User Guide

Amazon Nova Act

Amazon Nova Act: User Guide

Table of Contents

What is Amazon Nova Act?

Amazon Nova Act is available as a new AWS service to build and manage fleets of reliable AI agents for automating production UI workflows at scale. Nova Act completes repetitive UI workflows in the browser and escalates to a human supervisor when appropriate. Nova Act also integrates with external tools through API calls, remote MCP, or agentic frameworks, such as Strand Agents (Preview). You can define workflows by combining the flexibility of natural language with Python code. Start by exploring in the web playground at nova.amazon.com/act, develop and debug in the Nova Act IDE extension, deploy to AWS, and monitor your workflows in the AWS Management Console, all in just a few steps.

Nova Act terminology

Nova Act uses the following concepts and terminology.

- **Act:** An act() call is how you pass a natural language task to the Nova Act model. For example: `nova.act("search for 'rubber duck debugging'")`. Each act() executes the agentic loop within a session context.
- **Step**: Each act() call consists of multiple steps. A step is one cycle of the model observing the page and taking an action. Steps run sequentially.
- **Session**: A session represents a browser instance or API client instance. A session contains one or more act() calls that run sequentially. Multiple sessions can run in parallel within a workflow run.
- **Workflow**: A workflow defines your agent's end-to-end task. Workflows are comprised of act() statements and Python code that orchestrate the automation logic.
- **Workflow run**: Each invocation of a workflow is a workflow run. A workflow can be run multiple times with different inputs, producing a begin time, end time, and result.

When to use Nova Act

Here are some common use cases for Nova Act.

- **Data Entry:** Replace manual copy-paste and repetitive typing with agents that understand and populate complex web forms across multiple sites or portals, boosting accuracy and throughput while freeing teams from undifferentiated work. This includes updating CRM records or executing tasks in ERP systems. Nova Act agents adapt to varied layouts and complete submissions consistently ingesting structured data.

- **Data Extraction:** Nova Act agents can navigate sites and search results, filter and locate relevant data, and extract key fields from unstructured web sources or dashboards. Ideal for gathering competitive intelligence, monitoring supplier portals, or consolidating data from industry-specific platforms that lack programmatic access or standardized data exports.
- **Checkout Flows:** Complete purchase or booking flows autonomously across e-commerce, travel, or ticketing sites from product selection through payment completion. Your agents can search inventory, apply rules or preferences, and execute end-to-end checkout flows, including handling payments, confirmations, and validation. Perfect for testing retail experiences, managing bulk transactions, or automating recurring procurement tasks or multi-step booking flows across diverse e-commerce platforms where traditional automation breaks down due to site variations and frequent UI changes.
- **Web QA Testing:** Accelerate release cycles with automated full user-journey validation directly in the browser. Your agents execute QA test cases intuitively and naturally navigating through UI workflows. This is ideal for teams managing detailed test scenarios and scaling coverage across web properties where maintaining traditional scripts can be resource-intensive and API testing can miss critical interface changes.

Benefits

Experience the benefits of Nova Act:

- **High reliability**: Nova Act achieves high reliability through vertical integration of a custom foundation model, orchestrator, browser actuator, and other tools, with end-to-end training across the full stack. This approach ensures reliable performance for browser-based automation workflows.
- **Ease of use**: Create reliable workflows using natural language prompts, without managing orchestration logic, setting up agent loops, tool integrations, or configuring models.
- **End-to-end developer experience**: Access a complete toolkit including a custom foundation model, web playground, Python SDK, IDE extension, CLI, AWSconsole, and Amazon Bedrock AgentCore Runtime and Browser.
- **Production-scale deployment**: Deploy workflows to AWS with built-in monitoring, management, and infrastructure that scales with your needs.

Availability

Nova Act is supported in the following AWS Region:

- US East (N. Virginia)

Pricing for Nova Act

For the most recent pricing information, visit the Amazon Nova Act pricing page.

Nova Act model version selection

The Nova Act service relies on a custom foundation model, specially trained for UI-forward applications such as browser automation, tool use, and Human-in-the-Loop.

We release models under two support levels:

- **General Availability (GA)** - Stable releases for production applications, supported for at least 1 year.
- **Preview** - Preview releases that give you an opportunity to try out new functionality and model performance.

There are two approaches for selecting the model version for your workflow:

- **Automatically track to the latest version**
  - Select the `nova-act-latest` alias to automatically use the latest GA model version.
  - Select the `nova-act-preview` alias to automatically use the latest preview model.
- **Pin to a specific model version** - Select a specific model version that will be supported for at least 1 year.

Note

`nova-act-latest` only tracks GA models and will never automatically update to a preview model.

Available model selection IDs

Model-Id	Description
`nova-act-latest`	Alias to the latest GA model. Your workflows automatically track the latest GA model that your Nova Act SDK version supports.
`nova-act-preview`	Alias to the latest preview model. Your workflow automatic ally tracks the latest preview model that your Nova Act SDK supports. **Note:** If the latest preview model requires a newer SDK version than you're using, you'll get the most recent GA model that your SDK supports, along with a warning message.
`nova-act-v1.0`	The GA model released on December 2, 2025. This is the first example of a pinable model selection ID.

Note

At launch (December 2, 2025), all three model IDs point to the same v1.0 model. The aliases automatically track newer versions as they become available.

Model Selection Options

Specify the model version using the **`model_id`** parameter in your workflow definition. The following example uses **`model_id="nova-act-latest"`**:

```
from nova_act import NovaAct, workflow

@workflow(workflow_definition_name="<your-workflow-name-here>", model_id="nova-act-
latest")
def explore_travel_destinations():
    """Explore travel destinations."""
    with NovaAct(starting_page="https://nova.amazon.com/act/gym") as nova:
        nova.act("Click on NextDot, then explore possible destinations.")
```

Understand the available interfaces for using Nova Act

Nova Act consists of the Nova Act AWS service for production deployment and monitoring, plus Nova Act developer tools (SDK, CLI, and IDE extension) that support your development journey from exploration to production. The Nova Act Playground provides a separate environment for initial exploration at nova.amazon.com/act.

Note

When using the Nova Act Playground and/or choosing Nova Act developer tools with API key authentication, access and use are subject to the nova.amazon.com Terms of Use. When choosing Nova Act developer tools with AWS IAM authentication and/or deploying workflows to the Nova Act AWS service, your AWS Service Terms and/or Customer Agreement (or other agreement governing your use of the AWS Service) apply.

The following interfaces are available:

Explore - Start with the Nova Act playground at nova.amazon.com/act to test automation ideas without any setup.

Develop - Use the Nova Act SDK and IDE extension to build and debug workflows locally.

Deploy - Use the Nova Act CLI to package and deploy workflows to AWS.

Monitor - Use the Nova Act AWS console to track execution and troubleshoot issues.

Most developers progress through these interfaces as they move from experimentation to production deployment.

Nova Act playground

The Nova Act playground provides an instant, browser-based environment to explore Nova Act capabilities, without any setup or AWS account. Test automation ideas using natural language commands, watch the agent execute actions in a hosted web browser, and export Python scripts when you’re ready to develop locally.

Access the playground at nova.amazon.com/act with your Amazon account.

Nova Act SDK

The Nova Act SDK enables you to build automation workflows quickly using Python. Install with a single command and define agents using Python code, natural language, or both. The SDK provides powerful debugging capabilities, such as setting breakpoints, adding assertions, and iterating rapidly during development. The SDK supports both API Key authentication for quick experimentation and AWS IAM authentication for production development.

To install the SDK:

```
pip install nova-act
```

Visit the nova-act GitHub repository for install instructions and detailed documentation.

Nova Act extension

The Nova Act extension brings the complete agent development experience into IDEs such as Visual Studio Code, Cursor, and Kiro. Build, test, and debug agents in a unified environment without switching between tools. Key features include Builder Mode with live browser preview, chat-to-generate scripts, code templates, and integrated deployment capabilities.

Visit the nova-act-extension GitHub repository for installation instructions and detailed documentation.

Nova Act CLI

The Nova Act CLI automates the deployment process by turning your local scripts into remote workflows on Amazon Bedrock AgentCore Runtime. It handles the complexity of containerization, ECR management, S3 bucket creation, and IAM role creation automatically. Use the CLI to create Nova Act workflow definitions, deploy your scripts, and invoke workflow—all from your command line with account-based configuration management. The CLI is also integrated into the Nova Act extension for seamless deployment as you develop. The CLI requires AWS credentials configured via the AWS CLI or AWS profiles.

Visit the nova-act GitHub repository for install instructions and detailed documentation.

Nova Act AWS console

The Nova Act AWS console provides a web-based interface to manage and monitor your deployed workflows. Create workflow definitions, view workflow runs with detailed execution traces, and analyze agent behavior through the observability hierarchy (runs → sessions → acts → steps). The console displays step execution data, observation and invocation loops, and provides access to artifacts stored in S3.

Access the Nova Act AWS console through the AWS Management Console.

Get started with Nova Act

Note

When using the Nova Act Playground and/or choosing Nova Act developer tools with API key authentication, access and use are subject to the nova.amazon.com Terms of Use. When choosing Nova Act developer tools with AWS IAM authentication and/or deploying workflows to the Nova Act AWS service, your AWS Service Terms and/or Customer Agreement (or other agreement governing your use of the AWS service) apply.

Follow these steps to get started:

Step 1: Explore Nova Act in the playground

Start by exploring Nova Act capabilities in the web playground—no setup or AWS account required. Test natural language commands, watch the agent execute actions, and when ready, download Python scripts for local development.

Step 2: Develop locally

Develop and debug workflows in your IDE such as Visual Studio Code, Cursor, and Kiro using the Nova Act extension. Choose API key authentication for quick experimentation, or AWS IAM authentication when you're ready for production development.

Step 3: Deploy to AWS

Deploy your workflows to AWS using the IDE extension's one-click deployment, or use the CDK deployment examples in the Nova Act repository to build production infrastructure tailored to your organization's needs.

Step 4: Review your workflow runs in the Nova Act AWS console

Monitor your deployed workflows in the Nova Act AWS console to track execution, view detailed traces, and troubleshoot issues.

Step 1: Explore Nova Act in the playground

The Nova Act playground allows you to explore the capabilities of Nova Act in a hosted web environment, no setup and no AWS account required.

To access the Nova Act playground:

- Navigate to nova.amazon.com/act and log in with your Amazon.com account. Access and use subject to the nova.amazon.com Terms of Use.
- Select **Nova Act Playground**.
- Experiment with Nova Act using natural language commands and watch the agent taking actions in the hosted web browser.
- When ready, download the Python script for local development.

Step 2: Develop locally

Prerequisites: Python development environment (Python 3.10 or later)

The Nova Act extension transforms how you build with Nova Act by bringing the entire agent development experience directly into IDEs such as Visual Studio Code, Cursor, and Kiro. Develop, test, and debug your agent in a single IDE environment.

Choose your authentication method

Nova Act supports two authentication methods for local development:

API key authentication

- **Prerequisites:** API key from nova.amazon.com/act. Your API key is tied to your Amazon.com account.
- **Billing:** Free of charge with daily limits
- **Best for:** Quick experimentation and prototyping

Note

By authenticating using your API key, your access and use of the Nova Act extension is subject to the nova.amazon.com Terms of Use and Amazon collects information from your interactions with Nova Act.

AWS IAM authentication (Recommended for production)

- **Prerequisites:** AWS account with IAM permissions (see AWS Managed Policies)
- **Billing:** Usage charges apply to your AWS account. Visit the Nova Act pricing page for details.
- **Best for:** Production development and deployment

Note

By authenticating using your AWS IAM, your access and use of the Nova Act extension is subject to your AWS Service Terms and/or Customer Agreement (or other agreement governing your use of the AWS service). Amazon temporarily collects information from your interactions with Nova Act to support runtime of the model.

When using the Nova Act SDK, you'll see a console message indicating which authentication method is active.

Which authentication method should I choose?

Use API key authentication when you want to quickly explore Nova Act capabilities without AWS setup. This is ideal for learning, prototyping, and validating automation ideas before committing to production deployment.

Use AWS IAM authentication when you're ready to build production workflows. This method provides full access to AWS services, monitoring, and production-scale infrastructure.

Develop with API key authentication

With API key authentication, you can develop Nova Act workflows using either the Nova Act SDK or the Nova Act extension. You can start from scratch, import scripts from the playground, or generate workflows using the IDE's chat experience.

1. Get your API key from nova.amazon.com/act
2. Install the Nova Act extension (check the nova-act-extension GitHub repository for instructions and supported environments).
3. Open Builder Mode in the Nova Act extension to access the live browser preview
4. Generate your workflow using one of these approaches:
   - **Chat-to-code**: Use GitHub Copilot in VS Code to describe your automation needs in natural language
     - Example: `@novaAct: "I need an agent that logs into a customer portal, searches for unresolved tickets, and updates their status based on completion criteria."`
   - **Import from playground**: Download a script you created in the Nova Act playground and open it in the IDE
   - **Start from template**: Select a workflow template from the extension
5. Run and debug your workflow in Builder Mode with live browser preview

Example script with API key authentication:

```
from nova_act import NovaAct
import os

# Browser args enables browser debugging on port 9222.
os.environ["NOVA_ACT_BROWSER_ARGS"] = "--remote-debugging-port=9222"
# Get your API key from https://nova.amazon.com/act
# Set API Key using Set API Key command (CMD/Ctrl+Shift+P) or set it below.
# os.environ["NOVA_ACT_API_KEY"] = "<YOUR_API_KEY>"

# Initialize Nova Act with your starting page.
nova = NovaAct(
    starting_page="https://nova.amazon.com/act/gym",
    headless=True,
    tty=False
)

# Running nova.start will launch a new browser instance.
# Only one nova.start() call is needed per Nova Act session.
nova.start()

# Add your nova.act(<prompt>) statement here
```

```
nova.act("Click on NextDot, then explore possible destinations.")

# Leaving nova.stop() commented keeps NovaAct session running.
# To stop a NovaAct instance, press "Restart Notebook" (top-right) or uncomment
 nova.stop() - note this also shuts down the browser instantiated by NovaAct so
 subsequent nova.act() calls will fail.
# nova.stop()
```

Develop with AWS IAM authentication

To develop workflows locally using AWS IAM authentication:

1. Configure AWS credentials with appropriate IAM roles and permissions (see AWS Managed Policies) in your development environment.
2. Create a Nova Act workflow definition using the Nova Act AWS console or CLI:
   - In the Nova Act AWS console, select **Create workflow definition** and follow the instructions to create a workflow definition.
   - Or, using the Nova Act CLI, run `act workflow create —name <your-workflow-name>`
3. Use the AWS workflow template in the IDE extension or an example script, such as the one shown below, to start developing your script.

Example script with AWS IAM authentication (using workflow context manager)

```
import os
from nova_act import NovaAct, Workflow

def main(payload):
    with Workflow(
        workflow_definition_name="<your-workflow-name>",
        model_id="nova-act-latest"
    ) as workflow:
        with NovaAct(
            starting_page="https://nova.amazon.com/act/gym",
            workflow=workflow,
            headless=True,
            tty=False
        ) as nova:
            nova.act("Click on NextDot, then explore possible destinations.")

if __name__ == "__main__":
    main({})
```

Example script with AWS IAM authentication (using @workflow Python decorator to wrap your function)

```
from nova_act import NovaAct, workflow

@workflow(workflow_definition_name="<your-workflow-name>", model_id="nova-act-latest")
def search_space_destinations():
    with NovaAct(
        starting_page="https://nova.amazon.com/act/gym",
        headless=True,
        tty=False
    ) as nova:
        nova.act("Click on NextDot, then explore possible destinations.")

# main function MUST accept payload
def main(payload):
    search_space_destinations()

if __name__ == "__main__":
    main({})
```

Step 3: Deploy to AWS

Nova Act provides two deployment approaches depending on your needs.

Note

Your deployment to AWS is subject to AWS Service Terms and/or Customer Agreement (or other agreement governing your use of the AWS Service). Deployment creates AWS resources (AgentCore Runtime, ECR repository, S3 bucket) that incur separate charges. See Amazon Nova Act pricing for details.

Quick deployment to AWS with the IDE extension

For rapid deployment of individual workflows, use the Nova Act extension's built-in deployment capability.

To deploy your workflow:

1. Navigate to the **Deploy** tab in the Nova Act extension.
2. Verify your AWS credentials with `aws sts get-caller-identity`. If needed, configure or update your credentials.
3. Ensure your IAM Identity has a policy permitting read and write access to the Nova Act service (see AWS Managed Policies).
4. Enter your workflow name and select your AWS Region.
5. Click **Deploy your workflow**.

The IDE extension uses Amazon Bedrock AgentCore Runtime to automatically package your script into a container image and deploys it to AWS, creating all necessary resources including ECR repositories, S3 buckets, and IAM execution roles.

Deployment to AWS with CDK templates

For production environments, the nova-act-samples GitHub repository contains ready-to-use examples for deploying Nova Act on various AWS compute services. Each example includes a complete CDK construct, Docker configuration, and test scripts.

These examples can serve as a starting point for building production systems tailored to your organization’s needs, including enhanced monitoring, infrastructure management, and security configurations.

Step 4: Review your workflow runs in the Nova Act console

The Nova Act AWS console provides visibility into your workflow execution with detailed traces and artifacts.

To review a workflow run:

1. Navigate to the Nova Act AWS console.
2. From the workflow definitions list, select the workflow you want to review. Each workflow definition displays its name, creation date, and recent activity status.
3. Select a **Run ID** from the workflow runs list. Each entry shows the run ID, status (succeeded, failed, or in progress), when the run started and ended, and a link to download artifacts.

4. In the workflow run details page, you can see the run summary, execution timeline, selected model ID, and any artifacts generated during the workflow.
5. Use the **Step view** to drill down into specific sessions and act calls.

Human-in-the-loop (HITL)

Nova Act's Human-in-the-Loop (HITL) capability enables seamless human supervision within autonomous web workflows. HITL is available in the Nova Act SDK for you to implement in your workflows (not provided as a managed AWS service). When your workflow encounters scenarios requiring human judgment or intervention, HITL provides tools and interfaces for supervisors to assist, verify, or take control of the process.

HITL patterns

Nova Act supports the following HITL patterns:

Human approval

Human approval enables asynchronous human decision-making in automated processes. When Nova Act encounters a decision point requiring human judgment, it captures a screenshot of the current state and presents it to a human reviewer via a browser-based interface. Use this when you need binary or multi-choice decisions (Approve/Reject, Yes/No, or selecting from predefined options).

UI takeover

UI takeover enables real-time human control of a remote browser session. When Nova Act encounters a task that requires human interaction, it hands control of the browser to a human operator via a live-streaming interface. The operator can interact with the browser using mouse and keyboard in real-time.

Understanding HITL scenarios

Nova Act HITL supports the following intervention scenarios:

Using Human Approval. Useful for:

- **Expense and Purchase Approval**: Approve expense reports or purchase approvals.
- **Data Validation**: Confirm accuracy of data before submission.

Using UI Takeover. Useful for:

- **CAPTCHA Resolution**: Solve CAPTCHA challenges.
- **Login/AuthN Flows**: Enter username and password or MFA/2FA codes during login workflows.

Visit the nova-act-samples Github repository for examples of HITL use.

Implementing HITL

Nova Act provides two approaches for implementing HITL capabilities:

Option 1: Managed Human Intervention Service (Recommended)

Deploy the provided Human Intervention Service (HIS) that runs entirely within your AWS environment using the AWS CDK. This turnkey solution includes pre-built interfaces, workflows, and infrastructure.

Setup steps:

1. **Deploy the infrastructure:** Use the Human Intervention CDK package from the nova-act-human-intervention GitHub repository to deploy the HIS to your AWS account.
2. **Configure notifications:** The managed HIS supports Slack and email notifications to alert supervisors when intervention is needed.
   - **Slack**: Real-time notifications via Slack Bot SDK with rich formatting and threaded conversations
   - **Email**: HTML-formatted email notifications via AWS SES
3. **Integrate with your workflows:** Integrate the Human Intervention Service client SDK into your Nova Act workflows and configure callbacks for human approval or UI takeover patterns at the identified points with appropriate timeouts and retry policies. The managed HIS automatically handles supervisor routing and response collection. See the nova-act-human-intervention repository for code examples.
4. **Train supervisors:** Ensure your supervisors are familiar with the HIS interfaces.
   - **One-off action interface**: Streamlined experience for quick approvals or interventions
   - **Supervisor dashboard**: Comprehensive view of all HITL activities, including pending requests, historical actions, and performance metrics.

Option 2: Custom implementation

Implement the HITL interface yourself and integrate it with your own management system. This gives you complete control over the user interface, workflows, notifications, and integration with existing systems.

Setup steps:

- Integrate the HITL patterns (human approval and UI takeover) into your workflow code
- Build your own notification system to alert supervisors when intervention is needed
- Create interfaces for supervisors to respond to HITL requests
- Implement monitoring and logging for HITL interactions

This approach is ideal when you need to integrate with existing internal systems or require custom workflows that don't fit the managed HIS model.

Best practices

When you implement HITL, follow these best practices:

- Set appropriate timeouts based on session age. For example, login interventions might need longer timeouts than CAPTCHA resolutions.
- Implement robust error handling to manage timeout and rejection scenarios gracefully.
- Maintain comprehensive logs of all HITL interactions for audit and analysis purposes.

Additional resources

Visit the nova-act-human-intervention GitHub repository for CDK code and detailed setup instructions.

Tool use beyond the browser (Preview)

(Preview) Nova Act allows you to integrate external tools beyond the browser, such as an API call or database query, into workflows. This functionality enables developers to integrate remote MCP tools and agentic frameworks, such as Strands Agents, into their workflows.

Defining a local tool

The first step in the workflow is defining the tool. The tool definition must include all of the necessary context to guide the model on appropriate tool invocation.

To make a Python function available as a tool, you annotate it with the `@tool` decorator. This allows you to include the required context. To do so, define the following within the function's comments:

- a description of the tool
- its input parameters
- its return type

Below is an example of a tool defined to return a specific row from an Excel file:

```
@tool
def read_row_as_dict(file_path, row_number):
    """
    Reads a specific row from an Excel file and returns it as a dictionary
    where column headers are keys and row values are the corresponding
    dictionary values.

    Args:
        file_path (str): The path to the Excel file.
        row_number (int): The row number (1-based index) to retrieve.

    Returns:
        dict: A dictionary containing the data from the specified row.
    """
    # Read the Excel file using pandas
    df = pd.read_excel(file_path)
    # Check if the row_number is within the valid range
    if row_number < 1 or row_number > len(df):
        raise ValueError(f"Row number {row_number} is out of range. " f"The sheet has
{len(df)} rows.")

    # Get the row data as a dictionary
    row_data = df.iloc[row_number - 1].to_dict()

    return row_data
```

Using a tool

To use a tool with Nova Act, add it to the tools attribute of the Nova Act constructor. Nova Act will decide whether to call a tool given the request.

To define the tools available when initializing Nova Act, pass an array of tools, like below.

```
with NovaAct(
    starting_page=file_path,
    tools=[read_row_as_dict],
)
```

Note

For best performance, limit the number of tools you provide to the model in a single act command. For higher accuracy, you can be more specific in the prompt to let Nova Act know when you want it to invoke a specific tool. Nova Act performs best if you match the instructions to the tool description. Similar to browser commands, we recommend being prescriptive and succinct in what the agent should do and breaking down larger acts into smaller ones to improve reliability.

Returning tool results

When Nova Act determines a tool is needed, it will call the tool with the appropriate arguments. Here is an example of how tool results show in the Nova Act logs:

```
think("I need to read the data from row number 1 in the Excel file.");
tool({"name":"read_row_as_dict","input":{"file_path":[file_path],"row_number":1}});
```

```
Result for tool call 'read_row_as_dict': {'First Name': 'John', 'Last Name': 'Doe',
 'Email': 'jdoe@example.com'}
```

Using MCP with Nova Act

The Model Context Protocol (MCP) is an open standard that enables developers to build secure, two-way connections between their data sources and AI-powered tools. Instead of writing custom adapters for each API or service, you can run an MCP server and integrate its tools with Nova Act automatically through a client bridge. Once connected, Nova Act treats these tools like any other external integration: it decides when to call them, sends the required parameters, and incorporates the results into its response.

Below is an example using the AWS Documentation MCP Server. This example requires installing the Strands Agents library.

```
from mcp import StdioServerParameters, stdio_client
from strands.tools.mcp import MCPClient
with MCPClient(
    lambda: stdio_client(
        StdioServerParameters(
            command="uvx",
            args=["awslabs.aws-documentation-mcp-server@latest"]
        )
) as aws_docs_client:
```

Once the MCP is initialized, retrieve the list of available tools from the MCP server and pass them to NovaAct.

```
with NovaAct(
    starting_page=file_path,
    tools=aws_docs_client.list_tools_sync(),
)
```

Integrate other services and frameworks with Nova Act

Nova Act can run as a standalone agent or as a `tool` used by another agent. This guide shows both approaches: running Nova Act as an agent on Amazon Bedrock AgentCore, and as a `tool` for an agent with Strands Agents.

Contents

Integrating Nova Act with Bedrock AgentCore

Amazon Bedrock AgentCore offers purpose-built infrastructure to deploy and operate production AI agents at scale. AgentCore is comprised of several modules that can be used a la carte depending on your requirements. In this guide we show an example of using each integration with Nova Act.

AgentCore Runtime (ACR)

When you deploy a Nova Act workflow to ACR, ACR provisions an endpoint which you use to invoke the Nova Act workflow. Deploying Nova Act workflows on ACR has the added benefit of simple integrations with other AgentCore services. ACR is serverless and does not require you to maintain any infrastructure. For a step-by-step guide of deploying a Nova Act workflow to ACR, refer to this article.

AgentCore Identity (ACI)

ACI allows developers to provide identities for their agents, which can be used to obtain necessary access to resources and to isolate activity to specific agents. Using ACI with Nova Act enables you to securely grant permission to your workflow to access resources when needed to complete an action. A common use case for this is accessing log credentials to login to a web application. ACI also supports injecting user context from the caller which allows your Nova Act workflow to run using a user’s context.

In this example, we use a custom OAuth provider to generate access tokens for a user, and then have Nova Act use this access token to take action on behalf of the user.

First, we create a credential provider using the AWS CLI. Refer to the Bedrock AgentCore documentation for more details on how to create these. Below is a partial sample for a `CustomOauth2` provider:

```
aws bedrock-agentcore-control create-oauth2-credential-provider \
  --region us-east-1 \
  --name "custom-oauth-provider" \
  --credential-provider-vendor "CustomOauth2" \
  --oauth2-provider-config-input '{
  "customOauth2ProviderConfig": {
    "oauthDiscovery": {
       ...
    },
    "clientId": "string",
    "clientSecret": "string"
  }
  }'
```

AgentCore Runtime automatically handles all identity management. When your agent is deployed, the runtime provides the agent identity and user context from the caller, then injects credentials via the decorator `@requires_access_token`.

```
from bedrock_agentcore.identity.auth import requires_access_token
from nova_act_sdk import NovaAct

@requires_access_token(
    provider_name="custom-oauth-provider",
)
def automate_customer_outreach(*, access_token: str):
    with NovaAct() as browser:
        # Token automatically injected by AgentCore Runtime
        result = browser.act(f"""
        Use OAuth token {access_token} to:
        1. Log into the site
        2. Search for software engineers in Seattle
        3. Send personalized connection requests
        """)
        return result
```

AgentCore Browser Tool (ACBT)

ACBT provides serverless infrastructure for running a browser in the cloud. ACBT supports features such as live streaming the browser session from the AgentCore AWS Console, as well as the ability for someone to take over the browser for a period of time. Common examples of this include solving a CAPTCHA, or entering sensitive data, before returning control back to the Nova Act workflow. For an example of running Nova Act with ACBT, refer to this article.

AgentCore Observability (ACO)

ACO allows developers to instrument and emit OpenTelemetry Data from a Nova Act workflow. You can instrument the Nova Act SDK with OTEL for compatibility with the Observability dashboard.

To do this, follow these steps:

- Install the open telemetry library: `pip install aws-opentelemetry-distro`
- Import the library in the Nova Act code: `from opentelemetry import baggage, context`
- Obtain your session ID from your active Nova Act workflow, and pass this into your OTEL session context.

Here is an example of instrumenting a Nova Act workflow and writing the telemetry data to CloudWatch logs:

```
import uuid
import os
import boto3
import logging
import argparse
from opentelemetry import baggage, context
from nova_act import NovaAct

# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)


def setup_agentcore_observability(agent_id):
    """Setup AgentCore observability environment and create log groups"""
    agent_name = "nova-act-browser"
    log_group = "/aws/bedrock-agentcore/runtimes/nova-act-poc"

    # Create CloudWatch log group and stream
    logs_client = boto3.client('logs')
    try:
        logs_client.create_log_group(logGroupName=log_group)
        logger.info(f"Created log group: {log_group}")
    except logs_client.exceptions.ResourceAlreadyExistsException:
        logger.info(f"Log group already exists: {log_group}")

    # Create log stream
    log_stream = "runtime-logs"
    try:
        logs_client.create_log_stream(logGroupName=log_group, logStreamName=log_stream)
        logger.info(f"Created log stream: {log_stream}")
    except logs_client.exceptions.ResourceAlreadyExistsException:
        logger.info(f"Log stream already exists: {log_stream}")

    # Write AgentCore environment variables to .env file
    # Note that you can export these via the command line when running the script
    env_vars = {
        "AGENT_OBSERVABILITY_ENABLED": "true",
        "OTEL_PYTHON_DISTRO": "aws_distro",
        "OTEL_PYTHON_CONFIGURATOR": "aws_configurator",
        "OTEL_RESOURCE_ATTRIBUTES":
 f"service.name={agent_name},aws.log.group.names={log_group}",
        "OTEL_EXPORTER_OTLP_LOGS_HEADERS": f"x-aws-log-group={log_group},x-aws-log-
stream=runtime-logs,x-aws-metric-namespace=bedrock-agentcore",
        "OTEL_EXPORTER_OTLP_LOGS_PROTOCOL": "http/protobuf",
        "OTEL_LOGS_EXPORTER": "otlp",
        "OTEL_TRACES_EXPORTER": "otlp"
    }

    with open('.env', 'w') as f:
        for key, value in env_vars.items():
            f.write(f"{key}={value}\n")
    logger.info("Created .env file with AgentCore observability settings")

def set_session_context(session_id):
    """Set the session ID in OpenTelemetry baggage for trace correlation"""
    ctx = baggage.set_baggage("session.id", session_id)
    token = context.attach(ctx)
    logging.info(f"Session ID '{session_id}' attached to telemetry context")
    return token
```

```

def main():
    # Generate agent ID and session ID
    agent_id = str(uuid.uuid4())[:8]

    setup_agentcore_observability(agent_id)

    # Execute Nova Act - ADOT auto-instrumentation handles the rest
    client = NovaAct(starting_page="https://nova.amazon.com/act")
    client.start()

    # Set session context for trace correlation
    session_id = client.get_session_id()
    context_token = set_session_context(session_id)

    try:
        result = client.act("Click Learn More")
        logger.info(f"Agent ID: {agent_id}")
        logger.info(f"Session ID: {session_id}")
        logger.info(f"Success: {result}")
    finally:
        client.stop()
        # Detach context when done
        context.detach(context_token)
        logger.info(f"Session context for '{session_id}' detached")

if __name__ == "__main__":
    main()
```

AgentCore Gateway (ACG)

ACG provides an easy and secure way for developers to build, deploy, discover, and connect to tools at scale. Nova Act integrates with ACG in two ways.

- Nova Act workflows can connect to an existing Gateway for agentic tool use.
- Nova Act workflows themselves can be exposed on the Gateway as a tool for other agents to use. Refer to the ACG developer guide for help on getting started.

Strands Agents Integrations

Strands Agents is an open source framework for building AI agents with a broad set of tool support. Strands makes it simple to equip an agent with tools it can use. For example, you can equip an agent with a tool that enables it to search the web. In the example below, we configure Nova Act for a Strands agent to use it as a tool.

Note

When Nova Act is run as a tool, the run time (Nova Act SDK) is run on the same compute as the Strands agent. For example, if the Strands agent is running on AgentCore Runtime, the Nova Act SDK will also run there. If you want the browser that Nova Act uses to run on AgentCore Browser Tool, you define that behavior within the Nova Act tool definition.

Running a local Strands agent that uses a Nova Act tool

In this example, we run Strands Agents on our local machine. We assume that Strands has already been installed. The Nova Act SDK automatically installs Chromium the first time you run it, if Chrome is not already installed on your system. On some platforms where Chromium requires root installation, you may need to install it manually.

Start by installing `nova-act` and the Bedrock AgentCore libraries:

- `pip install nova-act, bedrock_agentcore`

Now let's create Nova Act as a tool for our Strands agent. The tool will take in a website url, an instruction, and an option to download content. Save the below in a folder named `tools` in a file named `web-browser-tool.py`. Note the description of the tool at the start of the `get_current_ticker_price`. This informs the Strands agent what this tool should be used for, arguments to pass in, and the return type.

```
from strands import Agent, tool
from nova_act import NovaAct

@tool
def browser_automation_tool(starting_url:str, instr: str, download: bool) -> str:
    """
    Automates tasks in a browser on the starting_url website, using the instructions
 provided.
    Multiple sessions can be run in parallel. The tool can do some basic reasoning of
 its own.

    Args:
        starting_url (str): The website url to perform actions on
        instr (str): the instruction in natural language to be sent to the browser for
 the task to be performed
        download (bool): Whether to download generated content from the page or not

    Returns:
        str: The result of the action performed.
    """

    with NovaAct(
        starting_page=starting_url
    ) as browser:
        try:
            result = browser.act(instr, max_steps=10)
            if download:
                with browser.page.expect_download() as download_info:
                    browser.act("click on the download button")
                    # Temp path for the download is available.
                    print(f"Downloaded file {download_info.value.path()}")
                    download_info.value.save_as("my_downloaded_file.png")
                    print("Image downloaded successfully.")
                    success = True
            return success

        except Exception as e:
            error_msg = f"Error processing instruction: {instr}. Error: {str(e)}"
            print(error_msg)
            return error_msg
```

To make this tool available to a Strands agent, import the tool, and then add it to the list of tools available to the agent.

```
from tools import browser_automation_tool

def generate_and_retrieve_image():
    agent = Agent(tools=[browser_automation_tool])
    agent("On nova.amazon.com, generate an image of a wise robot contemplating life,
 and download the image.")

def main():
    generate_and_retrieve_image()
```

Running a Strands agent on AgentCore that uses a Nova Act tool

Now let's run the same agent as above but instead of running on our local machine, this time let's run it on AgentCore Runtime with AgentCore BrowserTool. Refer to this guide for instructions on deploying a Strands agent on AgentCore, and this guide for using Nova Act with AgentCore Browser Tool.

Responsible use

Responsible AI

Nova Act is built following the Responsible AI policies documented in the Amazon Nova User Guide.

The AWS AI Service Card for Amazon Nova Act explains the use cases for which the Nova Act AWS service is intended, how machine learning (ML) is used by the service, and key considerations in the responsible design and use of the service.

Code examples

The nova-act-samples GitHub repository contains sample code and solutions demonstrating Nova Act capabilities for web automation and AI-powered browser interactions, including standalone scripts, and CDK deployment examples.

Security in Amazon Nova Act

Note

This security section applies to your deployment on the Nova Act AWS service as an AWS customer. Not all of this content in this section applies to Nova Act developer tools on nova.amazon.com/act.

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

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

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

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

Contents

Nova Act SDK security

Be aware that Nova Act may encounter commands in the content it observes on third party websites, including user-generated content on trusted websites such as social media posts, search results, forum comments, news articles, and document attachments. These unauthorized commands, known as prompt injections, may cause the model to make mistakes or act in a manner that differs from its instructions, such as ignoring your instructions, performing unauthorized actions, or exfiltrating sensitive data.

We have trained Nova Act to deflect these attacks but cannot guarantee all prompt injection attacks will be deflected. To reduce the risks associated with prompt injections, it is important to monitor Nova Act and review its actions, especially when processing untrusted user-contributed content. We also recommend that developers use the following approaches to further reduce their risk wherever possible and appropriate. See below for details.

1. Domain restriction – use SDK state guardrails to enforce an allow/block list of URLs.
2. Tool use restriction – to minimize attack surfaces, only register tools relevant for a given workflow.
3. Local file access restriction - restrict access to `file://` path access unless necessary for a specific workflow. This is blocked by default in the SDK. We recommend developers to allow this capability only for select file-paths to complete a given workflow as needed.

Nova Act SDK security options

The Nova Act SDK ships with secure default behaviors that should remain enabled unless your use case specifically requires otherwise. Disabling these defaults reduces the security posture of your system.

Allow navigation to local file:// URLs

To enable local file navigation, define one or more filepath patterns in `SecurityOptions.allowed_file_open_paths`

```
from nova_act import NovaAct, SecurityOptions

NovaAct(starting_page="file://home/nova-act/site/index.html",
 SecurityOptions(allowed_file_open_paths=['/home/nova-act/site/*']))
```

Allow file uploads

To allow the agent to upload files to websites, define one or more filepath patterns in `SecurityOptions.allowed_file_upload_paths`.

```
from nova_act import NovaAct, SecurityOptions
NovaAct(starting_page="https://example.com",
 SecurityOptions(allowed_file_upload_paths=['/home/nova-act/shared/*']))
```

Filepath structures

The filepath parameters support the following formats:

- `["/home/nova-act/shared/*"]` - Allow from specific directory
- `["/home/nova-act/shared/file.txt"]` - Allow a specific filepath
- `["*"]` - Enable for all paths
- `[]` - Disable the feature (Default)

State guardrails

State guardrails allow you to control which URLs the agent can visit during execution. You can provide a callback function that inspects the browser state after each observation and decides whether to allow or block continued execution. If blocked, act() will raise ActStateGuardrailError. This is useful for preventing the agent from navigating to unauthorized domains or sensitive pages.

```
from nova_act import NovaAct, GuardrailDecision, GuardrailInputState
from urllib.parse import urlparse
```

```
import fnmatch

def url_guardrail(state: GuardrailInputState) -> GuardrailDecision:
    hostname = urlparse(state.browser_url).hostname
    if not hostname:
        return GuardrailDecision.BLOCK

    # Example URL block-list
    blocked = ["*.blocked-domain.com", "*.another-blocked-domain.com"]
    if any(fnmatch.fnmatch(hostname, pattern) for pattern in blocked):
        return GuardrailDecision.BLOCK

    # Example URL allow-list
    allowed = ["allowed-domain.com", "*.another-allowed-domain.com"]
    if any(fnmatch.fnmatch(hostname, pattern) for pattern in allowed):
        return GuardrailDecision.PASS

    return GuardrailDecision.BLOCK

with NovaAct(starting_page="https://allowed-domain.com", state_guardrail=url_guardrail)
 as nova:
    # The following will be blocked if agent tries to visit a blocklisted domain or
 leave one of the allowlisted domains
    nova.act("Navigate to the homepage")
```

Data protection in Amazon Nova Act

Note

This data protection section applies to your deployment on the Nova Act AWS service as an AWS customer.

The AWS shared responsibility model applies to data protection in Nova Act. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ.

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

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

We strongly recommend you never put sensitive identifying information, such as your customers' account numbers, into free-form fields such as a **Name** field. This includes when you work with Amazon Nova Act or other AWS services using the console, API, AWS Command Line Interface, or AWS SDKs. Any data that you enter into Amazon Nova Act or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, don’t include credentials information in the URL to validate your request to that server.

Nova Act considerations

The following are special security considerations for Nova Act.

- We recommend you do not provide sensitive information within the act() statements, such as account passwords. Instead, such information should be passed directly to the browser using Playwright keyboard API calls. Note that if you use sensitive information through Playwright, the information could still be collected in screenshots by AWS if it appears unobstructed on the browser when Nova Act is engaged in completing an action.
- Nova Act accepts inputs from customers for tool definitions and service details, which can potentially contain personally identifiable information (PII).
- Usage metrics and logs are stored in the customer account’s CloudWatch.

Data encryption in Amazon Nova Act

Encryption at rest

Nova Act stores data at rest using Amazon DynamoDB and Amazon Simple Storage Service (Amazon S3). The data at rest is encrypted using AWS encryption solutions by default. Nova Act encrypts your data using AWS owned encryption keys from AWS Key Management Service. You do not need to take any action to protect the AWS managed keys that encrypt your data. For more information, see AWS owned keys in the AWS KMS Developer Guide.

Key considerations:

- Nova Act temporarily stores Agent Trajectory data, which includes the input prompt, screenshots, and agent response to maintain historical context while executing a workflow.
- If you wish to persist the Agent Trajectory Data indefinitely, you may opt into the service writing this data to an S3 bucket that you own and control. We strongly encourage you to enable encryption on this S3 bucket.
- The following data is not encrypted by default:
  - WorkflowDefinition Names
  - Workflow Run Ids

Encryption in transit

All communication between customers and the Nova Act AWS service, as well as between Nova Act and its downstream dependencies, is protected using TLS 1.2 or higher connections.

Key management

All AWS KMS keys are managed by the Nova Act service. At this time, there is no support for customer-managed keys (CMK), but is expected to be added in the subsequent release.

Internetwork traffic policy (VPC and PrivateLink)

PrivateLink is not yet supported by Nova Act, but this functionality is expected to be added in a subsequent release.

Identity and ccess management for Amazon Nova Act

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

Audience

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

- **Service user –** If you use the Amazon Nova Act service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon Nova Act features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon Nova Act, see Troubleshooting Amazon Nova Act identity and access.
- **Service administrator –** If you're in charge of Amazon Nova Act resources at your company, you probably have full access to Amazon Nova Act. It's your job to determine which Amazon Nova Act features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon Nova Act, see How Amazon Nova Act works with IAM.
- **IAM administrator –** If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon Nova Act. To view example Amazon Nova Act identity-based policies that you can use in IAM, see Amazon Nova Act identity-based policy examples.

Authenticating with identities

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

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities.

When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see Signature Version 4 signing process in the *AWS General Reference*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide* and Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account root user and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *Account Management Reference Guide*.

Federated identity

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

A federated identity is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An IAM group is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Creating a role for a third-party Identity Provider in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions.
  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an Amazon EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the Amazon EC2 instance. To assign an AWS role to an Amazon EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the Amazon EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

To learn whether to use IAM roles, see When to create an IAM role (instead of a user) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

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

Every IAM entity (user or role) starts with no permissions. By default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, role, or group. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource such as an Amazon S3 bucket. Service administrators can use these policies to define what actions a specified principal (account member, user, or role) can perform on that resource and under what conditions. Resource-based policies are inline policies. There are no managed resource-based policies.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – Set the maximum available permissions for resources in your accounts. For more information, see How RCPs work in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

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

How Amazon Nova Act works with IAM

How Amazon Nova Act works with IAM

Before you use IAM to manage access to Nova Act, learn what IAM features are available to use with Nova Act.

IAM Feature	Amazon Nova Act Support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys	Yes
ACLs	No
ABAC (tags in policies)	No
Temporary credentials	Yes
Principal permissions	Yes
Service roles	No
Service-linked roles	Yes

Identity-based policies for Nova Act

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Resource-based Policies

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

Amazon Nova Act does not support resource-based policies

Policy Actions for Nova Act

The `Action` element of an IAM identity-based policy describes the specific action or actions that will be allowed or denied by the policy. Policy actions usually have the same name as the associated AWS API operation. The action is used in a policy to grant permissions to perform the associated operation.

Policy actions in Amazon Nova Act use the following prefix before the action: `nova-act:`. For example, to grant someone permission to run an Amazon EC2 instance with the Amazon EC2 `RunInstances` API operation, you include the `ec2:RunInstances` action in their policy. Policy statements must include either an `Action` or `NotAction` element. Amazon Nova Act defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": [
      "ec2:action1",
      "ec2:action2"
]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "ec2:Describe*"
```

Policy Resources for Nova Act

The `Resource` element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. You specify a resource using an ARN or using the wildcard (*) to indicate that the statement applies to all resources.

The Amazon EC2 instance resource has the following ARN:

```
arn:${Partition}:ec2:${Region}:${Account}:instance/${InstanceId}
```

For more information about the format of ARNs, see Amazon Resource Names (ARNs) and AWS service Namespaces.

For example, to specify the `i-1234567890abcdef0` instance in your statement, use the following ARN:

```
"Resource": "arn:aws:ec2:us-east-1:123456789012:instance/i-1234567890abcdef0"
```

To specify all instances that belong to a specific account, use the wildcard (*):

```
"Resource": "arn:aws:ec2:us-east-1:123456789012:instance/*"
```

Some Amazon Nova Act actions, such as those for creating resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (*).

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

Many Amazon EC2 API actions involve multiple resources. For example, `AttachVolume` attaches an Amazon EBS volume to an instance, so an IAM user must have permissions to use the volume and the instance. To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": [
      "resource1",
      "resource2"
]
```

Policy condition keys for Nova Act

The Condition element (or Condition`*block*) lets you specify conditions in which a statement is in effect. The `Condition element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

Amazon Nova Act defines its own set of condition keys and also supports using some global condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

All Amazon EC2 actions support the `aws:RequestedRegion` and `ec2:Region` condition keys. For more information, see Example: Restricting access to a specific region.

Examples

To view examples of Amazon Nova Act identity-based policies, see Amazon Nova Act identity-based policy examples.

Access control lists (ACLs)

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

Amazon Nova Act does not support ACLs.

ABAC (tags in policies)

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes called tags.

Amazon Nova Act does not support ABAC.

Amazon Nova Act IAM roles

An IAM role is an entity within your AWS account that has specific permissions.

Using temporary credentials with Amazon Nova Act

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as AssumeRole or GetFederationToken.

Amazon Nova Act supports using temporary credentials.

Service-linked roles for Nova Act

Service-linked roles allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view but not edit the permissions for service-linked roles.

Amazon Nova Act supports service-linked roles. The service uses the NovaActServiceRolePolicy to publish operational metrics to CloudWatch. The service-linked role is automatically created when you start using Amazon Nova Act, and it uses the permissions defined in the NovaActServiceRolePolicy managed policy described in the AWS managed policies for Amazon Nova Act section.

Service roles for Nova Act

This feature allows a service to assume a service role on your behalf. This role allows the service to access resources in other services to complete an action on your behalf.

Amazon Nova Act does not support service roles.

Amazon Nova Act identity-based policy examples

By default, IAM users and roles don't have permission to create or modify Amazon Nova Act resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating policies on the JSON tab in the *IAM User Guide*.

Policy best practices

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

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

Using the Amazon Nova Act console

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

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

AWS managed policies for Amazon Nova Act

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

AWS addresses many common use cases by providing standalone IAM policies that are created and administered by AWS. These AWS managed policies grant necessary permissions for common use cases so that you can avoid having to investigate which permissions are needed. For more information, see AWS managed policies in the *IAM User Guide*.

AWS managed policy: NovaActServiceRolePolicy

This policy is attached to a service-linked role that allows the service to perform actions on your behalf. You cannot attach this policy to your users, groups, or roles.

This policy grants permissions that Amazon Nova Act to publish operational metrics to Amazon CloudWatch under the AWS/NovaAct namespace. These metrics provide visibility into agent performance and operational health.

Amazon Nova Act uses this AWS managed policy to publish metrics that help you monitor the service. The policy grants the `cloudwatch:PutMetricData` permission with a condition that restricts metric publication to the AWS/NovaAct namespace only, ensuring that the service can only publish its own operational metrics.

For the JSON policy document for NovaActServiceRolePolicy, see NovaActServiceRolePolicy in the *AWS Managed Policy Reference Guide*.

Customer managed policy: NovaActFullAccess

Note

Amazon Nova Act does not yet define a managed policy which grants access to the Nova Act service. You will need to define a customer-managed policy for this purpose.

For example:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "nova-act:*"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "nova-act.amazonaws.com"
                 }
                }
            }
        ]
    }
```

AWS managed policy updates

View details about updates to AWS managed policies for Amazon Nova Act since this service began tracking these changes.

Change	Description	Date
NovaActServiceRolePolicy	Amazon Nova Act added a new policy. This policy allows Amazon Nova Act to publish operational metrics to CloudWatch using the cloudwatch:PutMetricData action, scoped to the AWS/NovaAct namespace through a condition statement.	December 2, 2025
Amazon Nova Act started tracking changes	Amazon Nova Act started tracking changes for its AWS managed policies.	December 2, 2025

Troubleshooting Amazon Nova Act identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon Nova Act and IAM.

Topics

I am not authorized to perform an action in Amazon Nova Act

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about workflow runs but does not have `nova-act:ListWorkflowRuns` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform: nova-
act:ListWorkflowRuns on resource: my-workflow-definition
```

In this case, Mateo asks his administrator to update his policies to allow him to access the `my-workflow-definition` resource using the `nova-act:ListWorkflowRuns` action.

I want to allow people outside of my AWS account to access my Amazon Nova Act resources

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

To learn more, consult the following:

- To learn whether Amazon Nova Act supports these features, see How Amazon Nova Act works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing Access to Externally Authenticated Users (Identity Federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Incident response in Nova Act

Security is the highest priority at AWS. As part of the AWS Cloud shared responsibility model, AWS manages a data center, network, and software architecture that meets the requirements of the most security-sensitive organizations. AWS is responsible for any incident response with respect to the Amazon Nova Act service itself. Also, as an AWS customer, you share a responsibility for maintaining security in the cloud. This means that you control the security you choose to implement from the AWS tools and features you have access to. In addition, you're responsible for incident response on your side of the shared responsibility model.

By establishing a security baseline that meets the objectives for your applications running in the cloud, you're able to detect deviations that you can respond to. To help you understand the impact that incident response and your choices have on your corporate goals, we encourage you to review the following resources:

- AWS Security Incident Response Guide
- AWS Best Practices for Security
- Security Perspective of the AWS Cloud Adoption Framework (CAF) whitepaper

Compliance validation for Nova Act

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

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

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

Resilience in Amazon Nova Act

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

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

Infrastructure security in Nova Act

As a managed service, Amazon Nova Act is protected by the AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in **Security Pillar AWS Well‐Architected Framework**. You use AWS published API calls to access Amazon Nova Act through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal.

Configuration and vulnerability analysis in Nova Act

Configuration and IT controls are a shared responsibility between AWS and you, our customer. For more information, see the AWS shared responsibility model.

Cross-service confused deputy prevention for Nova Act

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account. For more information, see Cross-service confused deputy prevention in the IAM user gude.

Monitoring Amazon Nova Act

Monitoring

Monitoring is an important part of maintaining the reliability, availability, and performance of Nova Act and your other AWS solutions. When deployed to AWS, you can monitor Nova Act using Amazon CloudWatch, which collects raw data and processes it into readable, near real-time metrics. You can graph the metrics using the CloudWatch console. You can also set alarms that watch for certain thresholds and send notifications or take actions when values exceed those thresholds. For more information, visit What is Amazon CloudWatch in the *Amazon CloudWatch User Guide*.

The following table describes the runtime metrics provided by Nova Act.

Metric Name	Unit	Dimensions	Description
Invocations	Count	Workflow, ApiName	Number of API requests received for a workflow.
Latency	Milliseconds	Workflow, ApiName	End-to-end latency for the API call. You can query percentiles (p50, p90, p99) to understand latency distribut ion.
UserErrors	Count	Workflow, ApiName	Count of user-side errors, such as invalid parameters, bad workflow or session IDs, or IAM permission issues.
SystemErrors	Count	Workflow, ApiName	Count of service-side errors, including internal faults, dependenc y failures, and timeouts.
Throttles	Count	Workflow, ApiName	Number of requests rejected due to throttling or exceeding service quotas.

Metric dimensions

Amazon Nova Act metrics use the following dimensions:

- **Workflow** - The name of the workflow. This dimension allows you to filter metrics by specific workflows in your account.
- **ApiName** - The name of the API operation. This dimension allows you to filter metrics by specific API calls.

Viewing metrics in CloudWatch

You can view Amazon Nova Act metrics using the CloudWatch console, AWS Command Line Interface, or CloudWatch API.

To view metrics in the CloudWatch console:

- Open the CloudWatch console at console.aws.amazon.com/cloudwatch/.
- In the navigation pane, choose **Metrics**, then choose **All metrics**.
- Choose the **AWS/NovaAct** namespace.
- Select a metric dimension (for example, **By Workflow**) to view available metrics.
- Select the checkbox next to the metrics you want to view.

The metrics appear in the graph on the page. You can customize the time range, statistic, and period for the metrics.

Setting alarms

You can create CloudWatch alarms to monitor Nova Act metrics and receive notifications when metric values exceed thresholds you define. For example, you can create an alarm to notify you when:

- The number of system errors exceeds a threshold
- Workflow latency increases beyond acceptable levels
- Throttling occurs, indicating you may need to request quota increases

For information about creating alarms, visit docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/AlarmThatSendsEmail.html in the Amazon CloudWatch User Guide.

Monitoring best practices

Consider the following best practices when monitoring Nova Act:

- **Monitor error rates** - Track both UserErrors and SystemErrors metrics to identify issues with your workflows or service availability.
- **Set up latency alarms** - Create alarms for the Latency metric to detect performance degradation early.
- **Track throttling** - Monitor the Throttles metric to identify when you're approaching service quotas.
- **Use percentile statistics** - For the Latency metric, use p90 or p99 statistics to understand tail latency and ensure consistent performance.
- **Create dashboards** - Build CloudWatch dashboards that combine multiple metrics to get a comprehensive view of your workflow health.

Quotas for Nova Act

Your AWS account has default quotas, formerly referred to as limits, for each AWS service. You can request increases for some quotas, while other quotas cannot be increased. To request a quota increase, contact AWS support.

To maintain the performance of the service and to ensure appropriate usage of Nova Act, the default quotas assigned to an account might be updated depending on regional factors, payment history, fraudulent usage, and/or approval of a quota increase request.

To view service quotas for Nova Act, do one of the following:

- Follow the steps at Viewing service quotas and select **Amazon Nova Act** as the service.
- Refer to **Amazon Nova Act service quotas** in the AWS General Reference.

Adjustable Quotas

Control Plane API Quotas

The following table describes the Control Plane API quotas for Nova Act:

API	Default Limit	Scope	Adjustable
CreateWorkflowDefinition	100 TPS	Account	Yes
DeleteWorkflowDefinition	100 TPS	Account	Yes
DeleteWorkflowRun	100 TPS	Account	Yes
GetWorkflowDefinition	100 TPS	Account	Yes
ListWorkflowDefinitions	100 TPS	Account	Yes
GetWorkflowRun	100 TPS	Account	Yes

API	Default Limit	Scope	Adjustable
ListWorkflowRuns	100 TPS	Account	Yes
ListActs	100 TPS	Account	Yes
ListModels	100 TPS	Account	Yes
ListSessions	100 TPS	Account	Yes

Data Plane API Quotas

The following table describes the Data Plane API quotas for Nova Act:

API	Default Limit	Scope	Adjustable
CreateWorkflowRun	100 TPS	Account	Yes
CreateSession	100 TPS	Account	Yes
CreateAct	100 TPS	Account	Yes
InvokeActStep	5 TPS	Account	Yes
UpdateAct	100 TPS	Account	Yes
UpdateWorkflowRun	100 TPS	Account	Yes

Resource Count Quotas

The following table describes the Resource Count quotas for Nova Act:

Resource	Default Limit	Scope	Adjustable
Workflow Definitions	100K	Account	Yes

Non-Adjustable Quotas

The following table describes the non-adjustable quotas for Nova Act:

Limit	Limit	Scope	Adjustable
Steps per Act	200	Session	No
InvokeActStep Payload Size	5 mb	Account	No
Workflow Run Timeout	1 Week	Account	No
Act Timeout	24 hours	Account	No
Tool Spec Payload Size	350 kb	Account	No
Tools per Act	100	Account	No

Document history for the Amazon Nova Act User Guide

The following table describes the documentation releases for Amazon Nova Act.

Change	Description	Date
Initial release	Initial release of the Amazon Nova Act User Guide	12/2/2025