DevOps Agent can connect directly with AWS Support to streamline your incident response process. When you need additional help from AWS Support, from your DevOps Agent Space web app you can create support cases that automatically share investigation context with AWS Support engineers, reducing the time needed to explain your issue.

How it works

When investigating an incident, AWS DevOps Agent builds a comprehensive log of its analysis, including:

- Root cause investigation findings
- Metrics, logs, and traces analyzed

- Code changes and deployment history reviewed
- Remediation actions recommended
- Timeline of events and system behavior

You can escalate your investigation to AWS Support directly from the AWS DevOps Agent Space web app. When you do, AWS DevOps Agent automatically passes its investigation log to AWS Support, providing the support engineer with full context about your investigation without requiring you to manually gather and explain the details.

Chatting with AWS Support

Once you create a support case, you can communicate with AWS Support in a separate chat window within your AWS DevOps Agent Space web app. This allows you to:

- Discuss your issue with AWS Support engineers alongside your AWS DevOps Agent's investigation timeline
- View both AWS DevOps Agent's automated analysis and AWS Support's expert guidance in the same interface
- Seamlessly share additional information or clarification as needed

The chat experience keeps your AWS DevOps Agent investigation and AWS Support conversation readily accessible, enabling faster collaboration and resolution.

Support plan requirements

Your ability to create and interact with support cases through AWS DevOps Agent depends on your AWS Support plan. Please refer to the Support Plans user guide to learn more about your entitlements.

Note Basic Support customers cannot create technical support cases and therefore cannot escalate AWS DevOps Agent investigations to AWS Support **Developer Support customers** can create cases through AWS DevOps Agent, but must visit the AWS Support Center to correspond with Support engineers, as Developer Support does not include chat-based support **All other plans** can use the integrated chat experience within AWS DevOps Agent. For complete details about support plan entitlements, including response times and available case severities, see the AWS Support Plans User Guide.

What information is shared with AWS Support

When you create a support case from AWS DevOps Agent Space web app, the following information is automatically shared with AWS Support:

- **Investigation timeline**: Chronological record of AWS DevOps Agent's analysis
- **Resource information**: Affected AWS resources
- **Observability data**: Relevant metrics, logs, and traces from your integrated monitoring tools
- **Recent changes**: Code deployments, infrastructure changes, and configuration updates
- **Remediation attempts**: Actions AWS DevOps Agent recommended
- **Impact assessment**: Scope and severity of the incident

All data shared with AWS Support follows your existing AWS data residency and security configurations. AWS DevOps Agent shares only information related to your specific investigation and respects your organization's data governance policies.

Getting started

To use AWS DevOps Agent's AWS Support integration:

1. Ensure you have an active AWS Support plan.
2. Verify your AWS DevOps Agent’s IAM permissions include support case creation (support:CreateCase, support:DescribeCases).
3. When AWS DevOps Agent is investigating an issue and you need AWS Support assistance, choose **Ask for human support** from your DevOps Agent Space web app.
4. Review the investigation summary that will be shared with AWS Support.
5. Select the appropriate case severity based on your support plan entitlements.
6. Submit the case - AWS DevOps Agent automatically includes your investigation log.

The chat window opens automatically, allowing you to begin collaborating with AWS Support immediately.

Preventing future incidents

AWS DevOps Agent analyzes patterns across your incident investigations to deliver targeted recommendations that continuously improve your operational posture and prevent future incidents. Access the Prevention feature through the Prevention page in the Operator Web App.

How prevention works

AWS DevOps Agent evaluates recent incident investigations to identify lasting improvements to prevent future incidents and quicken the mean time to detection (MTTD). The agent analyzes multiple incidents to identify recommendations that may prevent whole classes of incidents in the future, focusing on the most impactful recommendations to ensure they are actionable.

The agent automatically runs evaluations weekly. You can also manually trigger an evaluation at any time, which is useful when a recent investigation warrants a quick turnaround on recommended improvements.

The agent identifies improvements across four areas:

- **Observability posture** – Recommendations to enhance monitoring, alerting, and logging capabilities to detect issues quicker and more accurately.
- **Testing gaps** – Recommendations to strengthen testing and validation processes in your deployment pipelines.
- **Code changes** – Recommendations to better improve resilience and performance.
- **Infrastructure architecture** – Recommendations to optimize resource configurations, implement autoscaling, and improve system resilience.

Benefits

- **Prevent recurring incidents** – Address root causes systematically rather than repeatedly responding to the same types of issues
- **Reduce operational toil** – Free your team from repetitive firefighting to focus on innovation and strategic improvements
- **Improve system resilience** – Strengthen your infrastructure, observability, and deployment processes based on real incident data
- **Learn from historical patterns** – Leverage insights from past incidents to make targeted improvements that have the greatest impact

Agent summary

The Agent Summary in the Prevention page of the Web App provides a description of the outcomes from the last evaluation of recent incidents. The summary explains the number of incident investigations analyzed, which incidents are similar to past ones, and which recommendations were created or updated with new information.

The summary helps you quickly understand what the agent discovered during its most recent evaluation and highlights the most notable recommendations that could have the greatest impact on your operational posture.

Recommendation categorization

The Recommendation Categorization chart shows the distribution of recommendations across four key categories:

- **Code optimization** – Recommendations to improve application code quality, performance, and error handling.
- **Observability** – Recommendations to enhance monitoring, alerting, logging, and system visibility.
- **Infrastructure** – Recommendations to optimize resource configurations, capacity tuning, and architectural resilience.
- **Governance** – Recommendations to strengthen deployment processes, testing practices, and operational controls.

This categorization helps you understand where your operational improvements are most needed and allows you to prioritize recommendations based on your team's focus areas.

Controlling evaluations

You can control when AWS DevOps Agent evaluates incidents and generates recommendations:

- **Running evaluations manually** – Click the **Run Now** button in the Prevention page to start an evaluation immediately. This is useful when a recent investigation warrants a quick turnaround on recommended improvements.
- **Stopping active evaluations** – Click the **Stop Evaluation** button in the Prevention page to halt an evaluation that is currently in progress.

Managing recommendations

AWS DevOps Agent provides recommendations in the Prevention page where you can review and manage them:

- **Viewing recommendation details** – Click on a recommendation to open the recommendation details page, where you can see more information about the suggested improvement including the incidents that informed the recommendation, the expected impacts, and next steps.
- **Accepting recommendations** – To accept a recommendation, click 'Accept' in the recommendation table. When you accept a recommendation, it remains in the recommendation table for tracking. This allows you to monitor which improvements you plan to implement and track their progress.
- **Rejecting recommendations** – To reject a recommendation, click 'Reject' in the recommendation table. When you reject a recommendation, you can provide a natural language explanation of why it doesn't meet your needs. The agent learns from this feedback and uses it to inform future recommendations, ensuring they become more aligned with your operational priorities and requirements over time.
- **Automatic removal** – Recommendations that are not accepted may be removed after approximately 6 weeks if no new incidents would have been prevented by implementing the recommendation. This ensures the Prevention page focuses on the most relevant improvements for your operational challenges.
- **Recommendation updates** – Existing recommendations are updated when newer incidents are found that would have been prevented by the recommendation. Updates may change the recommendation's priority or refine the recommendation based on new insights.

Implementing recommendations

To maximize the value of Prevention recommendations, consider the following practices for acting on them:

- **Adding recommendations to your ticket backlog** – Copy accepted recommendations to your team's ticketing system or project management tool to ensure they are prioritized alongside other engineering work.
- **Prioritizing recommendations based on impact** – Focus first on recommendations that address the most frequent or severe incident types, or those that affect critical systems.

- **Tracking implementation progress** – Monitor which recommendations have been implemented and measure their effectiveness by observing whether similar incidents decrease over time.
- **Coordinating with development teams** – Share recommendations with the appropriate teams who own the affected systems, ensuring they have the context and resources needed to implement improvements.

Configuring capabilities for AWS DevOps Agent

AWS DevOps Agent capabilities extend your agent's functionality by connecting it to your existing tools and infrastructure. Configure these capabilities to enable comprehensive incident investigation, automated response workflows, and seamless integration with your DevOps ecosystem.

The following capabilities help you maximize your DevOps Agent's effectiveness:

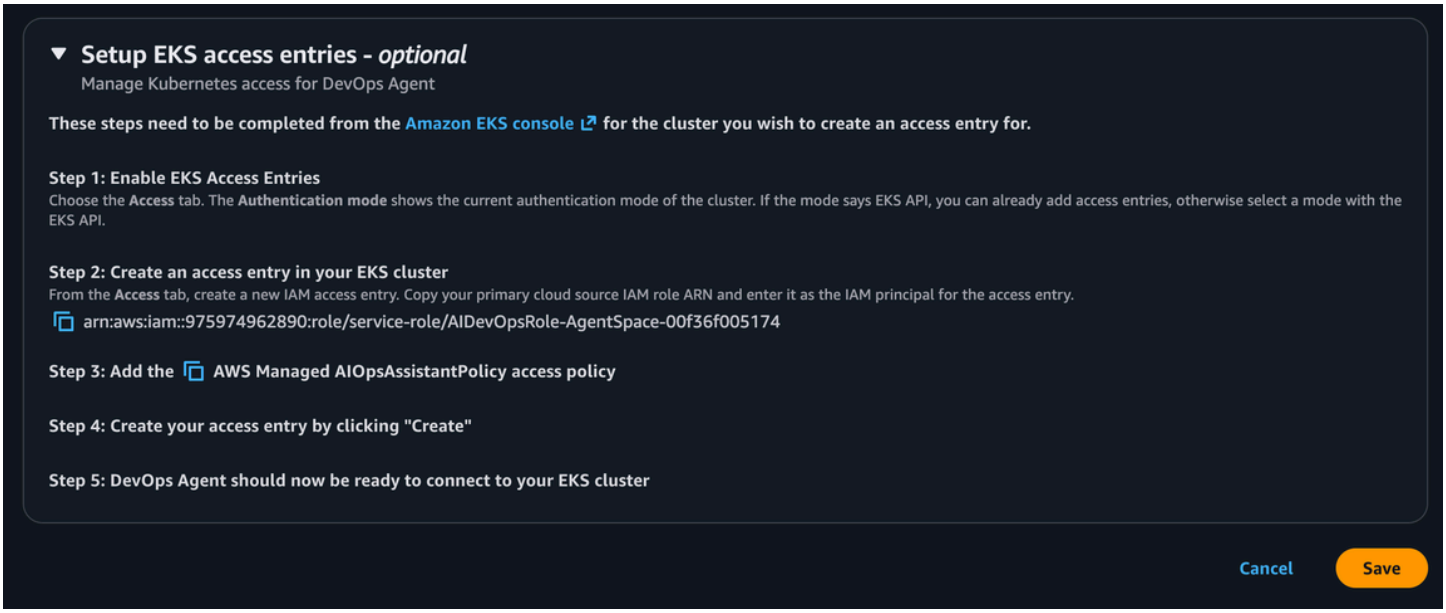
- **AWS EKS Access Setup** - Enable introspection of Kubernetes clusters, pod logs, and cluster events for both public and private EKS environments
- **CI/CD Pipeline Integration** - Connect GitHub and GitLab pipelines to correlate deployments with incidents and track code changes during investigations
- **MCP Server Connections** - Extend investigation capabilities by connecting external observability tools and custom monitoring systems through Model Context Protocol
- **Multi-Account AWS Access** - Configure secondary AWS accounts to investigate resources across your entire organization during incident response
- **Telemetry Source Integration** - Connect monitoring platforms like Datadog, New Relic, and Splunk for comprehensive observability data access
- **Ticketing and Chat Integration** - Connect ServiceNow, PagerDuty, and Slack to automate incident response workflows and enable team collaboration
- **Webhook Configuration** - Allow external systems to automatically trigger DevOps Agent investigations through HTTP requests

You can configure each capability independently based on your team's specific needs and existing tool stack. Start with the integrations most critical to your incident response workflow, then expand to additional capabilities as needed.

AWS EKS access setup

You can enable AWS DevOps Agent to describe your Kubernetes cluster objects, retrieve pod logs and cluster events, for either public or private AWS EKS clusters (only accessible with a VPC).

Within a DevOps Agent Space navigate to Capabilities > Cloud > Primary Source > Edit and follow the setup instructions.



Connecting to CI/CD pipelines

CI/CD pipeline integration enables AWS DevOps Agent to monitor deployments and correlate code changes with operational incidents during investigations. By connecting your CI/CD providers, the agent can track deployment events and associate them with AWS resources to help identify potential root causes during incident response.

AWS DevOps Agent supports integration with popular CI/CD platforms through a two-step process:

1. **Account-level registration** – Register your CI/CD provider once at the AWS account level
2. **Agent Space connection** – Connect specific projects or repositories to individual Agent Spaces based on your organizational needs

This approach allows you to share CI/CD provider registrations across multiple Agent Spaces while maintaining granular control over which projects are monitored by each space.

Supported CI/CD providers

AWS DevOps Agent supports the following CI/CD platforms:

- **GitHub** – Connect repositories from GitHub.com using the AWS DevOps Agent GitHub app.
- **GitLab** – Connect projects from GitLab.com, managed GitLab instances, or publicly accessible self-hosted GitLab deployments.

Topics

Connecting GitHub

GitHub integration enables AWS DevOps Agent to access code repositories and receive deployment events during incident investigations. This integration follows a two-step process: account-level registration of GitHub, followed by connecting specific repositories to individual Agent Spaces.

Prerequisites

Before connecting GitHub, ensure you have:

- Access to the AWS DevOps Agent admin console
- A GitHub user account or organization with admin permissions
- Authorization to install GitHub apps in your account or organization

Registering GitHub (account-level)

GitHub is registered at the AWS account level and shared among all Agent Spaces in that account. You only need to register GitHub once per AWS account.

Step 1: Navigate to pipeline providers

1. Sign in to the AWS Management Console
2. Navigate to the AWS DevOps Agent console
3. Go to the **Capabilities** tab
4. In the **Pipeline** section, click **Add**
5. Select **GitHub** from the list of available providers

If GitHub hasn't been registered yet, you'll be prompted to register it first.

Step 2: Choose connection type

On the "Register GitHub Account / Organization" screen, select whether you're connecting as a user or organization:

- **User** – Your personal GitHub account with a username and profile
- **Organization** – A shared GitHub account where multiple people can collaborate across many projects at once

Step 3: Complete GitHub OAuth flow

1. Click **Submit** to initiate the GitHub authentication flow
2. You'll be redirected to GitHub to install the AWS DevOps Agent GitHub app
3. Select which account or organization to install the app in
4. The app allows AWS DevOps Agent to receive events from connected repositories, including deployment events

Step 4: Select repositories

Choose which repositories to allow the app to access:

- **All repositories** – Grant access to all current and future repositories
- **Select repositories** – Choose specific repositories from your account or organization

Step 5: Complete installation

After selecting repositories, complete the GitHub app installation. You'll be redirected back to the AWS DevOps Agent console, where GitHub will appear as registered at the account level.

Connecting repositories to an Agent Space

After registering GitHub at the account level, you can connect specific repositories to individual Agent Spaces:

1. In the AWS DevOps Agent console, select your Agent Space
2. Go to the **Capabilities** tab
3. In the **Pipeline** section, click **Add**

4. Select **GitHub** from the list of available providers
5. Select the subset of repositories relevant to this Agent Space
6. Click **Add** to complete the connection

You can connect different sets of repositories to different Agent Spaces based on your organizational needs.

Associating AWS resources with project deployments

See the section called "Associating AWS resources with project deployments" to associate deployments with AWS resources. This helps incident investigations correlate recent deployments with possible root causes.

Understanding the GitHub app

The AWS DevOps Agent GitHub app:

- Requests read-only access to your repositories
- Receives deployment events and other repository events
- Allows AWS DevOps Agent to correlate code changes with operational incidents
- Can be uninstalled at any time through your GitHub settings

Managing GitHub connections

- **Updating repository access** – To change which repositories the GitHub app can access, go to your GitHub account or organization settings, navigate to installed GitHub apps, and modify the AWS DevOps Agent app configuration.
- **Viewing connected repositories** – In the AWS DevOps Agent console, select your Agent Space and go to the Capabilities tab to view connected repositories in the Pipeline section.
- **Removing GitHub connection** – To disconnect GitHub from an Agent Space, select the connection in the Pipeline section and click **Remove**. To uninstall the GitHub app completely, uninstall it from your GitHub account or organization settings.

Connecting GitLab

GitLab integration enables AWS DevOps Agent to monitor deployments from GitLab Pipelines to inform causal investigations during incident response. This integration follows a two-step process: account-level registration of GitLab, followed by connecting specific projects to individual Agent Spaces.

Registering GitLab (account-level)

GitLab is registered at the AWS account level and shared among all Agent Spaces in that account. Individual Agent Spaces can then choose which specific projects apply to their Agent Space.

Step 1: Navigate to pipeline providers

1. Sign in to the AWS Management Console
2. Navigate to the AWS DevOps Agent console
3. Go to the **Capabilities** tab
4. In the **Pipeline** section, click **Add**
5. Select **GitLab** from the list of available providers

If GitLab hasn't been registered yet, you'll be prompted to register it first.

Step 2: Choose connection type

On the "Register GitLab Account / Group" screen, select whether you're connecting as a person or a group:

- **Personal** – Your individual GitLab user account with a username and profile
- **Group** – In GitLab, you use groups to manage one or more related projects at the same time

Step 3: Select GitLab instance type

Choose which type of GitLab instance you're connecting to:

- **GitLab.com** (default) – The public GitLab service
- **Publicly accessible Managed GitLab instance** – A managed GitLab deployment accessible from the public internet

- **Publicly accessible self-hosted GitLab** – Your own GitLab deployment accessible from the public internet

If you're using a self-hosted or managed GitLab instance, check the box "Use GitLab self hosted endpoint" and provide the URL to your GitLab instance.

> **Note**
>
> Currently, only publicly accessible GitLab instances are supported.

Step 4: Create and provide an access token

1. In a separate browser tab, log in to your GitLab account
2. Navigate to your user settings and select **Access Tokens**
3. Create a new personal access token with the following permissions:
   - `read_repository` – Required to access repository content
   - `read_virtual_registry` – Required to access virtual registry information
   - `read_registry` – Required to access registry information
   - `api` – Required for read and write API access
   - `self_rotate` - Required for rotating tokens. This feature is currently unsupported by AWS DevOps Agent but will be supported at a later date. Adding now prevents the need to create a new token in the future.
4. Set the token expiration to a maximum of 365 days from the current date
5. Copy the generated token
6. Return to the AWS DevOps Agent console
7. Paste the token into the "Access Token" field

Step 5: Complete registration

Click **Submit** to complete the GitLab registration process. The system will validate your access token and establish the connection.

Connecting projects to an Agent Space

After registering GitLab at the account level, you can connect specific projects to individual Agent Spaces:

1. In the AWS DevOps Agent console, select your Agent Space
2. Go to the **Capabilities** tab
3. In the **Pipeline** section, click **Add**
4. Select **GitLab** from the list of available providers
5. Select the GitLab projects relevant to your Agent Space
6. Click **Save**

AWS DevOps Agent will monitor these projects for deployments from GitLab Pipelines to inform causal investigations.

Associating AWS resources with project deployments

See the section called “Associating AWS resources with project deployments” to associate deployments with AWS resources. This helps incident investigations correlate recent deployments with possible root causes.

Managing GitLab connections

- **Updating access token** – If your access token expires or needs to be updated, you can update it in the AWS DevOps Agent console by modifying the GitLab registration at the account level.
- **Viewing connected projects** – In the AWS DevOps Agent console, select your Agent Space and go to the Capabilities tab to view connected projects in the Pipeline section.
- **Removing GitLab connection** – To disconnect GitLab projects from an Agent Space, select the connection in the Pipeline section and click **Remove**. To remove the GitLab registration completely, remove it from all Agent Spaces first, then delete the registration at the account level.

Associating AWS resources with project deployments

After connecting projects to your Agent Space, you must configure the association between projects and AWS resources to enable AWS DevOps Agent to track deployments of CloudFormation templates, CDK, Elastic Container Repository images, and Terraform.

Step 1: Edit project settings

1. In the **Pipeline** section of your Agent Space, locate your connected GitLab or GitHub project in the sources list
2. Click the **Edit** button

Step 2: Associate AWS resources from primary account

Under **Associate AWS resources**, provide the corresponding resource ARNs for resources that your project deploys to:

- **CloudFormation stacks** – Enter the CloudFormation stack ARN
- **Amazon ECR repositories** – Enter the ECR repository ARN
- **AWS CDK deployments** – Enter the relevant CloudFormation stack ARNs created by CDK
- **Terraform** – Enter the S3 object ARN where your Terraform state file is stored. Currently only one Terraform state file is supported.

> ⚠ **Important**
>
> Do not include sensitive data in Terraform state files.

Click **Add new resource** to associate additional resources if needed.

Step 3: Associate resources from secondary AWS accounts

If your project deploys resources to secondary AWS accounts, provide those resource ARNs under **Associate resources from secondary AWS accounts**. Click **Add new resource** to add additional resources if needed.

Step 4: Save your changes

Click **Update Association** to save your AWS resource associations.

Following successful configuration, AWS DevOps Agent will automatically tracking deployment artifacts in GitLab Pipelines and GitHub Actions. Note that deployment artifacts that are deployed by external systems such as Jenkins or ArgoCD will not be tracked.

Connecting MCP Servers

Model Context Protocol (MCP) servers extend AWS DevOps Agent's investigation capabilities by providing access to data from your external observability tools, custom monitoring systems, and operational data sources. This guide explains how to connect an MCP server to AWS DevOps Agent.

Requirements

Before connecting an MCP server, ensure your server meets these requirements:

- **Publicly accessible endpoint** – MCP servers must be accessible from the public internet over HTTPS. AWS DevOps Agent does not support connecting to servers hosted in VPCs.
- **Streamable HTTP transport protocol** – Only MCP servers that implement the Streamable HTTP transport protocol are supported.
- **Authentication support** – Your MCP server must support OAuth 2.0 authentication flows or API key/token-based authentication.

Security considerations

When connecting MCP servers to AWS DevOps Agent, consider these security aspects:

- **Tool allowlisting –** You should allowlist only the specific tools your Agent Space needs, rather than exposing all tools from your MCP server. See the section called “Connecting MCP Servers” for how to allow list tools per Agent Space.

Please note that the maximum tool length of any MCP tool is 64.

- **Prompt injection risks** – Custom MCP servers can introduce additional risk of prompt injection attacks. See *AWS DevOps Agent Security* for more information.
- **Read-only tools and access –** Only allowlist read-only MCP tools and ensure that authentication credentials are only permitted read-only access.

See *AWS DevOps Agent Security* for more information on prompt injection and the shared responsibility model.

Registering an MCP server (account-level)

MCP servers are registered at the AWS account level and shared among all Agent Spaces in that account. Individual Agent Spaces can then choose which specific tools they need from each MCP server.

Step 1: MCP server details

1. Sign in to the AWS Management Console
2. Navigate to the AWS DevOps Agent console
3. Go to the **Capabilities** tab
4. In the **MCP Servers** section, click **Add**
5. On the **MCP server details** page, enter the following information:
   - **Name** – Enter a descriptive name for your MCP server
   - **Endpoint URL** – Enter the full HTTPS URL of your MCP server endpoint
   - **Description** (optional) – Add a description to help identify the server's purpose
   - **Enable Dynamic Client Registration** – Select this checkbox if you want to allow AWS DevOps Agent to automatically register with your MCP server's authorization server
6. Click **Next**

> **Note**
>
> The MCP server endpoint URL will be displayed in AWS CloudTrail logs in your account.

Step 2: Authorization flow

Select the authentication method for your MCP server:

OAuth Client Credentials – If your MCP server uses OAuth Client Credentials flow:

1. Select **OAuth Client Credentials**
2. Click **Next**

OAuth 3LO (Three-Legged OAuth) – If your MCP server uses OAuth 3LO for authentication:

1. Select **OAuth 3LO**
2. Click **Next**

API Key – If your MCP server uses API key authentication:

1. Select **API Key**
2. Click **Next**

Step 3: Authorization configuration

Configure additional authorization parameters based on the selected authentication method:

For OAuth Client Credentials:

1. **Client ID** – Enter the client ID of the OAuth client
2. **Client Secret** – Enter the client secret of the OAuth client
3. **Exchange URL** – Enter the OAuth token exchange endpoint URL
4. **Exchange Parameters** – Enter OAuth token exchange parameters for authenticating with the service
5. **Add Scope** – Add OAuth scopes for authentication
6. Click **Next**

For OAuth 3LO:

1. **Client ID** – Enter the client ID of the OAuth client
2. **Client Secret** – Enter the client secret of the OAuth client if it's required by your OAuth client
3. **Exchange URL** – Enter the OAuth token exchange endpoint URL
4. **Authorization URL** - Enter the OAuth authorization endpoint URL
5. **Code Challenge Support** - Select this checkbox if your OAuth client supports code challenge
6. **Add Scope** – Add OAuth scopes for authentication
7. Click **Next**

For API Key:

1. Enter an API key name
2. Enter the the name of the header that will contain the API key in the request
3. Enter your API key value
4. Click **Next**

Step 4: Review and submit

1. Review all the MCP server configuration details
2. Click **Submit** to complete the registration
3. AWS DevOps Agent will validate the connection to your MCP server
4. Upon successful validation, your MCP server will be registered at the account level

Configuring MCP tools in an Agent Space

After registering an MCP server at the account level, you can configure which tools from that server are available to specific Agent Spaces:

1. In the AWS DevOps Agent console, select your Agent Space
2. Go to the **Capabilities** tab
3. In the **MCP Servers** section, click **Add**
4. Select the registered MCP server you want to connect to this Agent Space
5. Configure which tools from this MCP server should be available to the Agent Space:
   - **Allow all tools** – Makes all tools from the MCP server available
   - **Select specific tools** – Allows you to choose which tools to allowlist
6. Click **Add** to connect the MCP server to your Agent Space

AWS DevOps Agent will now be able to use the allowlisted tools from your MCP server during investigations in this Agent Space.

Managing MCP server connections

Updating authentication credentials – If your authentication credentials need to be updated, you will need to re-register your MCP server. Navigate to the **Settings** page the AWS DevOps Agent console, locate your MCP server, remove any active associations, and click **Deregister**. Next,

register your MCP server with the new authentication credentials and re-create any necessary associations with your Agent Space.

Viewing connected MCP servers – To see all MCP servers connected to your Agent Space, select your Agent Space, go to the **Capabilities** tab, and check the **MCP Servers** section. You can also update selected tools here.

Removing MCP server connections – To disconnect an MCP server from an Agent Space, select the server in the **MCP Servers** section and click **Remove**. To completely delete an MCP server registration, remove it from all Agent Spaces first, then delete the account-level registration.

Related topics

- Security in AWS DevOps Agent
- Setting up an Agent Space
- Prompt Injection Protection

Connecting multiple AWS Accounts

Secondary AWS accounts allow AWS DevOps Agent to investigate resources across multiple AWS accounts in your organization. When your applications span multiple accounts, adding secondary accounts ensures the agent has visibility into all relevant resources during incident investigations. Greater access to the accounts and resources composing an application ensures greater investigation accuracy.

Prerequisites

Before adding a secondary AWS account, ensure you have:

- Access to the AWS DevOps Agent console in the primary account
- Administrative access to the secondary AWS account
- IAM permissions to create roles in the secondary account

Adding a secondary AWS account

In addition to the steps below, you can use the the section called “CLI onboarding guide” to programmatically add secondary accounts.

Step 1: Start the secondary account configuration

1. Sign in to the AWS Management Console and navigate to the AWS DevOps Agent console
2. Select your Agent Space
3. Go to the **Capabilities** tab
4. In the **Cloud** section, locate the **Secondary sources** subsection
5. Click **Add**

Step 2: Specify the role name

1. In the **Name your role** field, enter a name for the role you'll create in the secondary account
2. Note this name—you'll use it again when creating the role in the secondary account
3. Copy the trust policy provided in the console and save it in a scratch space

Step 3: Create the role in the secondary account

1. Open a new browser tab and sign in to the IAM console in the secondary AWS account
2. Navigate to **IAM >Roles** > **Create role**
3. Select **Custom trust policy**
4. Paste the trust policy you copied from Step 2
5. Click **Next**

Step 4: Attach the AWS managed policy

1. In the **Permissions policies** section, search for **AIOpsAssistantPolicy**
2. Select the checkbox next to the **AIOpsAssistantPolicy** managed policy
3. Click **Next**

Step 5: Name and create the role

1. In the **Role name** field, enter the same role name you provided in Step 2
2. (Optional) Add a description to help identify the role's purpose
3. Review the trust policy and attached permissions

4. Click **Create role**

Step 6: Attach the inline policy

1. In the IAM console, locate and select the role you just created
2. Go to the **Permissions** tab
3. Click **Add permissions** > **Create inline policy**
4. Switch to the **JSON** tab
5. Paste the policy you saved in Step 2
6. Paste the policy into the JSON editor in the IAM console
7. Click **Next**
8. Provide a name for the inline policy (for example, "DevOpsAgentInlinePolicy")
9. Click **Create policy**

Step 7: Complete the configuration

1. Return to the AWS DevOps Agent console in the primary account
2. Click **Next** to complete the secondary account configuration
3. Verify the connection status shows as **Active**

Understanding the required policies

AWS DevOps Agent requires three policy components to access resources in a secondary account:

- **Trust policy** – Allows AWS DevOps Agent in the primary account to assume the role in the secondary account. This establishes the trust relationship between accounts.
- **AIOpsAssistantPolicy (AWS managed policy)** – Provides the core read-only permissions AWS DevOps Agent needs to investigate resources in the secondary account. This policy is maintained by AWS and updated as new capabilities are added.
- **Inline policy** – Provides additional permissions specific to your Agent Space configuration. This policy is generated based on your Agent Space settings and may include permissions for specific integrations or features.

In the primary account, the AWS DevOps Agent IAM Role must be able to assume the role created in the secondary account.

Managing secondary accounts

- **Viewing connected accounts** – In the **Capabilities** tab, the **Secondary sources** subsection lists all connected secondary accounts with their connection status.
- **Updating the IAM role** – If you need to modify permissions, update the inline policy attached to the role in the secondary account. Changes take effect immediately.
- **Removing a secondary account** – To disconnect a secondary account, select it in the **Secondary sources** list and click **Remove**. This does not delete the IAM role in the secondary account.

Connecting telemetry sources

AWS DevOps Agent provides three ways to connect to your telemetry sources.

Built-in, 2-way integration

Currently, AWS DevOps Agent supports Dynatrace users with a built-in, 2-way integration enabling the following:

- **Topology resource mapping** - AWS DevOps Agent will augment your DevOps Agent Space Topology with entities and relationships available to it via a AWS DevOps Agent-hosted Dynatrace MCP server.
- **Automated Investigation triggering** - Dynatrace Workflows can be configured to trigger incident resolution Investigations from Dynatrace Problems.
- **Telemetry introspection** - AWS DevOps Agent can introspect Dynatrace telemetry as it investigates an issue via the AWS DevOps Agent-hosted Dynatrace MCP server.
- **Status updates** - AWS DevOps Agent will publish key investigation findings, root cause analyses, and generated mitigation plans to the Dynatrace user interface.

To learn about 2-way integrations, see

- the section called “Connecting Dynatrace”

Built-in, 1-way integration

Currently, AWS DevOps Agent supports AWS CloudWatch, Datadog, New Relic, and Splunk users with built-in, 1 way integrations.

The AWS CloudWatch built-in, 1-way integration requires no additional setup and enables the following:

- **Topology resource mapping** - AWS DevOps Agent will augment your DevOps Agent Space Topology with entities and relationships available to it via your configured primary and secondary AWS cloud accounts.
- **Telemetry introspection** - AWS DevOps Agent can introspect AWS CloudWatch telemetry as it investigates an issue via the IAM role(s) provided during primary and secondary AWS cloud account configuration.

The Datadog, New Relic, and Splunk built-in, 1 way integrations require setup and enable the following:

- **Automated Investigation triggering -** Datadog, New Relic, and Splunk events can be configured to trigger AWS DevOps Agent incident resolution Investigations via AWS DevOps Agent webhooks.
- **Telemetry introspection** - AWS DevOps Agent can introspect Datadog, New Relic, and Splunk telemetry as it investigates an issue via the each providers remote MCP server.

To learn about 1-way integrations, see the following:

- the section called “Connecting DataDog”
- the section called “Connecting New Relic”
- the section called “Connecting Splunk”

Bring-your-own telemetry sources

For any other telemetry source, including Grafana dashboards/alarms and Prometheus metrics, you can leverage AWS DevOps Agent’s support for both webhook and MCP server integration.

To learn about bring-your-own integrations, see the following

- the section called “Invoking DevOps Agent through Webhook”
- the section called “Connecting MCP Servers”

Connecting Dynatrace

Built-in, 2-way integration

Currently, AWS DevOps Agent supports Dynatrace users with a built-in, 2-way integration enabling the following:

- **Topology resource mapping** - AWS DevOps Agent will augment your DevOps Agent Space Topology with entities and relationships available to it from your Dynatrace environment.
- **Automated Investigation triggering** - Dynatrace Workflows can be configured to trigger incident resolution Investigations from Dynatrace Problems.
- **Telemetry introspection** - AWS DevOps Agent can introspect Dynatrace telemetry as it investigates an issue via the AWS DevOps Agent-hosted Dynatrace MCP server.
- **Status updates** - AWS DevOps Agent will publish key investigation findings, root cause analyses, and generated mitigation plans to the Dynatrace user interface.

Onboarding

Onboarding Process

Onboarding your Dynatrace observability system involves three stages:

1. **Connect** - Establish connection to Dynatrace by configuring account access credentials, with all the environments you may need
2. **Enable** - Activate Dynatrace in specific Agent spaces with specific Dynatrace environments
3. **Configure your Dynatrace environment -** download the workflows and dashboard and import into Dynatrace, making a note of the webhooks details to trigger investigations in designated Agent spaces

Step 1: Connect

Establish connection to your Dynatrace environment

Configuration

1. Open the hamburger menu and select Settings
2. Scroll to the Available - Telemetry section. Press Register next to Dynatrace
3. **Create OAuth client in Dynatrace, with the detailed permissions.**
   a. See Dynatrace documentation
   b. When ready press next
   c. You can connect multiple dynatrace environments and later scope to specific ones for each DevOps Agent Space you may have.
4. Enter your Dynatrace details from the OAuth client setup:
   a. **Client Name**
   b. **Client ID**
   c. **Client Secret**
   d. **Account URN**
5. Click Next
6. Review and add

Step 2: Enable

Activate Dynatrace in a specific Agent space and configure appropriate scoping

Configuration

1. From the agent spaces page, select an agent space and press view details
2. Select the Capabilities tab
3. Locate the Telemetry section, Press Add
4. You will notice Dynatrace with 'Registered' status. Click on add to add this to your agent space
5. Dynatrace Environment ID - Provide the Dynatrace environment ID you would like to associate with this DevOps agent space.
6. Enter one or more Dynatrace Entity IDs - these help DevOps agent discover your most important resources, examples might be services or applications. **If you are unsure you can press remove.**
7. Review and press Save
8. Copy the Webhook URL and Webhook Secret and follow the instructions [https://docs.dynatrace.com/docs/shortlink/aws-devops-agent] to add this credentials to Dynatrace.

Step 3: Configure your Dynatrace environment

To complete your Dynatrace set up you will need to perform certain setup steps in your Dynatrace environment. Follow the instructions here: https://docs.dynatrace.com/docs/shortlink/aws-devops-agent

Removal

The telemetry source is connected at two levels at the agent space level and at account level. To completely remove it you must first remove from all agentspaces where it is used and then it can be unregistered.

Step 1: Remove from agent space

1. From the agent spaces page, select an agent space and press view details
2. Select the Capabilities tab
3. Scroll down to the Telemetry section
4. Select Dynatrace
5. Press remove

Step 2: Remove from agent space

1. Open the hamburger menu and select Settings
2. Scroll to the **Currently registered** section.
3. Check the agent space count is zero (if not repeat Step 1 above in your other agent spaces)
4. Press Deregister next to Dynatrace

Connecting DataDog

Built-in, 1 way integration

Currently, AWS DevOps Agent supports Datadog users with built-in, 1 way integration, enabling the following:

- **Automated Investigation triggering -** Datadog events can be configured to trigger AWS DevOps Agent incident resolution Investigations via AWS DevOps Agent webhooks.

- **Telemetry introspection** - AWS DevOps Agent can introspect Datadog telemetry as it investigates an issue via the each providers remote MCP server.

Onboarding

Step 1: Connect

Establish connection to your Datadog remote MCP endpoint with account access credentials

Configuration

1. Open the hamburger menu and select Settings
2. Scroll to the Available - Telemetry section. Press Register next to Datadog
3. Enter your Datadog MCP server details:
   - **Server Name** - Unique identifier (e.g., my-datadog-server)
   - **Endpoint URL** - Your Datadog MCP server endpoint (typically https://mcp.datadoghq.com/api/unstable/mcp-server/mcp)
   - **Description** - Optional server description
4. Click Next
5. Review and submit

Authorization

Complete OAuth authorization by:

- Authorizing as your user on the Datadog OAuth page
- If not logged in, click Allow, login, then authorize

Once configured, Datadog becomes available across all Agent spaces.

Step 2: Enable

Activate DataDog in a specific Agent space and configure appropriate scoping

Configuration

1. From the agent spaces page, select an agent space and press view details (if you have not yet created an agent space see the section called “Creating an Agent Space”)

2. Select the Capabilities tab
3. Scroll down to the Telemetry section
4. Press Add
5. Select Datadog
6. Next
7. Review and press Save
8. Copy the Webhook URL and API Key

Step 3: Configure webhooks

Using the Webhook URL and API Key you can configure Datadog to send events to trigger an investigation, for example from an alarm.

To ensure that events sent can be used by the DevOps Agent, make sure that the data transmitted to the webhook matches the data schema specified below. Events that do not match this schema may be ignored by DevOps Agent.

Set the method and the headers

```
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      "Authorization": "Bearer <Token>",
    },
```

Send the body as a JSON string.

```
{
    eventType: 'incident';
    incidentId: string;
    action: 'created' | 'updated' | 'closed' | 'resolved';
    priority: "CRITICAL" | "HIGH" | "MEDIUM" | "LOW" | "MINIMAL";
    title: string;
    description?: string;
    timestamp?: string;
    service?: string;
    // The original event generated by service is attached here.
    data?: object;
```

```
}
```

Send webhooks with Datadog https://docs.datadoghq.com/integrations/webhooks/ (note select no authorization and instead use the custom header option).

Learn more: Datadog Remote MCP Server

Removal

The telemetry source is connected at two levels at the agent space level and at account level. To completely remove it you must first remove from all agentspaces where it is used and then it can be unregistered.

Step 1: Remove from agent space

1. From the agent spaces page, select an agent space and press view details
2. Select the Capabilities tab
3. Scroll down to the Telemetry section
4. Select Datadog
5. Press remove

Step 2: Remove from agent space

1. Open the hamburger menu and select Settings
2. Scroll to the **Currently registered** section.
3. Check the agent space count is zero (if not repeat Step 1 above in your other agent spaces)
4. Press Deregister next to Datadog

Connecting New Relic

Built-in, 1 way integration

Currently, AWS DevOps Agent supports New Relic users with built-in, 1 way integration, enabling the following:

- **Automated Investigation triggering -** New Relic events can be configured to trigger AWS DevOps Agent incident resolution Investigations via AWS DevOps Agent webhooks.

- **Telemetry introspection** - AWS DevOps Agent can introspect New Relic telemetry as it investigates an issue via the each providers remote MCP server.

Onboarding

Step 1: Connect

Establish connection to your New Relic remote MCP endpoint with account access credentials

Please use a Full Platform User (not Basic/Core) in New relic to enable New Relic MCP tools.

Configuration

1. Open the hamburger menu and select Settings
2. Scroll to the Available - Telemetry section. Press Register next to New Relic
3. Follow the instructions to obtain your New Relic API Key
4. Enter your New Relic MCP server API Key details:
   - **Account ID:** - Enter your New Relic account ID obtained above
   - **API Key:** Enter the API Key obtained above
   - **Select US or EU region** based on where your New Relic account is.
5. Click Add

Step 2: Enable

Activate New Relic in a specific Agent space and configure appropriate scoping

Configuration

1. From the agent spaces page, select an agent space and press view details (if you have not yet created an agent space see the section called “Creating an Agent Space”)
2. Select the Capabilities tab
3. Scroll down to the Telemetry section
4. Press Add
5. Select New Relic
6. Next
7. Review and press Save

8. Copy the Webhook URL and API Key

Step 3: Configure webhooks

Using the Webhook URL and API Key you can configure New Relic to send events to trigger an investigation, for example from an alarm.

To ensure that events sent can be used by the DevOps Agent, make sure that the data transmitted to the webhook matches the data schema specified below. Events that do not match this schema may be ignored by DevOps Agent.

Set the method and the headers

```
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      "Authorization": "Bearer <Token>",
    },
```

Send the body as a JSON string.

```
{
    eventType: 'incident';
    incidentId: string;
    action: 'created' | 'updated' | 'closed' | 'resolved';
    priority: "CRITICAL" | "HIGH" | "MEDIUM" | "LOW" | "MINIMAL";
    title: string;
    description?: string;
    timestamp?: string;
    service?: string;
    // The original event generated by service is attached here.
    data?: object;
}
```

Send webhooks with New Relic https://newrelic.com/instant-observability/webhook-notifications (note select no authorization and instead use the custom header option)

Learn more: https://docs.newrelic.com/docs/agentic-ai/mcp/overview/

Removal

The telemetry source is connected at two levels at the agent space level and at account level. To completely remove it you must first remove from all agentspaces where it is used and then it can be unregistered.

Step 1: Remove from agent space

1. From the agent spaces page, select an agent space and press view details
2. Select the Capabilities tab
3. Scroll down to the Telemetry section
4. Select New Relic
5. Press remove

Step 2: Remove from agent space

1. Open the hamburger menu and select Settings
2. Scroll to the **Currently registered** section.
3. Check the agent space count is zero (if not repeat Step 1 above in your other agent spaces)
4. Press Deregister next to New Relic

Connecting Splunk

Built-in, 1 way integration

Currently, AWS DevOps Agent supports Splunk users with built-in, 1 way integration, enabling the following:

- **Automated Investigation triggering -** Splunk events can be configured to trigger AWS DevOps Agent incident resolution Investigations via AWS DevOps Agent webhooks.
- **Telemetry introspection** - AWS DevOps Agent can introspect Splunk telemetry as it investigates an issue via the each providers remote MCP server.

Prerequisites

Getting a Splunk API token

You will need an MCP URL and token to connect Splunk.

Splunk Administrator steps

Your Splunk Administrator needs to perform the following steps:

- enable REST API access
- enable token authentication on the deployment.
- create a new role 'mcp_user', the new role does not need to have any capabilities.
- assign the role 'mcp_user' to any users on the deployment who are authorized to use the MCP server.
- create the token for the authorized users with audience as 'mcp' and set the appropriate expiration, if the user does not have the permission to create tokens themselves.

Splunk User steps

A Splunk user needs to perform the following steps:

- Get an appropriate token from the Splunk Administrator or create one themselves, if they have the permission. The audience for the token must be 'mcp'.

Onboarding

Step 1: Connect

Establish connection to your Splunk remote MCP endpoint with account access credentials

Configuration

1. Open the hamburger menu and select Settings
2. Scroll to the Available - Telemetry section. Press Register next to Splunk
3. Enter your Splunk MCP server details:
   - **Server Name** - Unique identifier (e.g., my-splunk-server)
   - **Endpoint URL** - Your Splunk MCP server endpoint:

`https://<YOUR_SPLUNK_DEPLOYMENT_NAME>.api.scs.splunk.com/<YOUR_SPLUNK_DEPLOYMENT_NAME>/mcp/v1/`

- **Description** - Optional server description
- **Token Name** - The name of the bearer token for authentication: `my-splunk-token`
- **Token Value** The bearer token value for authentication

Step 2: Enable

Activate Splunk in a specific Agent space and configure appropriate scoping

Configuration

1. From the agent spaces page, select an agent space and press view details (if you have not yet created an agent space see the section called “Creating an Agent Space”)
2. Select the Capabilities tab
3. Scroll down to the Telemetry section
4. Press Add
5. Select Splunk
6. Next
7. Review and press Save
8. Copy the Webhook URL and API Key

Step 3: Configure webhooks

Using the Webhook URL and API Key you can configure Splunk to send events to trigger an investigation, for example from an alarm.

To ensure that events sent can be used by the DevOps Agent, make sure that the data transmitted to the webhook matches the data schema specified below. Events that do not match this schema may be ignored by DevOps Agent.

Set the method and the headers

```
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      "Authorization": "Bearer <Token>",
```

```
    },
```

Send the body as a JSON string.

```
{
    eventType: 'incident';
    incidentId: string;
    action: 'created' | 'updated' | 'closed' | 'resolved';
    priority: "CRITICAL" | "HIGH" | "MEDIUM" | "LOW" | "MINIMAL";
    title: string;
    description?: string;
    timestamp?: string;
    service?: string;
    // The original event generated by service is attached here.
    data?: object;
}
```

Send webhooks with Splunk https://help.splunk.com/en/splunk-enterprise/alert-and-respond/alerting-manual/9.4/configure-alert-actions/use-a-webhook-alert-action (note select no authorization and instead use the custom header option)

Learn more:

- Splunk's MCP Server Documentation: https://help.splunk.com/en/splunk-cloud-platform/mcp-server-for-splunk-platform/about-mcp-server-for-splunk-platform
- Access requirements and limitations for the Splunk Cloud Platform REST API: https://docs.splunk.com/Documentation/SplunkCloud/latest/RESTTUT/RESTandCloud
- Manage authentication tokens in Splunk Cloud Platform: https://help.splunk.com/en/splunk-cloud-platform/administer/manage-users-and-security/9.3.2411/authenticate-into-the-splunk-platform-with-tokens/manage-or-delete-authentication-tokens
- Create and manage roles with Splunk Web: https://docs.splunk.com/Documentation/SplunkCloud/latest/Security/Addandeditroles

Removal

The telemetry source is connected at two levels at the agent space level and at account level. To completely remove it you must first remove from all agentspaces where it is used and then it can be unregistered.

Step 1: Remove from agent space

1. From the agent spaces page, select an agent space and press view details
2. Select the Capabilities tab
3. Scroll down to the Telemetry section
4. Select Splunk
5. Press remove

Step 2: Remove from agent space

1. Open the hamburger menu and select Settings
2. Scroll to the **Currently registered** section.
3. Check the agent space count is zero (if not repeat Step 1 above in your other agent spaces)
4. Press Deregister next to Splunk

Connecting to ticketing and chat

AWS DevOps Agent is designed to act as a member of your team by participating in your team's existing communication channels. You can connect DevOps Agent to your ticketing and alarming systems, like ServiceNow and PagerDuty, to automatically launch investigations from incident tickets, accelerating incident response within your existing workflows to reduce meant time to recover (MTTR). You can also connect your DevOps Agent to your team collaboration systems like Slack to receive activity summaries from your DevOps Agent in a chat channel.

Connecting ServiceNow

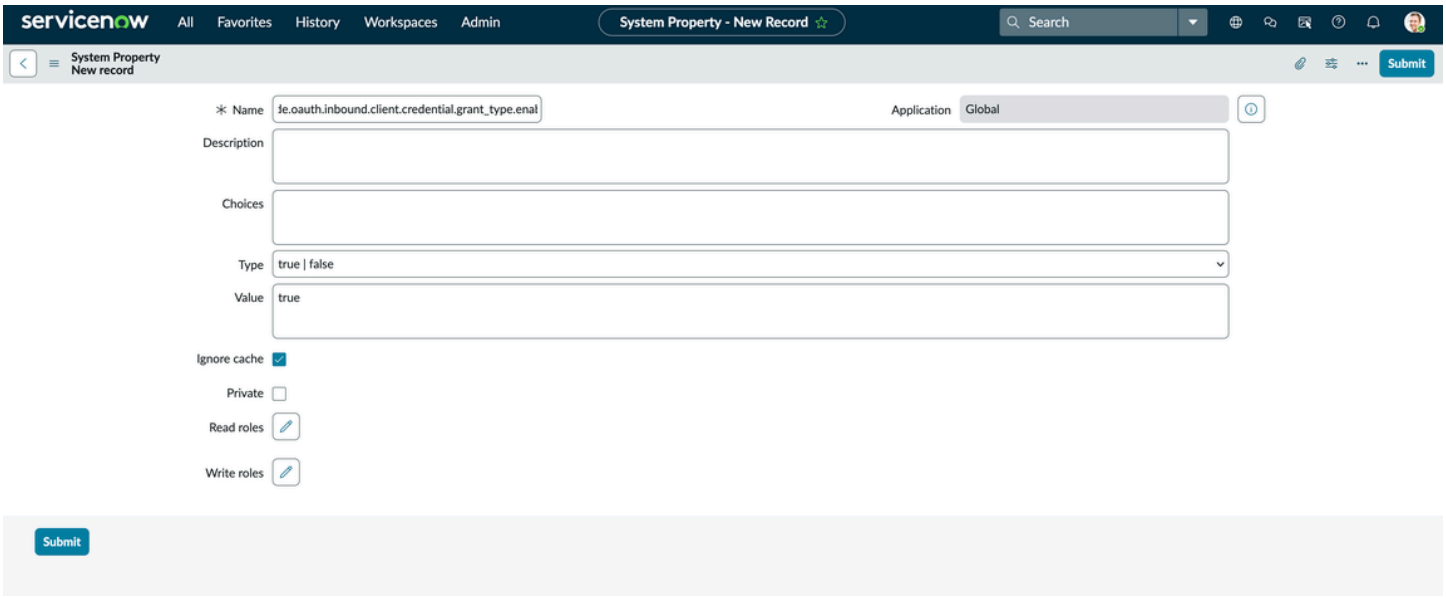
This tutorial walks you through connecting a ServiceNow instance to AWS DevOps Agent to enable it to automatically initiate incident response investigations when a ticket is created and post its key findings into the originating ticket. It also contains examples for how to configure your ServiceNow instance to send only specific tickets to a DevOps Agent Space and how to orchestrate ticket routing across multiple DevOps Agent Spaces.

Initial Setup

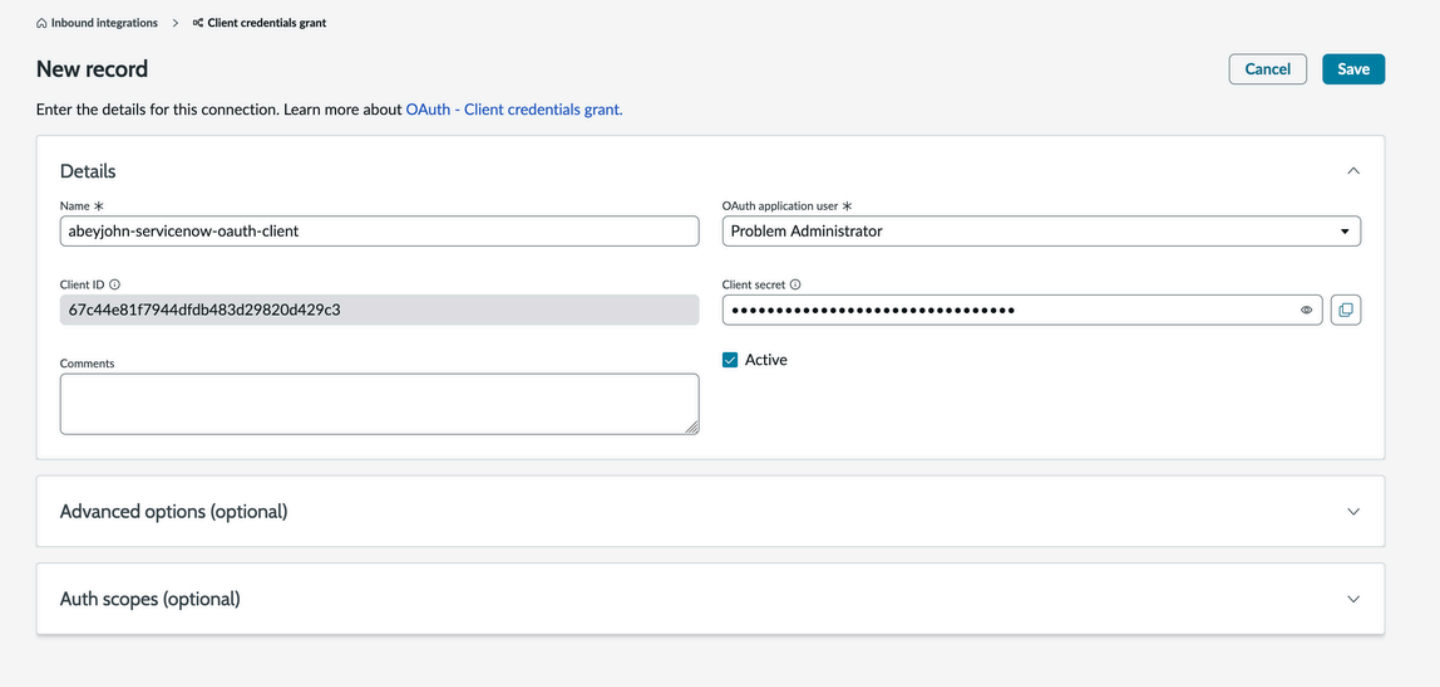
The first step is to create in ServiceNow an OAuth application client that AWS DevOps can use to access your ServiceNow instance.

Create a ServiceNow OAuth application client

1. Enable your instance's client credential system property
   a. Search `sys_properties.list` in the filter search box and then hit enter (it will not show the option but hitting enter works)
   b. Choose New
   c. Add the name as `glide.oauth.inbound.client.credential.grant_type.enabled` and the value to true with type as true | false

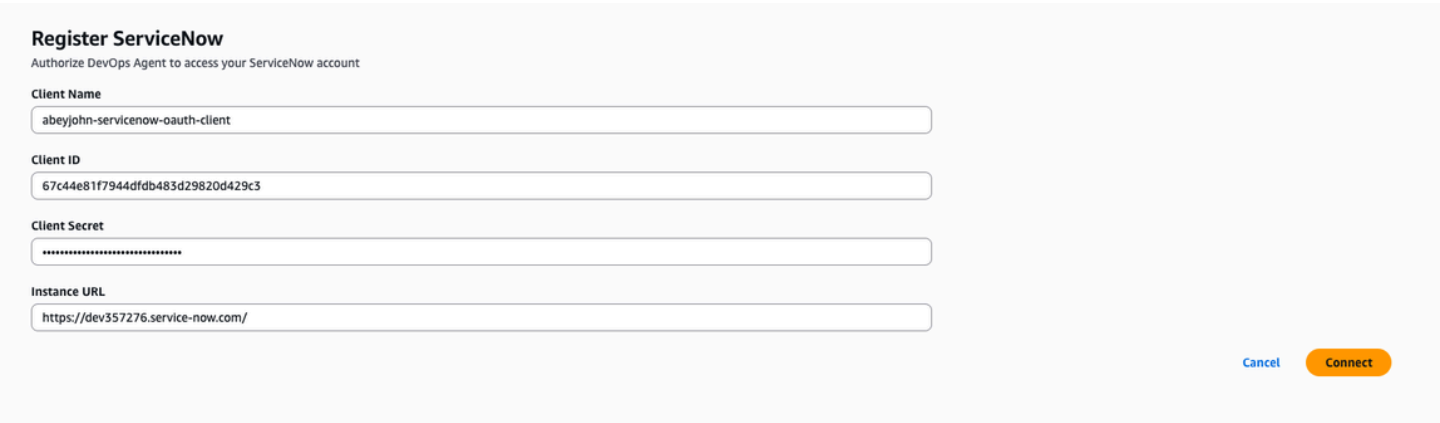


1. Navigate to System OAuth > Application Registry from the filter search box
2. Choose "New" > "New Inbound Integration Experience" > "New Integration" > "OAuth - Client Credentials Grant"
3. Pick a name and set the OAuth application user to "Problem Administrator", click "Save"



Connect your ServiceNow OAuth client to AWS DevOps Agent

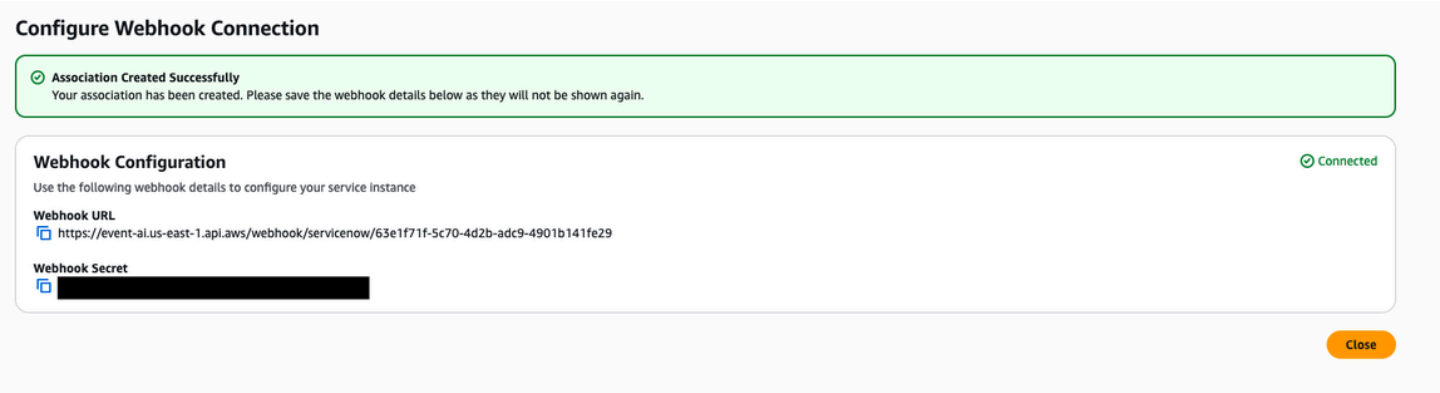
1. You can start this process in two places. First, by navigating to AWS DevOps Agent → Settings → Communications, selecting ServiceNow from the list, and clicking register. Alternatively you can select any DevOps Agent Space you may have created and navigate to Capabilities → Communications → Add → ServiceNow and click Register.
2. Next, authorize DevOps Agent to access your ServiceNow instance using the OAuth application client you just created.



- Follow the next steps, and save the resulting information about the webhook

Important

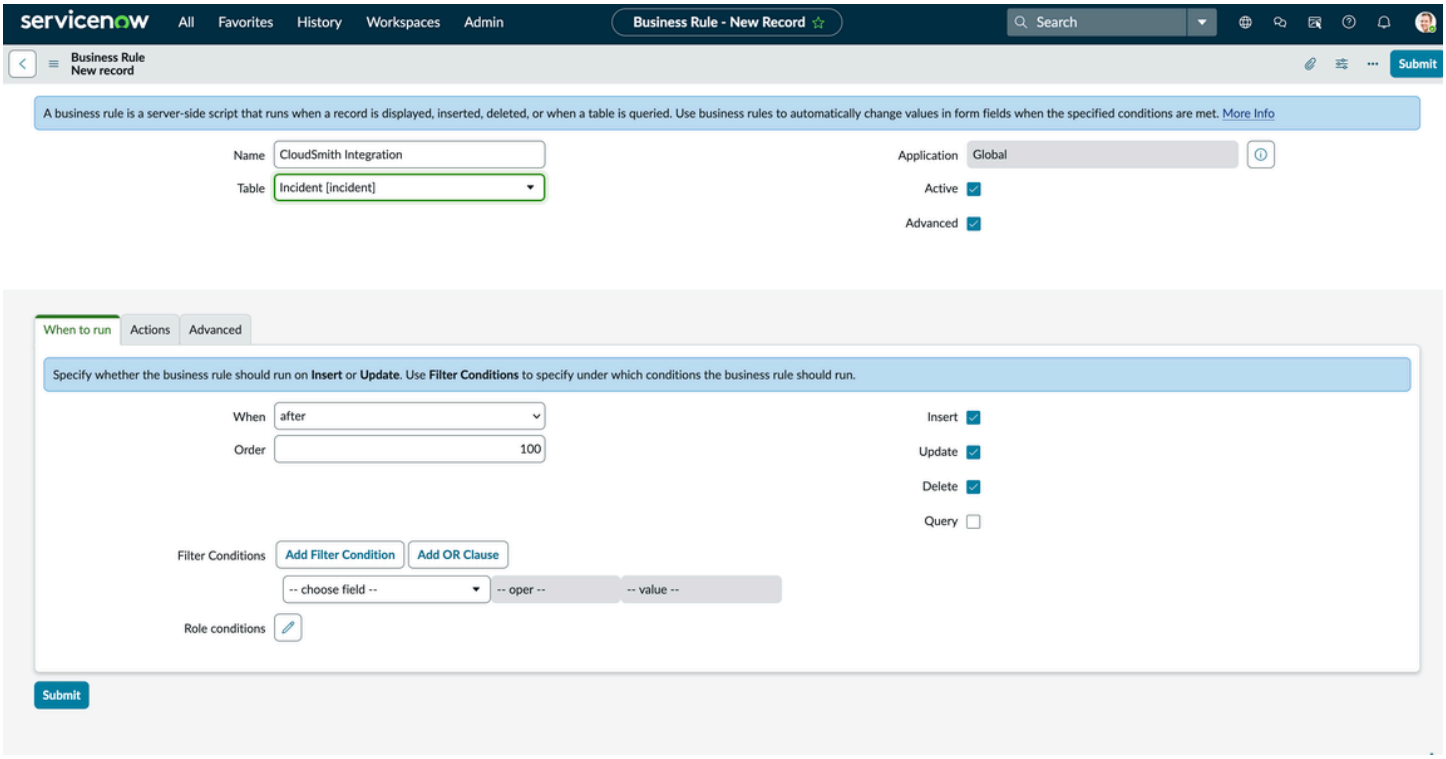
You will not see this information again



Configure your ServiceNow Business Rule

Once you have established connectivity, you'll need to configure a business rule in ServiceNow to send tickets to your DevOps Agent Space(s).

1. Navigate to Activity Subscriptions → Administration → Business Rules, and click New.
2. Set the "Table" field to "Incident [incident]", check the "Advanced" box, and set the rule to run after Insert, Update, and Delete.



1. Navigate to the “Advanced” tab and add the following webhook script, inserting your webhook secret and URL where indicated, and click Submit.

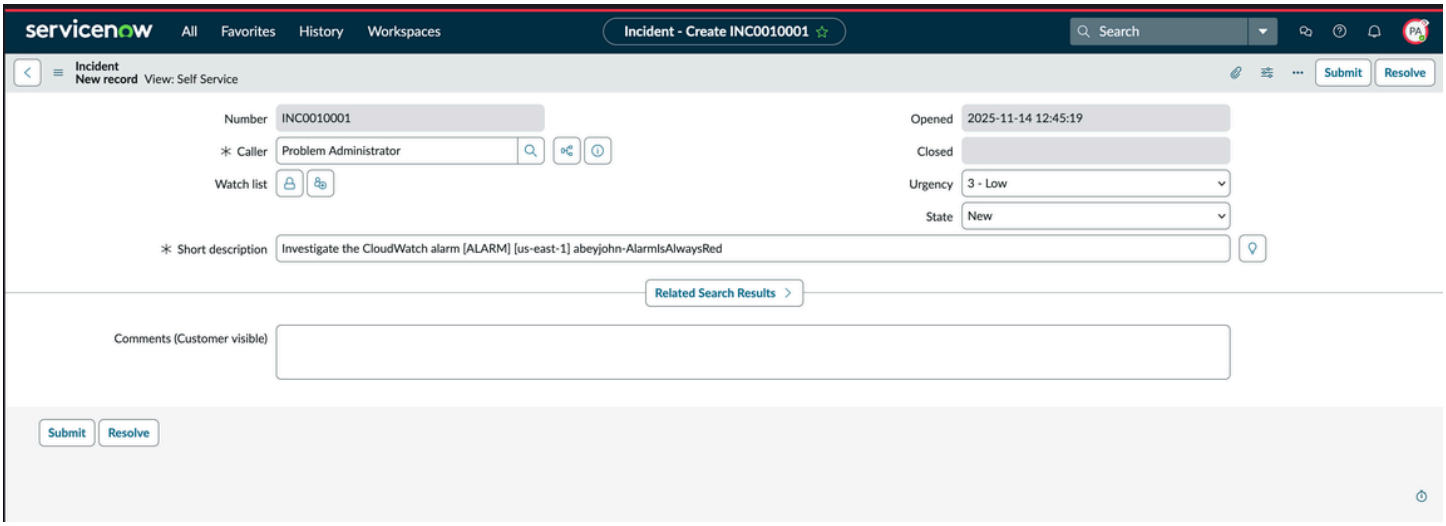
```
(function executeRule(current, previous /*null when async*/ ) {

    var WEBHOOK_CONFIG = {
        webhookSecret: GlideStringUtil.base64Encode('<<< INSERT WEBHOOK SECRET HERE
 >>>'),
        webhookUrl: '<<< INSERT WEBHOOK URL HERE >>>'
    };

    function generateHMACSignature(payloadString, secret) {
        try {
            var mac = new GlideCertificateEncryption();
            var signature = mac.generateMac(secret, "HmacSHA256", payloadString);
            return signature;
        } catch (e) {
            gs.error('HMAC generation failed: ' + e);
            return null;
        }
    }
```

```
    function callWebhook(payload, config) {
        try {
            var timestamp = new Date().toISOString();
            var payloadString = JSON.stringify(payload);
            var payloadWithTimestamp =`${timestamp}:${payloadString}`;

            var signature = generateHMACSignature(payloadWithTimestamp,
config.webhookSecret);

            if (!signature) {
                gs.error('Failed to generate signature');
                return false;
            }

            gs.info('Generated signature: ' + signature);

            var request = new sn_ws.RESTMessageV2();
            request.setEndpoint(config.webhookUrl);
            request.setHttpMethod('POST');

            request.setRequestHeader('Content-Type', 'application/json');
            request.setRequestHeader('x-amzn-event-signature', signature);
            request.setRequestHeader('x-amzn-event-timestamp', timestamp);

            request.setRequestBody(payloadString);

            var response = request.execute();
            var httpStatus = response.getStatusCode();
            var responseBody = response.getBody();

            if (httpStatus >= 200 && httpStatus < 300) {
                gs.info('Webhook sent successfully. Status: ' + httpStatus);
                return true;
            } else {
                gs.error('Webhook failed. Status: ' + httpStatus + ', Response: ' +
responseBody);
                return false;
            }

        } catch (ex) {
            gs.error('Error sending webhook: ' + ex.getMessage());
            return false;
        }
    }
```

```
    function createReference(field) {
        if (!field || field.nil()) {
            return null;
        }

        return {
            link: field.getLink(true),
            value: field.toString()
        };
    }

    function getStringValue(field) {
        if (!field || field.nil()) {
            return null;
        }
        return field.toString();
    }

    function getIntValue(field) {
        if (!field || field.nil()) {
            return null;
        }
        var val = parseInt(field.toString());
        return isNaN(val) ? null : val;
    }

    var eventType = (current.operation() == 'insert') ? "create" : "update";

    var incidentEvent = {
        eventType: eventType.toString(),
        sysId: current.sys_id.toString(),
        priority: getStringValue(current.priority),
        impact: getStringValue(current.impact),
        active: getStringValue(current.active),
        urgency: getStringValue(current.urgency),
        description: getStringValue(current.description),
        shortDescription: getStringValue(current.short_description),
        parent: getStringValue(current.parent),
        incidentState: getStringValue(current.incident_state),
        severity: getStringValue(current.severity),
        problem: createReference(current.problem),
        additionalContext: {}
    };
```

```
    incidentEvent.additionalContext = {
        number: current.number.toString(),
        opened_at: getStringValue(current.opened_at),
        opened_by: current.opened_by.nil() ? null :
 current.opened_by.getDisplayValue(),
        assigned_to: current.assigned_to.nil() ? null :
 current.assigned_to.getDisplayValue(),
        category: getStringValue(current.category),
        subcategory: getStringValue(current.subcategory),
        knowledge: getStringValue(current.knowledge),
        made_sla: getStringValue(current.made_sla),
        major_incident: getStringValue(current.major_incident)
    };

    for (var key in incidentEvent.additionalContext) {
        if (incidentEvent.additionalContext[key] === null) {
            delete incidentEvent.additionalContext[key];
        }
    }

    gs.info(JSON.stringify(incidentEvent, null, 2)); // Pretty print for logging only

    if (WEBHOOK_CONFIG.webhookUrl && WEBHOOK_CONFIG.webhookSecret) {
        callWebhook(incidentEvent, WEBHOOK_CONFIG);
    } else {
        gs.info('Webhook not configured.');
    }

})(current, previous);
```

If you chose to register your ServiceNow connection from the AWS DevOps Agent Setting tab, you now need to navigate to the DevOps Agent Space you want to investigate ServiceNow incident tickets, select Capabilities → Communications and then register the ServiceNow instance you connected to in the Settings tab. Now, everything should be set up, and all incidents where the caller is set to “Problem Administrator” (to mimic the permissions you gave the AWS DevOps OAuth client) will trigger a incident response investigation in the configured DevOps Agent Space. You can test this by creating a new incident in ServiceNow and setting the Caller field of the incident as “Problem Administrator.”



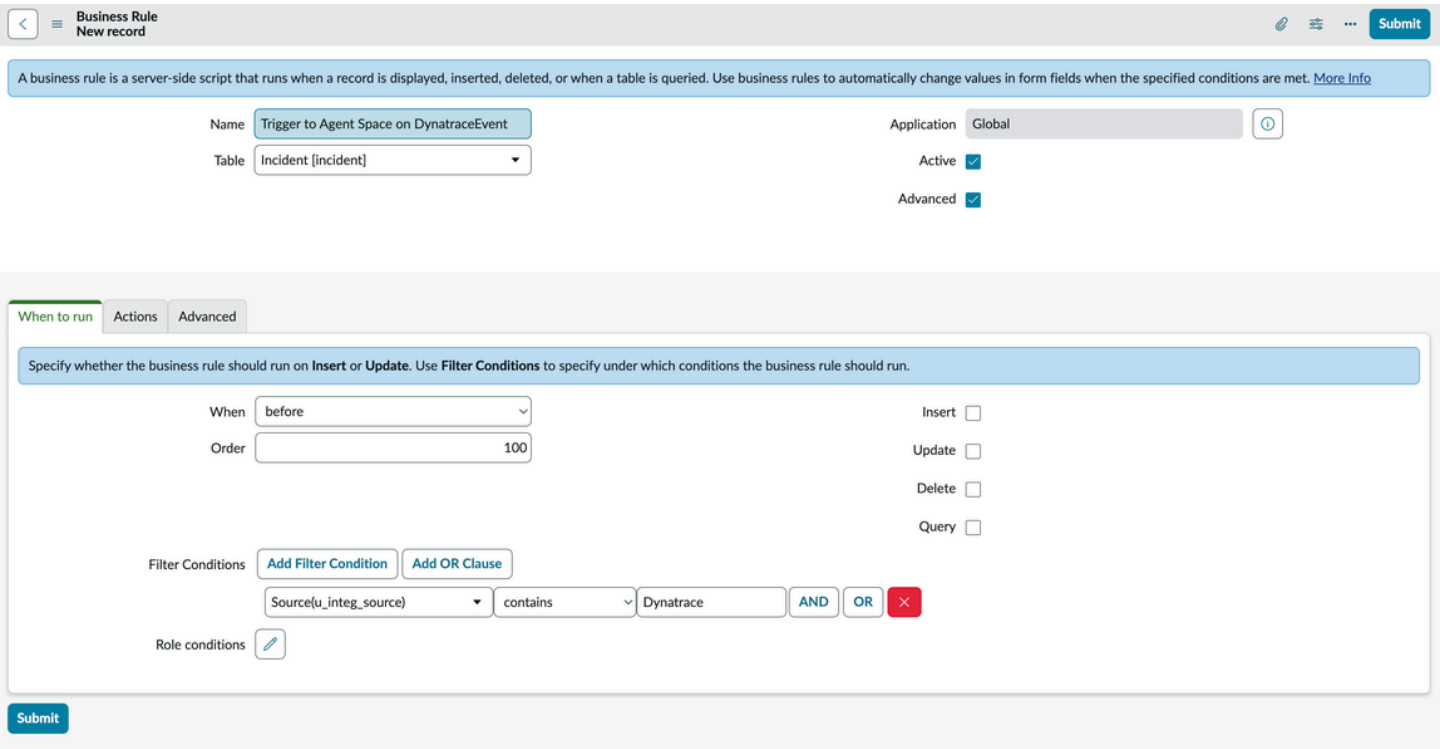
ServiceNow ticket updates

During all triggered incident response Investigations, your DevOps Agent will provide updates of its key findings, root cause analyses, and mitigation plans into the originating ticket. The agent findings are posted to the comments of an incident, and we'll currently only post agent records of type `finding`, `cause`, `investigation_summary`, `mitigation_summary` , and investigation status updates (e.g `AWS DevOps Agent started/finished its investigation`).

Ticket routing and orchestration examples

Scenario: Filtering which incidents are sent to a DevOps Agent Space

This is a simple scenario but needs some configuration in ServiceNow to create a field in ServiceNow to track incident source. For the purpose of this example, create a new Source (u_source) field using the SNOW form builder. This will enable tracking the incident source and use it to route requests from a particular source to a DevOps Agent Space. Routing is accomplished by creating a Service Now Business Rule and in the When to run tab setting “When” triggers and “Filter Conditions.” In this example the filter conditions are set as follows:

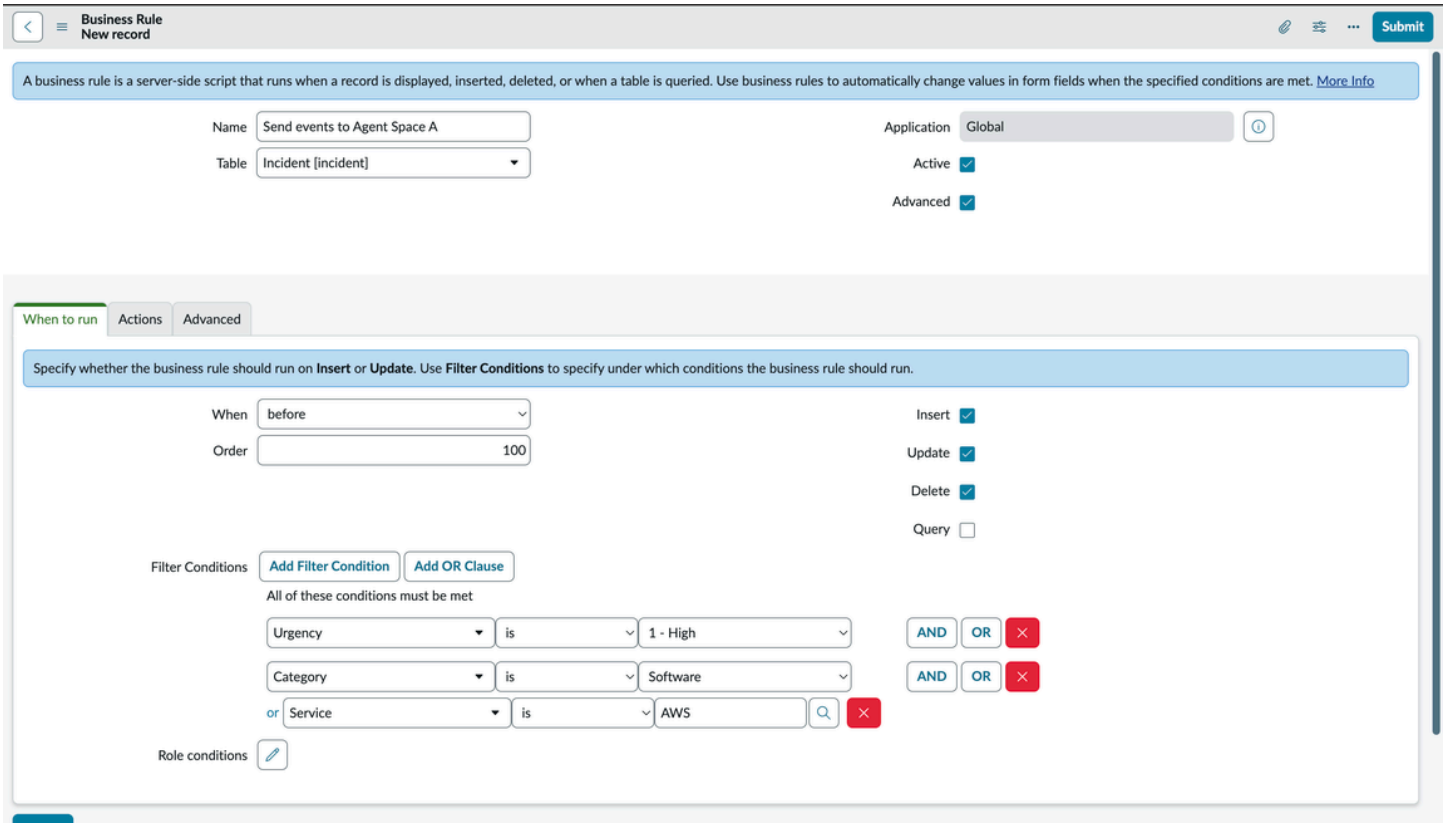


Scenario: Routing incidents across multiple DevOps Agent Spaces

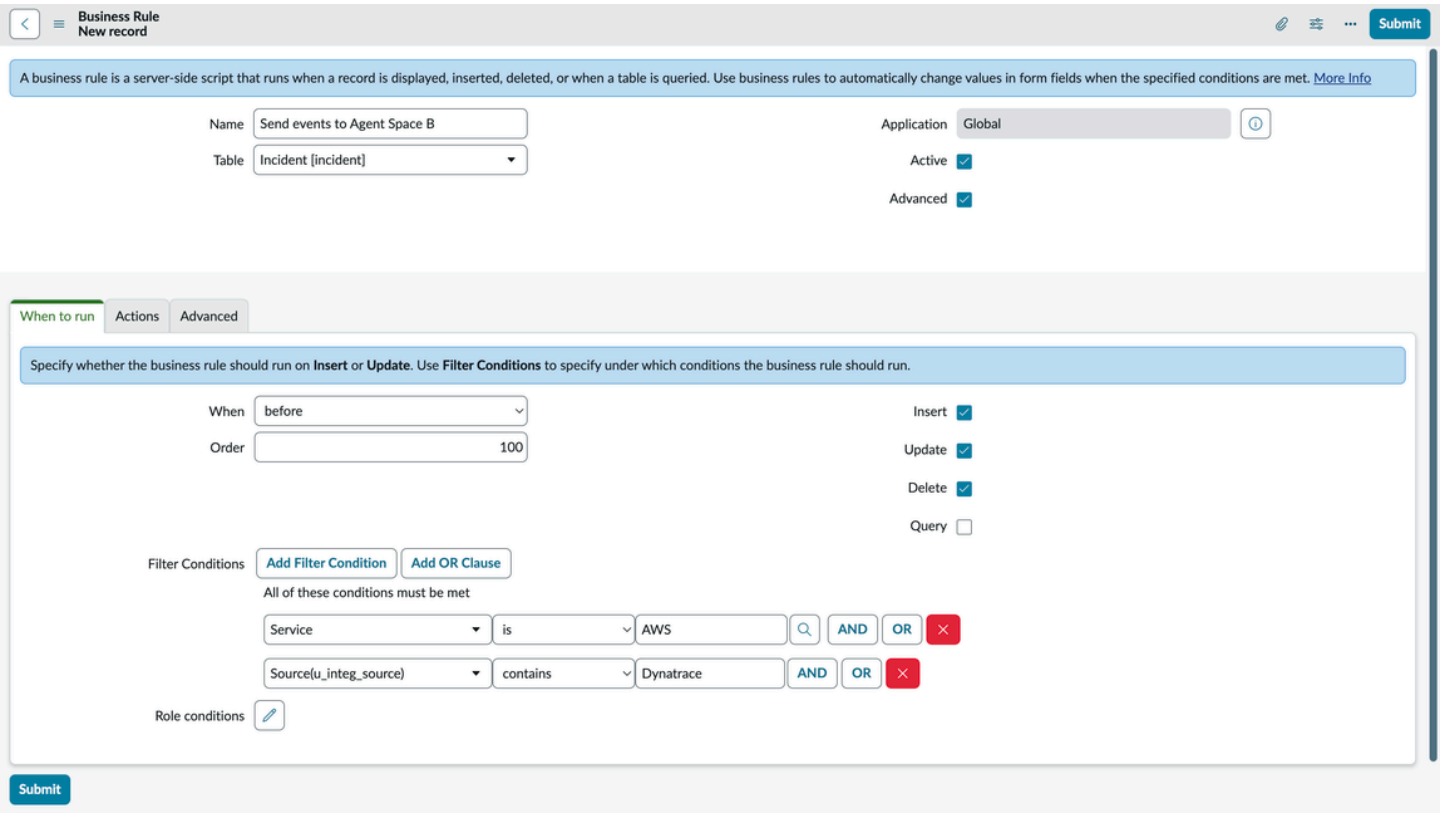
This example shows how to trigger an Investigation in DevOps Agent Space B when the urgency is 1, category is `Software` , or Service is `AWS`, and trigger an Investigation in DevOps Agent Space A when the service is `AWS`, and source is `Dynatrace`.

This scenario can be accomplished in two ways. The webhook script itself can be updated to include this business logic. In this scenario we will show how to accomplish it with a ServiceNow Business Rule, for transparency and simplify debugging. Routing is accomplished by creating two Service Now Business Rules.

- Create a Business Rule in ServiceNow for DevOps Agent Space A and create a condition using the condition builder to only send the events based on our specified condition.



- Next, create another Business Rule in ServiceNow for AgentSpace B for which the business rule will only trigger when Service is AWS and source is Dynatrace.



Now, when you create a new Incident that matches the condition specified, it will either trigger an investigation on DevOps Agent Space A or DevOps Agent Space B, providing you with fine grained control over incident routing.

Connecting Slack

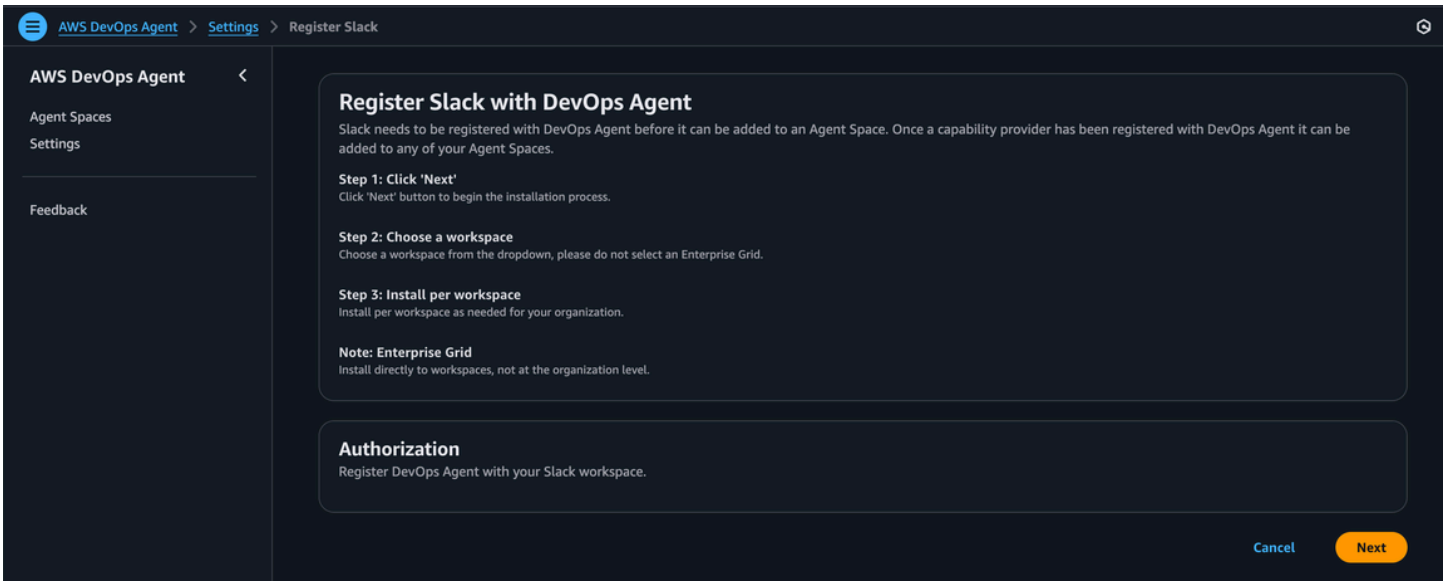
You can configure AWS DevOps Agent to update a Slack channel you select with incident response investigation key findings, root cause analyses, and generated mitigation plans.

Before you begin

Slack needs to be registered with DevOps Agent before it can be added to an Agent Space. To integrate AWS DevOps Agent with Slack you must meet these requirements:

- Have access to a Slack workspace with the ability to install and authorize third-party applications
- Have identified the Slack channels where you want AWS DevOps Agent to send notifications

Register Slack integration with AWS DevOps Agent



1. From the **Settings** tab in the AWS DevOps Agent console, navigate to **Communications > Slack.**
2. Choose the **Register** button.
3. You will be redirected to Slack to authorize the AWS DevOps Agent application for your workspace.
   a. Install directly to workspaces, not at the organization level.
   b. Choose a workspace from the dropdown. Do not select an Enterprise Grid.
   c. Install per workspace as needed for your organization.
4. Review the requested scopes and click **Allow** to authorize the integration.
5. After authorization, you'll return to the AWS DevOps Agent console.

Associate Slack with your DevOps Agent Space(s)

After registering Slack in your DevOps Agent Space, you can associate it with your DevOps Agent Space(s):

1. From the **Capabilities** tab within your configured AgentSpace, navigate to **Communications** > **Slack**.
2. Select **Add Slack**
3. Enter the Channel ID
4. Choose **Create** to complete the Slack configuration.

Note

The agent's bot user must be added to private channels before it can post messages.

Important

Uninstalling the Slack app may result in the Slack app not being able to be reinstalled. Avoid uninstalling the Slack app during the public preview.

Invoking DevOps Agent through Webhook

Webhooks allow external systems to automatically trigger AWS DevOps Agent investigations. This enables integration with ticketing systems, monitoring tools, and other platforms that can send HTTP requests when incidents occur.

Prerequisites

Before configuring webhook access, ensure you have:

- An Agent Space configured in AWS DevOps Agent
- Access to the AWS DevOps Agent console
- The external system that will send webhook requests

Webhook types

AWS DevOps Agent supports two types of webhooks:

- **Integration-specific webhooks** – Automatically generated when you configure third-party integrations like Dynatrace, Splunk, Datadog, New Relic, ServiceNow, or Slack. These webhooks are associated with the specific integration and use authentication methods determined by the integration type
- **Generic webhooks** – Can be manually created for triggering investigations from any source not covered by a specific integration. Generic webhooks currently use **HMAC** authentication (bearer token not currently available).

Webhook authentication methods

The authentication method for your webhook depends on which integration it's associated with:

HMAC authentication – Used by:

- Dynatrace integration webhooks
- Generic webhooks (not linked to a specific third-party integration)

Bearer token authentication – Used by:

- Splunk integration webhooks
- Datadog integration webhooks
- New Relic integration webhooks
- ServiceNow integration webhooks
- Slack integration webhooks

Configuring webhook access

Step 1: Navigate to the webhook configuration

1. Sign in to the AWS Management Console and navigate to the AWS DevOps Agent console
2. Select your Agent Space
3. Go to the **Capabilities** tab
4. In the **Webhook** section, click **Configure**

Step 2: Generate webhook credentials

For integration-specific webhooks:

Webhooks are automatically generated when you complete the configuration of a third-party integration. The webhook endpoint URL and credentials are provided at the end of the integration setup process.

For generic webhooks:

1. Click **Generate webhook**

2. The system will generate an HMAC key pair
3. Securely store the generated key and secret—you won't be able to retrieve them again
4. Copy the webhook endpoint URL provided

Step 3: Configure your external system

Use the webhook endpoint URL and credentials to configure your external system to send requests to AWS DevOps Agent. The specific configuration steps depend on your external system.

Managing webhook credentials

Removing credentials – To delete webhook credentials, go to the webhook configuration section and click **Remove**. After removing credentials, the webhook endpoint will no longer accept requests until you generate new credentials.

Regenerating credentials – To generate new credentials, remove the existing credentials first, then generate a new key pair or token.

Using the webhook

Webhook request format

To trigger an investigation, your external system should send an HTTP POST request to the webhook endpoint URL.

For Version 1 (HMAC authentication):

Headers:

- `Content-Type: application/json`
- `x-amzn-event-signature: <HMAC signature>`
- `x-amzn-event-timestamp: <+%Y-%m-%dT%H:%M:%S.000Z>`

The HMAC signature is generated by signing the request body with your secret key using SHA-256.

For Version 2 (Bearer token authentication):

Headers:

- `Content-Type: application/json`

- `Authorization: Bearer <your-token>`

Request body:

The request body should include information about the incident:

```
json

{
  "title": "Incident title",
  "severity": "high",
  "affectedResources": ["resource-id-1", "resource-id-2"],
  "timestamp": "2025-11-23T18:00:00Z",
  "description": "Detailed incident description",
  "data": {
    "metadata": {
        "region": "us-east-1",
        "environment": "production"
    }
  }
}
```

Example code

Version 1 (HMAC authentication) - JavaScript:

```
const crypto = require('crypto');

// Webhook configuration
const webhookUrl = 'https://your-webhook-endpoint.amazonaws.com/invoke';
const webhookSecret = 'your-webhook-secret-key';

// Incident data
const incidentData = {
    eventType: 'incident',
    incidentId: 'incident-123',
    action: 'created',
    priority: "HIGH",
    title: 'High CPU usage on production server',
    description: 'High CPU usage on production server host ABC in AWS account 1234
 region us-east-1',
    timestamp: new Date().toISOString(),
```

```
    service: 'MyTestService',
    data: {
      metadata: {
        region: 'us-east-1',
        environment: 'production'
      }
    }
};

// Convert data to JSON string
const payload = JSON.stringify(incidentData);
const timestamp = new Date().toISOString();
const hmac = crypto.createHmac("sha256", webhookSecret);
hmac.update(`${timestamp}:${payload}`, "utf8");
const signature = hmac.digest("base64");

// Send the request
fetch(webhookUrl, {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'x-amzn-event-timestamp': timestamp,
    'x-amzn-event-signature': signature
  },
  body: payload
})
.then(res => {
  console.log(`Status Code: ${res.status}`);
  return res.text();
})
.then(data => {
  console.log('Response:', data);
})
.catch(error => {
  console.error('Error:', error);
});
```

Version 1 (HMAC authentication) - cURL:

```
#!/bin/bash

# Configuration
WEBHOOK_URL="https://event-ai.us-east-1.api.aws/webhook/generic/YOUR_WEBHOOK_ID"
```

```
SECRET="YOUR_WEBHOOK_SECRET"

# Create payload
TIMESTAMP=$(date -u +%Y-%m-%dT%H:%M:%S.000Z)
INCIDENT_ID="test-alert-$(date +%s)"

PAYLOAD=$(cat <<EOF
{
"eventType": "incident",
"incidentId": "$INCIDENT_ID",
"action": "created",
"priority": "HIGH",
"title": "Test Alert",
"description": "Test alert description",
"service": "TestService",
"timestamp": "$TIMESTAMP"
}
EOF
)

# Generate HMAC signature
SIGNATURE=$(echo -n "${TIMESTAMP}:${PAYLOAD}" | openssl dgst -sha256 -hmac "$SECRET" -
binary | base64)

# Send webhook
curl -X POST "$WEBHOOK_URL" \
-H "Content-Type: application/json" \
-H "x-amzn-event-timestamp: $TIMESTAMP" \
-H "x-amzn-event-signature: $SIGNATURE" \
-d "$PAYLOAD"
```

Version 2 (Bearer token authentication) - JavaScript:

```
function sendEventToWebhook(webhookUrl, secret) {
  const timestamp = new Date().toISOString();

  const payload = {
    eventType: 'incident',
    incidentId: 'incident-123',
    action: 'created',
    priority: "HIGH",
    title: 'Test Alert',
    description: 'Test description',
```

```
    timestamp: timestamp,
    service: 'TestService',
    data: {}
  };

  fetch(webhookUrl, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      "x-amzn-event-timestamp": timestamp,
      "Authorization": `Bearer ${secret}`,  // Fixed: template literal
    },
    body: JSON.stringify(payload),
  });
}
```

Version 2 (Bearer token authentication) - cURL:

```
#!/bin/bash

# Configuration
WEBHOOK_URL="https://event-ai.us-east-1.api.aws/webhook/generic/YOUR_WEBHOOK_ID"
SECRET="YOUR_WEBHOOK_SECRET"

# Create payload
TIMESTAMP=$(date -u +%Y-%m-%dT%H:%M:%S.000Z)
INCIDENT_ID="test-alert-$(date +%s)"

PAYLOAD=$(cat <<EOF
{
"eventType": "incident",
"incidentId": "$INCIDENT_ID",
"action": "created",
"priority": "HIGH",
"title": "Test Alert",
"description": "Test alert description",
"service": "TestService",
"timestamp": "$TIMESTAMP"
}
EOF
)

# Send webhook
```

```
curl -X POST "$WEBHOOK_URL" \
-H "Content-Type: application/json" \
-H "x-amzn-event-timestamp: $TIMESTAMP" \
-H "Authorization: Bearer $SECRET" \
-d "$PAYLOAD"
```

Troubleshooting webhooks

If you do not receive a 200

A 200 and a message like webhook received indicate the authentication passed and the message has been queued for the system to verify and process. If you do not get a 200 but a 4xx most likely there is something wrong with the authentication or headers. Try sending manually using the curl options to help debug the authentication.

If you receive a 200 but an investigation does not start

Likely cause is a misformated payload.

1. Check both timestamp and incident id are updated and unique. Duplicate messages are deduplicated.
2. Check the message is valid JSON
3. Check the format is correct

If you receive a 200 and investigation is immediately cancelled

Most likely you have hit the limit for the month. Please talk to your AWS contact to ask for a rate limit change if appropriate.

Related topics

- the section called “Creating an Agent Space”
- the section called “What is a DevOps Agent Web App?”
- the section called “DevOps Agent IAM permissions”

AWS DevOps Agent Security

This document provides information about security considerations, data protection, access controls, and compliance capabilities for AWS DevOps Agent. Use this information to understand how AWS DevOps Agent is designed to meet your security and compliance requirements.

Agent Spaces

Agent Spaces serve as the primary security boundary in AWS DevOps Agent. Each Agent Space:

- Operates independently with its own configurations and permissions
- Defines which AWS accounts and resources the agent can access
- Establishes connections to third-party platforms

Agent Spaces maintain strict isolation to ensure security and prevent unintended access across different environments or teams.

Regional processing and data flow

AWS DevOps Agent operates from the US East (N. Virginia) region (us-east-1) as its primary processing hub. The agent retrieves operational data from AWS regions across all AWS accounts granted access within the configured Agent Space. This multi-region cross-account data collection ensures comprehensive incident analysis.

Amazon Bedrock usage and cross-region inference

AWS DevOps Agent will automatically select the optimal region within the US to process your inference requests. This maximizes available compute resources, model availability, and delivers the best customer experience. Your data will remain stored in the US East (N. Virginia) region, but inputs, requests, and output may be processed in other US regions. All data will be transmitted encrypted across Amazon's secure network.

- DevOps Agent and Bedrock are not impacted by customer policies in Service Control Policies (SCPs) or Control Tower that restrict customer content to specific regions
- Bedrock may use U.S. regions other than US East (N. Virginia) to perform stateless inference to optimize performance and availability

Identity and access management

Authentication methods

AWS DevOps Agent provides two authentication methods to log into the AWS DevOps Agent Space web app:

- **AWS Identity Center integration** – The primary authentication method uses OAuth 2.0 with session-based authentication using HTTP-only cookies. AWS Identity Center can federate with external identity providers through standard OIDC and SAML protocols, including providers like Okta, Ping Identity, and Microsoft Entra ID. This method supports multi-factor authentication through your identity provider. AWS Identity Center defaults to session durations of up to 12 hours and can be configured to a desired duration.
- **IAM authentication link** – An alternative method provides direct access to the web app from the AWS Management Console using JWT-based tokens derived from an existing AWS Management Console session. This option is useful for evaluating AWS DevOps Agent before implementing full Identity Center integration as well as gaining administrative access if the AWS DevOps Agent web app becomes inaccessible through Identity Center based authentication. Sessions are limited to 30 minutes.

IAM roles

AWS DevOps Agent uses IAM roles to define access permissions:

- **Primary account role** – Grants the agent access to resources in the AWS account where you create the Agent Space as well as access to secondary account roles.
- **Secondary account roles** – Grants the agent access to resources in additional AWS accounts connected to the Agent Space.
- **Web app role** – Grants users access to AWS DevOps Agent investigation data and findings in the web app.

These roles should be configured following the principle of least privilege, granting only the necessary read-only permissions required for investigations.

Data protection

Data encryption

AWS DevOps Agent encrypts all customer data:

- **Encryption at rest** – All data is encrypted with AWS-managed keys.
- **Encryption in transit** – All retrieved logs, metrics, knowledge items, ticket metadata, and other data are encrypted in transit inside the agent's private network and to outside networks.

Data storage and retention

Data is stored in the US East (N. Virginia) region

Personal identifiable information (PII)

AWS DevOps Agent does not filter PII information when summarizing data gathered during investigations, recommendation evaluations, or chat responses. It is recommended that PII data be redacted before storing in observability logs.

Agent journal and audit logging

Agent journal

Both the Incident Investigation and Prevention capabilities maintain detailed journals that:

- Log every reasoning step and action taken
- Create complete transparency into agent decision-making processes
- Cannot be modified by the agents once recorded, minimizing attacks such as prompt injection from hiding important actions
- Include all chat messages from the Investigation page

AWS CloudTrail integration

All AWS DevOps Agent API calls are automatically captured by AWS CloudTrail within the hosting AWS account. Using the information collected by CloudTrail, you can determine:

- The request that was made to the agent
- The IP address from which the request was made
- Who made the request
- When it was made

Prompt injection protection

A prompt injection attack occurs when an attacker embeds malicious instructions into external data, such as a webpage or document, that a generative AI system will later process. AWS DevOps Agent natively consumes many data sources as part of its normal operations, including logs, resource tags, and other operational data. AWS DevOps Agent protects against prompt injection attacks through the safeguards below, but it is important to ensure all connected data sources and user access to those data sources are trusted. See *AWS DevOps Agent Security* section for more.

Prompt injection safeguards:

- **Limited write capabilities** – The tools available to the agent are not able to mutate resources, with the exception of opening tickets and support cases. This prevents malicious instructions from modifying your infrastructure or applications.
- **Account boundary enforcement** – AWS DevOps Agent only operates within the boundary permitted by the roles assigned to the agent in the primary and connected secondary AWS accounts. The agent cannot access or modify resources outside of its configured scope.
- **AI safety protections** – AWS DevOps Agent uses Claude Sonnet 4.5, which includes AI Safety Level 3 (ASL-3) protections. These protections include classifiers that detect and prevent prompt injection attacks before they can affect agent behavior.
- **Immutable audit trail** – The agent journal logs every reasoning step and action taken. Journal entries cannot be modified by the agent once recorded, preventing prompt injection attacks from hiding malicious actions.

While AWS DevOps Agent provides multiple layers of protection against prompt injection attacks, certain configurations can increase risk:

- **Custom MCP server tools** – The bring-your-own MCP feature allows you to introduce custom tools to the agent, which can present additional opportunities for prompt injection. Custom tools may not have the same security controls as native AWS DevOps Agent tools, and malicious

instructions could potentially leverage these tools in unintended ways. See *AWS DevOps Agent Security* section for more.

- **Authorized user attacks** – Users who are authorized to operate within the AWS account boundary or connected tools have a higher chance of attempting an attack against the agent. These users may have the ability to modify data sources that the agent consumes, such as logs or resource tags, making it easier to embed malicious instructions that the agent will process.

To mitigate these risks:

1. Carefully review and test custom MCP servers before deploying them in Agent Spaces.
   a. Ensure they are only permitted to perform read-only actions
   b. Verify that users of external tools accessed by MCP servers are trusted entities, as AWS DevOps Agents interfacing with MCP rely on the implicit trust relationship established between these tool users and the AWS DevOps Agent
2. Apply the principle of least privilege when granting users access to systems that provide data to the agent
3. Regularly audit which MCP servers are connected to your Agent Spaces
4. Since any content retrieved from allowlisted URLs could attempt to manipulate the agent's behavior, only include trusted sources in your allowlist.

Integration security

AWS DevOps Agent supports several integration types, each with its own security model:

- **Native bidirectional integrations** – Built-in integrations that can send data to the agent and receive updates from the agent. This uses the vendor's authentication methods
- **MCP servers** – Remote Model Context Protocol servers that utilize OAuth 2.0 authentication flows and API keys to securely communicate with external systems.
- **Webhook triggers** – Investigation triggers from remote services such as tickets or observability systems. Webhooks use Hash-based Message Authentication Code (HMAC) for security.
- **Outbound communication** – Integrations like Slack and ticketing systems receive updates from the agent but do not yet support bidirectional communication.

Registration providers

Some external tools are authenticated at the account level and shared among all Agent Spaces in the account. When you register these tools, you authenticate once at the account level, and then each Agent Space can connect to specific resources within that registered connection.

The following tools use account-level registration:

- **GitHub** – Uses OAuth flow for authentication. After registering GitHub at the account level, each Agent Space can connect to specific repositories within your GitHub organization.
- **Dynatrace** – Uses OAuth token authentication. After registering Dynatrace at the account level, each Agent Space can connect to specific Dynatrace environments or monitoring configurations.
- **Slack** – Uses OAuth token authentication. After registering Slack at the account level, each Agent Space can connect to specific Slack channels channels.
- **Datadog** – Uses MCP with OAuth flow for authentication. After registering Datadog at the account level, each Agent Space can connect to specific Datadog monitoring resources.
- **New Relic** – Uses API key authentication. After registering New Relic at the account level, each Agent Space can connect to specific New Relic monitoring configurations.
- **Splunk** – Uses bearer token authentication. After registering Splunk at the account level, each Agent Space can connect to specific Splunk data sources.
- **GitLab** – Uses access token authentication. After registering GitLab at the account level, each Agent Space can connect to specific GitLab repositories.
- **ServiceNow** – Uses OAuth client key/token authentication. After registering ServiceNow at the account level, each Agent Space can connect to specific ServiceNow instances or ticket queues.
- **General public accessible remote MCP servers** – Use OAuth flow for authentication. After registering a remote MCP server at the account level, each Agent Space can connect to specific resources exposed by that server.

Network connectivity

AWS DevOps Agent connects to third-party systems, including remote MCP servers, from the following public IP addresses:

- 34.228.181.128
- 44.219.176.187

- 54.226.244.221

If your third-party systems or remote MCP servers use IP allowlisting or firewall rules, you must allow inbound connections from these IP addresses to enable AWS DevOps Agent to access your integrations.

Shared responsibility model

AWS responsibilities

AWS is responsible for:

- Maintaining the security of data retrieved by the agent
- Securing native tools available for use by the agent
- Protecting the infrastructure that runs AWS DevOps Agent

Customer responsibilities

Customers are responsible for:

- Managing user access to the agent space
- Limiting access to trusted users of external systems that provide inputs to the agent, such as services and resources that produce logs, CloudTrail events, tickets, and more – that may be used to attempt malicious prompt injection.
- Ensure all connected data sources have trusted data that is unlikely to be used to attempt prompt injection attacks
- Ensuring bring-your-own MCP server integrations operate securely
- Ensuring IAM roles assigned to the agent are properly scoped
- Redacting PII data before storing in observability logs and other agent data sources
- Following the recommended practice of granting only read-only permissions to connected data sources, including bring-your-own MCP servers

Data usage

AWS does not use agent data, chat messages, or data from integrated data sources to train models or improve the product. The AWS DevOps Agent Space uses customer in-product feedback to improve the agent's responses and investigations, but AWS does not use it to improve the service itself.

Compliance

At preview, AWS DevOps Agent is not compliant with standards including SOC 2, PCI-DSS, ISO 27001, or FedRAMP. AWS will announce which compliance certifications will be available at a later time.

DevOps Agent IAM permissions

AWS DevOps Agent uses service-specific IAM actions to control access to its features and capabilities. These actions determine what users can do within the AWS DevOps Agent console and Operator Web App. This is separate from the AWS service API permissions that the agent itself uses to investigate your resources. See Limiting Agent Access in an AWS Account[link] to learn more about managing the services and resources the agent itself has access to.

Agent Space management actions

These actions control access to Agent Space configuration and management:

- **aidevops:GetAgentSpace** – Allows users to view details about an Agent Space, including its configuration, status, and associated accounts. Users need this permission to access an Agent Space in the AWS Management Console.
- **aidevops:GetAssociation** – Allows users to view details about a specific account association, including the IAM role configuration and connection status.
- **aidevops:ListAssociations** – Allows users to list all AWS account associations configured for an Agent Space, including both primary and secondary accounts.

Investigation and execution actions

These actions control access to incident investigation features:

- **aidevops:SendChatMessage** – Allows users to ask questions about investigation or mitigation activities and provide instructions.
- **aidevops:ListExecutions** – Allows users to view execution metadata—including ID, status, and more—for investigations, mitigations, evaluations, and chat conversations associated with a task.
- **aidevops:ListJournalRecords** – Allows users to access detailed logs that show the agent's reasoning steps, actions taken, and data sources consulted during an investigation, mitigation, evaluation and chat conversations. This is useful for understanding how the agent reached its conclusions.

Topology and discovery actions

These actions control access to application resource mapping features:

- **aidevops:DiscoverTopology** – Allows users to trigger topology discovery and mapping for an Agent Space. This action initiates the process of scanning AWS accounts and building the application resource topology.

Prevention and recommendation actions

These actions control access to the Prevention feature:

- **aidevops:ListGoals** – Allows users to view prevention goals and objectives that the agent is working toward based on recent incident patterns.
- **aidevops:ListRecommendations** – Allows users to view all recommendations generated by the Prevention feature, including their priority and category.
- **aidevops:GetRecommendation** – Allows users to view detailed information about a specific recommendation, including the incidents it would have prevented and implementation guidance.

Backlog task management actions

These actions control the ability to manage recommendations as backlog tasks:

- **aidevops:CreateBacklogTask** – Allows users to create an incident investigation or prevention evaluation task.
- **aidevops:UpdateBacklogTask** – Allows users to approve a mitigation plan or cancel an active investigation or evaluation.

- **aidevops:GetBacklogTask** – Allows users to retrieve details about a specific task.
- **aidevops:ListBacklogTasks** – Allows users to list tasks for an Agent Space, filtered by task type, status, priority, or creation time.

Knowledge management actions

These actions control the ability to add and manage custom knowledge that the agent can use during investigations:

- **aidevops:CreateKnowledgeItem** – Allows users to add custom knowledge items, such as runbooks, troubleshooting guides, or application-specific information that the agent should reference.
- **aidevops:ListKnowledgeItems** – Allows users to view all knowledge items configured for an Agent Space.
- **aidevops:GetKnowledgeItem** – Allows users to retrieve the details of a specific knowledge item.
- **aidevops:UpdateKnowledgeItem** – Allows users to modify existing knowledge items to keep information current.
- **aidevops:DeleteKnowledgeItem** – Allows users to remove knowledge items that are no longer relevant.

AWS Support integration actions

These actions control integration with AWS Support cases:

- **aidevops:InitiateChatForCase** – Allows users to start a chat session with AWS Support directly from an investigation, automatically providing context about the incident.
- **aidevops:EndChatForCase** – Allows users to end an active AWS Support case chat session.
- **aidevops:DescribeSupportLevel** – Allows users to check the AWS Support plan level for the account to determine available support options.

Usage and monitoring actions

These actions control access to usage information:

- **aidevops:GetAccountUsage** – Allows users to view the AWS DevOps Agent monthly quota for investigation hours, prevention evaluation hours, and chat requests, as well as the current month's usage.

Common IAM policy examples

Administrator policy

This policy grants full access to all AWS DevOps Agent features:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "aidevops:*",
      "Resource": "*"
    }
  ]
}
```

Operator policy

This policy grants access to investigation and prevention features without administrative capabilities:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "aidevops:GetAgentSpace",
        "aidevops:InvokeAgent",
        "aidevops:ListExecutions",
        "aidevops:ListJournalRecords",
        "aidevops:ListAssociations",
        "aidevops:GetAssociation",
        "aidevops:DiscoverTopology",
        "aidevops:ListRecommendations",
        "aidevops:GetRecommendation",
```

```
        "aidevops:CreateBacklogTask",
        "aidevops:UpdateBacklogTask",
        "aidevops:GetBacklogTask",
        "aidevops:ListBacklogTasks",
        "aidevops:ListKnowledgeItems",
        "aidevops:GetKnowledgeItem",
        "aidevops:InitiateChatForCase",
        "aidevops:EndChatForCase",
        "aidevops:SendChatMessage",
        "aidevops:ListGoals",
        "aidevops:CreteKnowledgeItem",
        "aidevops:UpdateKnowledgelItem",
        "aidevops:DescribeSupportLevel",
        "aidevops:ListPendingMessages"
      ],
      "Resource": "*"
    }
  ]
}
```

Read-only policy

This policy grants view-only access to investigations and recommendations:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "aidevops:GetAgentSpace",
        "aidevops:ListAssociations",
        "aidevops:GetAssociation",
        "aidevops:ListExecutions",
        "aidevops:ListJournalRecords",
        "aidevops:ListRecommendations",
        "aidevops:GetRecommendation",
        "aidevops:ListBacklogTasks",
        "aidevops:GetBacklogTask",
        "aidevops:ListKnowledgeItems",
        "aidevops:GetKnowledgeItem",
        "aidevops:GetAccountUsage"
      ],
```

```
      "Resource": "*"
    }
  ]
}
```

Limiting Agent Access in an AWS Account

AWS DevOps Agent uses IAM roles to discover and describe AWS resources during incident investigations and preventative evaluations. You can control the level of access the agent has by configuring IAM policies attached to these roles. The application topology doesn't show everything the agent has access to—IAM policies are the only way to truly limit what AWS service APIs and resources the agent can access.

Understanding IAM roles for AWS DevOps Agent

AWS DevOps Agent uses IAM roles to access resources in two types of accounts:

- **Primary account role** – Grants the agent access to resources in the AWS account where you create the Agent Space.
- **Secondary account roles** – Grants the agent access to resources in additional AWS accounts that you connect to the Agent Space.

For either type of account, you can restrict which AWS services the agent can access, limit access to specific resources within those services, and control which regions the agent can operate in.

Choosing your resource boundaries

When limiting resource access, you need to include enough permissions for the agent to successfully investigate application incidents. This includes:

- All resources for in-scope applications that the agent should monitor and investigate
- All supporting infrastructure that those applications depend on

Supporting infrastructure may include:

- Networking components (VPCs, subnets, load balancers, API gateways)
- Data stores (databases, caches, object storage)

- Compute resources (EC2 instances, Lambda functions, containers)
- Monitoring and logging services (CloudWatch, CloudTrail)
- Identity and access management resources needed to understand permissions

If you restrict access too narrowly, the agent may not be able to identify root causes that originate in supporting infrastructure outside your defined boundaries.

Restricting service access

You can limit which AWS services the agent can access by modifying the IAM policies attached to the agent's roles. When creating custom policies, follow these best practices:

- **Grant only read-only permissions** – The agent needs to read resource configurations, metrics, and logs during investigations. Avoid granting permissions that allow the agent to modify or delete resources.
- **Limit to necessary services** – Include only the AWS services that contain resources relevant to your applications. For example, if your application doesn't use Amazon RDS, don't include RDS permissions in the policy.
- **Use specific actions instead of wildcards** – Instead of granting `service:*` permissions, specify individual actions like `cloudwatch:GetMetricData` or `ec2:DescribeInstances`.

Example policy restricting to specific services:

```
json

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:GetMetricData",
        "cloudwatch:GetMetricStatistics",
        "cloudwatch:DescribeAlarms",
        "logs:GetLogEvents",
        "logs:FilterLogEvents",
        "ec2:DescribeInstances",
        "lambda:GetFunction",
```

```
        "lambda:GetFunctionConfiguration"
      ],
      "Resource": "*"
    }
  ]
}
```

Restricting resource access

To limit the agent to specific resources within a service, use resource-level permissions in your IAM policies. This allows you to grant access only to resources that match specific patterns.

Using resource ARN patterns:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lambda:GetFunction",
        "lambda:GetFunctionConfiguration"
      ],
      "Resource": "arn:aws:lambda:*:*:function:production-*"
    }
  ]
}
```

This example limits the agent to accessing only Lambda functions with names that begin with "production-".

Using tag-based restrictions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeInstances",
        "ec2:DescribeInstanceStatus"
      ],
```

```
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Environment": "production"
        }
      }
    }
  ]
}
```

This example limits the agent to accessing only EC2 instances tagged with `Environment=production`.

Restricting regional access

To limit which AWS regions the agent can access, use the `aws:RequestedRegion` condition key in your IAM policies:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:Describe*",
        "lambda:Get*",
        "cloudwatch:Get*"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": [
            "us-east-1",
            "us-west-2"
          ]
        }
      }
    }
  ]
}
```

This example limits the agent to accessing resources only in the us-east-1 and us-west-2 regions.

Creating custom IAM policies

When you create an Agent Space or add secondary accounts, you have the option to create a custom IAM role using a policy template. This allows you to implement the principle of least privilege.

When creating an Agent Space

From the DevOps Agent console in the AWS Management Console...

- Choose **Create a new DevOps Agent role using a policy document** and follow the instructions

When editing an Agent Space

From the DevOps Agent console in the AWS Management Console...

- Select the **Capabilities** tab
- Select the secondary account you want to edit from the **Cloud** section and click Edit
- Chose **Create a new DevOps Agent policy using a template** and follow the instructions

Custom policy best practices

- **Grant only read-only permissions** – Avoid permissions that allow resource modification or deletion
- **Use resource-level permissions when possible** – Restrict access to specific resources using ARN patterns or tags
- **Regularly review and audit permissions** – Periodically review the agent's IAM policies to ensure they still align with your security requirements

Setting Up IAM Identity Center Authentication

IAM Identity Center authentication provides a centralized way to manage user access to the AWS DevOps Agent Space web application. This guide explains how to configure IAM Identity Center authentication and manage users.

Prerequisites

Before setting up IAM Identity Center authentication, ensure you have:

- IAM Identity Center enabled in your organization or account
- Administrator permissions in AWS DevOps Agent
- An Agent Space configured or ready to create

Authentication options

AWS DevOps Agent offers two authentication methods for accessing the Agent Space web app:

IAM Identity Center authentication – Recommended for production environments. Provides centralized user management, integration with external identity providers, and sessions up to 12 hours.

Admin access (IAM authentication) – Provides quick access for administrators during initial setup and configuration. Sessions are limited to 30 minutes.

Configuring IAM Identity Center during Agent Space creation

When you create an Agent Space, you can configure IAM Identity Center authentication on the **Web app** tab:

Step 1: Navigate to the Web app configuration

1. After configuring your Agent Space details and AWS account access, proceed to the **Web app** tab
2. You'll see two sections: "Connect IAM Identity Center" and "Admin access"

Step 2: Configure IAM Identity Center integration

In the **Connect [Agent Space] to IAM Identity Center** section:

1. **Verify the IAM Identity Center instance** – The console displays which Identity Center instance will manage Web App user access (for example, `ssoins-7223a9580931edbe`). Your closest IAM Identity Center instance will automatically be pre-populated.
2. **Select the IAM Identity Center Application Role Name option** – Choose one of three options:

Auto-create a new DevOps Agent role (recommended):

- The system automatically creates a new service role with appropriate permissions
- This is the simplest option and works for most use cases

Assign an existing role:

- Use an existing IAM role that you've already created
- The system will verify the role has the required permissions
- Choose this option if your organization has pre-created roles for AWS DevOps Agent

Create a new DevOps Agent role using a policy template:

- Use the provided policy details to create your own custom role in the IAM Console
- Choose this option if you need to customize the role permissions

After clicking Connect, the system automatically:

- Creates or configures the specified IAM role
- Sets up an IAM Identity Center application for your Agent Space
- Establishes trust relationships between IAM Identity Center and the Agent Space web app
- Configures OAuth 2.0 authentication flows for secure user access

Alternative: Using admin access

If you want to access the Agent Space web app immediately without setting up IAM Identity Center:

1. In the **Admin access** section, note the IAM Role ARN that provides administrator access (for example, `arn:aws:iam::440491339484:role/service-role/DevOpsAgentRole-WebappAdmin-15ppoc42`)
2. Click the blue **Admin access** button to launch the Agent Space web app with IAM authentication
3. Sessions using this method are limited to 30 minutes

> **Note**
>
> Admin access is intended for initial setup and configuration. For production use and ongoing operations, configure IAM Identity Center authentication.

Adding users and groups

After configuring IAM Identity Center authentication, you need to grant specific users and groups access to the Agent Space web app:

Step 1: Access user management

1. In the AWS DevOps Agent console, select your Agent Space
2. Go to the **Web app** tab
3. Under **User Access**, click **Manage Users and Groups**

Step 2: Add users or groups

1. Choose **Add Users or Groups**
2. Search for users or groups in your IAM Identity Center directory
3. Select the checkboxes next to the users or groups you want to add
4. Click **Add** to grant them access

The selected users can now access the Agent Space web app using their IAM Identity Center credentials.

Working with external identity providers

If you're using an external identity provider (such as Okta, Microsoft Entra ID, or Ping Identity) with IAM Identity Center:

- Users and groups are synchronized from your external identity provider to IAM Identity Center
- When you add users and groups to the Agent Space web app, you're selecting from the synchronized directory
- User attributes and group memberships are maintained by your external identity provider

- Changes in your identity provider are automatically reflected in IAM Identity Center after synchronization

How users access the Agent Space web app

After you've added users to your Agent Space:

1. Share the Agent Space web app URL with authorized users
2. When users navigate to the URL, they're redirected to the IAM Identity Center login page
3. After entering their credentials (and completing MFA if configured), they're redirected back to the Agent Space web app
4. Their session is valid for 8 hours by default (configurable by the Identity Center administrator)

Managing user access

You can update user access at any time:

Adding more users or groups:

- Follow the same steps described above to add additional users or groups

Removing access:

1. In the **User Access** section, find the user or group to remove
2. Click the **Remove** button next to their name
3. Confirm the removal

Removed users will lose access immediately, but active sessions may continue until they expire.

Session management

IAM Identity Center sessions for the Agent Space web app have the following characteristics:

- **Default session duration** – 8 hours
- **Session security** – HTTP-only cookies for enhanced protection
- **Multi-factor authentication** – Supported when configured in IAM Identity Center

- **API credentials** – Short-duration (15-minute) SigV4 credentials are issued for API calls and renewed automatically

To configure session duration:

1. Navigate to the IAM Identity Center console
2. Go to **Settings** > **Authentication**
3. Under **Session duration**, configure your preferred duration (from 1 hour to 12 hours)
4. Choose **Save changes**

Disconnecting Identity Center

1. In your Agent Space's console, click **Actions** in the top-right and select **Disconnect from IAM Identity Center**
2. Confirm in confirmation dialog