aws

User Guide

Amazon CodeGuru Profiler

Amazon CodeGuru Profiler: User Guide

Table of Contents

What is Amazon CodeGuru Profiler? 1
What can I do with CodeGuru Profiler? 1
What languages are supported by CodeGuru Profiler? 1
How do I get started with CodeGuru Profiler? 2
Setting up 3
Set up in the Lambda console 3
Step 1: Sign up for AWS 4
Step 2: Enable CodeGuru Profiler 4
Set up in the CodeGuru Profiler console 5
Step 1: Sign up for AWS 4
Step 2: Create a CodeGuru Profiler profiling group 4
Step 3: Set permissions 6
Step 4: Start CodeGuru Profiler in your application 9
Getting Started 11
Python sample application 11
Prerequisites 12
Step 1: Create a profiling group 12
Step 2: Set up the virtual environment 13
Step 3: Run the application 13
Step 4: Understanding the console 14
Cleanup 15
Java sample application 16
Option 1: Quick demo 16
Option 2: Complete demo 17
Integrating with JVM 22
Choosing the right integration option 22
Profiling your applications that run on AWS Lambda 23
All Java runtimes 24
Easier option for Java 8 on Amazon Linux 2 and Java 11 and Java 17 (Corretto) runtimes ... 26
Enabling the agent from the command line 27
Installation 27
Configuration 28
Supported runtime environments 29
Enabling the agent with code 34

Installation ... 34
Configuration ... 35
Supported languages ... 38
Java ... 38
Scala ... 39
Kotlin ... 39
Groovy ... 39
Jython ... 40
JRuby ... 40
Clojure ... 41
Integrating with Python ... 42
Profiling your applications that run on AWS Lambda ... 42
Apply the CodeGuru Profiler function decorator to your handler function ... 43
Use AWS Lambda layers ... 44
Enabling the agent with code ... 45
Supported web components ... 38
Django ... 47
Flask ... 47
WSGI servers ... 48
Enabling the agent from the command line ... 49
Profiling Distributed systems ... 50
Enabling logs ... 50
Working with Amazon EventBridge ... 51
Working with unsupported AWS Regions ... 54
Enabling the agent with code ... 54
Java ... 54
Python ... 55
Profiling applications that run on AWS Lambda ... 55
Java ... 55
Python ... 56
Working with profiling groups ... 57
Creating a profiling group ... 57
Deleting a profiling group ... 57
Working with visualizations ... 59
Accessing visualizations ... 59
Types of visualizations ... 60

Overview visualizations ... 60
Hotspots visualizations ... 64
Inspect visualizations ... 64
Exploring visualization data ... 64
Choosing my code in visualizations ... 65
Pausing over a frame ... 66
Zooming in on a frame ... 66
Resetting zoom in a visualization ... 66
Inspecting a frame ... 67
Understanding the dollar estimate of the CPU cost for frames ... 67
Filtering visualization data ... 67
Selecting and coloring thread states ... 68
Hiding a frame ... 69
Selecting a custom time range ... 70
Understanding the summary page ... 70
Profiling group status ... 71
CPU summary ... 71
Latency summary ... 72
Heap usage ... 72
Anomalies ... 72
Recommendations ... 73
Understanding the heap summary ... 73
Total capacity ... 73
Used space ... 73
Heap summary table ... 74
Comparing two time ranges ... 74
Understanding the comparison ... 75
Working with anomalies and recommendation reports ... 77
Viewing reports ... 77
Understanding performance improvement recommendations ... 78
Understanding anomaly reports ... 78
Tagging profiling groups ... 80
Add a tag to a profiling group ... 80
Add a tag to a profiling group ... 81
View tags for a profiling group ... 81
View tags for a profiling group ... 81

Edit tags for a profiling group ... 81
Edit a tag for a profiling group ... 82
Remove a tag from a profiling group ... 82
Remove a tag from a profiling group ... 83
Security ... 84
Data protection ... 84
Captured data ... 85
Data encryption ... 86
Data retention ... 86
Internetwork Traffic Privacy ... 86
Identity and access management ... 86
Audience ... 87
Authenticating with identities ... 87
Managing access using policies ... 88
Overview of managing access ... 90
Using identity-based policies ... 94
Resource-based policies ... 103
CodeGuru Profiler permissions reference ... 105
AWS managed policies ... 109
Troubleshooting ... 113
Using service-linked roles ... 116
Using tags to control access to Amazon CodeGuru Profiler resources ... 120
Compliance Validation ... 122
Using CodeGuru Profiler with VPC Endpoints ... 122
Creating Amazon VPC Endpoints for CodeGuru Profiler ... 123
Infrastructure security ... 124
Logging and monitoring ... 125
Logging CodeGuru Profiler API calls with CloudTrail ... 125
Amazon CodeGuru Profiler information in CloudTrail ... 125
Understanding Amazon CodeGuru Profiler log file entries ... 126
Monitoring CodeGuru Profiler with CloudWatch ... 129
Monitoring profiling groups with CloudWatch metrics ... 130
Monitoring profiling groups with CloudWatch alarms ... 130
Troubleshooting ... 132
Profile is missing expected methods ... 132
CodeGuru Profiler doesn't appear in application logs ... 132

An exception says the profiling group doesn't exist 133
Getting a 403 Forbidden error that the agent doesn't have permission to submit data 134
No data in the console 134
Profile shows only a few frames 134
No Lambda data 134
I have errors in my Lambda function log. 134
I received a `ValidationException` error in the agent. 135
Heap summary data 135
Garbage collection 135
Advanced troubleshooting 136
Quotas 137
Profiling groups 137
Document history 138
AWS Glossary 141

What is Amazon CodeGuru Profiler?

Amazon CodeGuru Profiler collects runtime performance data from your live applications, and provides recommendations that can help you fine-tune your application performance. Using machine learning algorithms, CodeGuru Profiler can help you find your most expensive lines of code and suggest ways you can improve efficiency and remove CPU bottlenecks.

CodeGuru Profiler provides different visualizations of profiling data to help you identify what code is running on the CPU, see how much time is consumed, and suggest ways to reduce CPU utilization.

What can I do with CodeGuru Profiler?

Use CodeGuru Profiler to help profile your applications in the cloud from a single, centralized dashboard.

Specifically, you can do the following:

- Troubleshoot latency and CPU utilization issues in your application.
- Learn where you could reduce the infrastructure costs of running your application.
- Identify application performance issues.
- Understand your application's heap utilization over time.

What languages are supported by CodeGuru Profiler?

CodeGuru Profiler currently supports applications written in all Java virtual machine (JVM) languages and runtimes and Python 3.6 or later. The following table explains which features of CodeGuru Profiler are supported by which language.

Feature	Java/JVM	Python
CPU profiling	Yes	Yes
Support for AWS Lambda and other AWS compute platforms	Yes	Yes

Feature	Java/JVM	Python
Anomalies and recommend ation reports	Yes	Yes
Colored thread states	Yes	Yes
Heap summary visualization	Yes	No

How do I get started with CodeGuru Profiler?

1. Prepare to use CodeGuru Profiler by following the steps in Setting up CodeGuru Profiler.
2. Learn how to use recommendation reports by following the steps in Working with anomalies and recommendation reports.
3. Graphically explore your application data by following the steps in Working with visualizations.

Setting up CodeGuru Profiler

An Amazon CodeGuru Profiler profiling group is a group of applications for which data is meant to be aggregated and analyzed together. To create a profiling group, sign in to the AWS Management Console and set permissions for the CodeGuru Profiler profiling agent.

The profiling agent collects runtime data from your applications. Data that the agent collects is analyzed to provide flame graphs and hourly reports with recommendations for how you can optimize your applications.

You can create a profiling group using your own application or the demo application. For more information about using the demo application, see Getting started with CodeGuru Profiler.

Before you can start using CodeGuru Profiler, you must complete setup. If your application runs on AWS Lambda, then you can enable profiling from the Lambda console. If your application runs on a platform other than Lambda, then you can complete the setup process in the CodeGuru Profiler console.

Topics

- Set up in the Lambda console
- Set up in the CodeGuru Profiler console

Set up in the Lambda console

You can use the following method to create a profiling group with your Lambda function. This method automatically creates a profiling group when a profile is available to submit. This method is applicable for runtimes Python 3.8, Python 3.9, Java 8 on Amazon Linux 2 and Java 11 and Java 17 (Corretto). Alternatively, you can create a profiling group by following the instructions in Setting up in the CodeGuru Profiler console.

If you want to integrate with Lambda for an application with a different runtime, see Profiling your Java applications that run on AWS Lambda or Profiling your Python applications that run on AWS Lambda..

Step 1: Sign up for AWS

When you sign up for Amazon Web Services (AWS), your AWS account is automatically signed up for all services in AWS, including CodeGuru Profiler. You're charged only for the services that you use.

If you have an AWS account already, move on to the next task. If you don't have an AWS account, use the following procedure to create one.

To create an AWS account

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

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

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

Step 2: Enable CodeGuru Profiler

A profiling group can profile one or more applications. Data is aggregated and displayed based on the whole profiling group.

For example, if you have a collection of microservices that handle restaurant recommendations, you can collect profile data and identify performance issues across all these microservices in a single profiling group named "Restaurant-Recommendations".

To enable profiling from the Lambda console

1. Sign in to the AWS Management Console, and then open the Lambda console.
2. Choose your Lambda function. In the **Configuration** tab, choose **Monitoring and operation tools**. Choose **Edit**.
3. Enable **Code profiling** in the **Amazon CodeGuru Profiler** section. This creates a profiling group when a profile is available to submit.

> **Note**
>
> If you want to delete your profiling group, visit the CodeGuru Profiler console. If you disable **Code profiling** in the Lambda console, your profiling group still exists.
> If the execution role of your Lambda function doesn't have the required CodeGuru Profiler permissions such as AmazonCodeGuruProfilerAgentAccess or your function doesn't have the required environment variables, the Lambda console attempts to add them.

Set up in the CodeGuru Profiler console

Step 1: Sign up for AWS

When you sign up for Amazon Web Services (AWS), your AWS account is automatically signed up for all services in AWS, including CodeGuru Profiler. You're charged only for the services that you use.

If you have an AWS account already, move on to the next task. If you don't have an AWS account, use the following procedure to create one.

To create an AWS account

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

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

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

Step 2: Create a CodeGuru Profiler profiling group

A profiling group can profile one or more applications. Data is aggregated and displayed based on the whole profiling group.

For example, if you have a collection of microservices that handle restaurant recommendations, you can collect profile data and identify performance issues across all these microservices in a single profiling group named "Restaurant-Recommendations".

To create a profiling group from the CodeGuru Profiler console

1. Sign in to the AWS Management Console, and then open the CodeGuru Profiler console at https://console.aws.amazon.com/codeguru/profiler.
2. In the navigation pane on the left, choose **Profiler**, and then choose **Profiling groups**.
3. On the **Profiling groups** page, choose **Create profiling group**.
4. Provide a **Name** for the new profiling group. Choose the compute platform that on which your applications are running. If your applications run on AWS Lambda, choose the **AWS Lambda** option. Choose **Other** if your applications run on a compute platform other than AWS Lambda, such as Amazon EC2, on-premises servers, or a different platform.
5. Choose **Create profiling group**.

Step 3: Set permissions

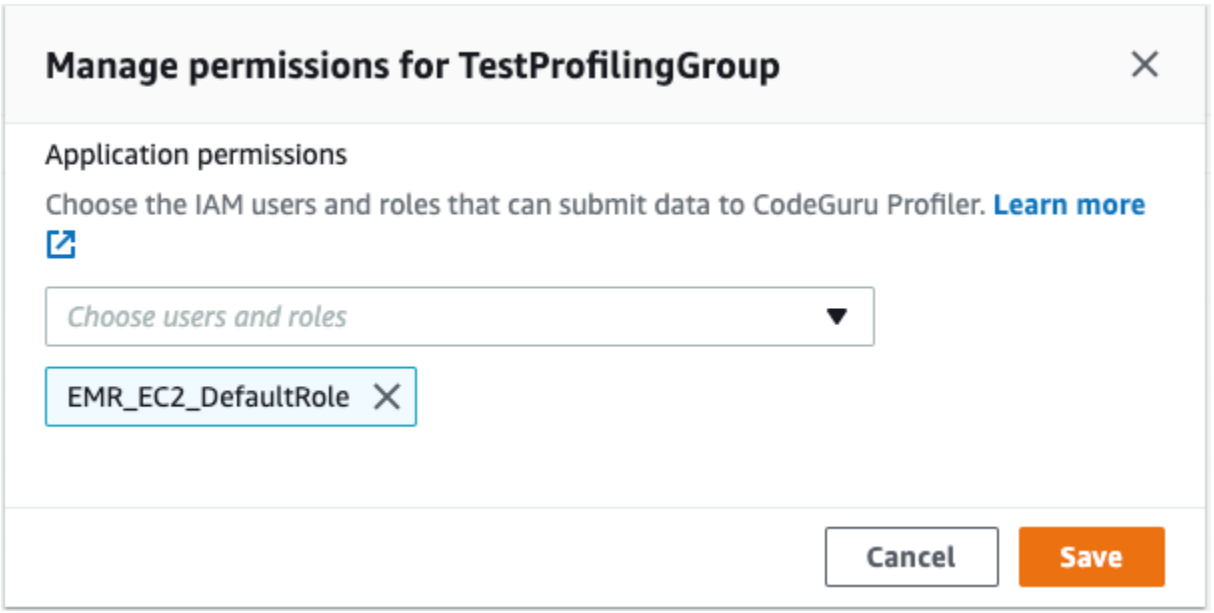
The CodeGuru Profiler profiling agent needs permissions to write data to the profiling group.

You can add the necessary permissions by adding the following policy.

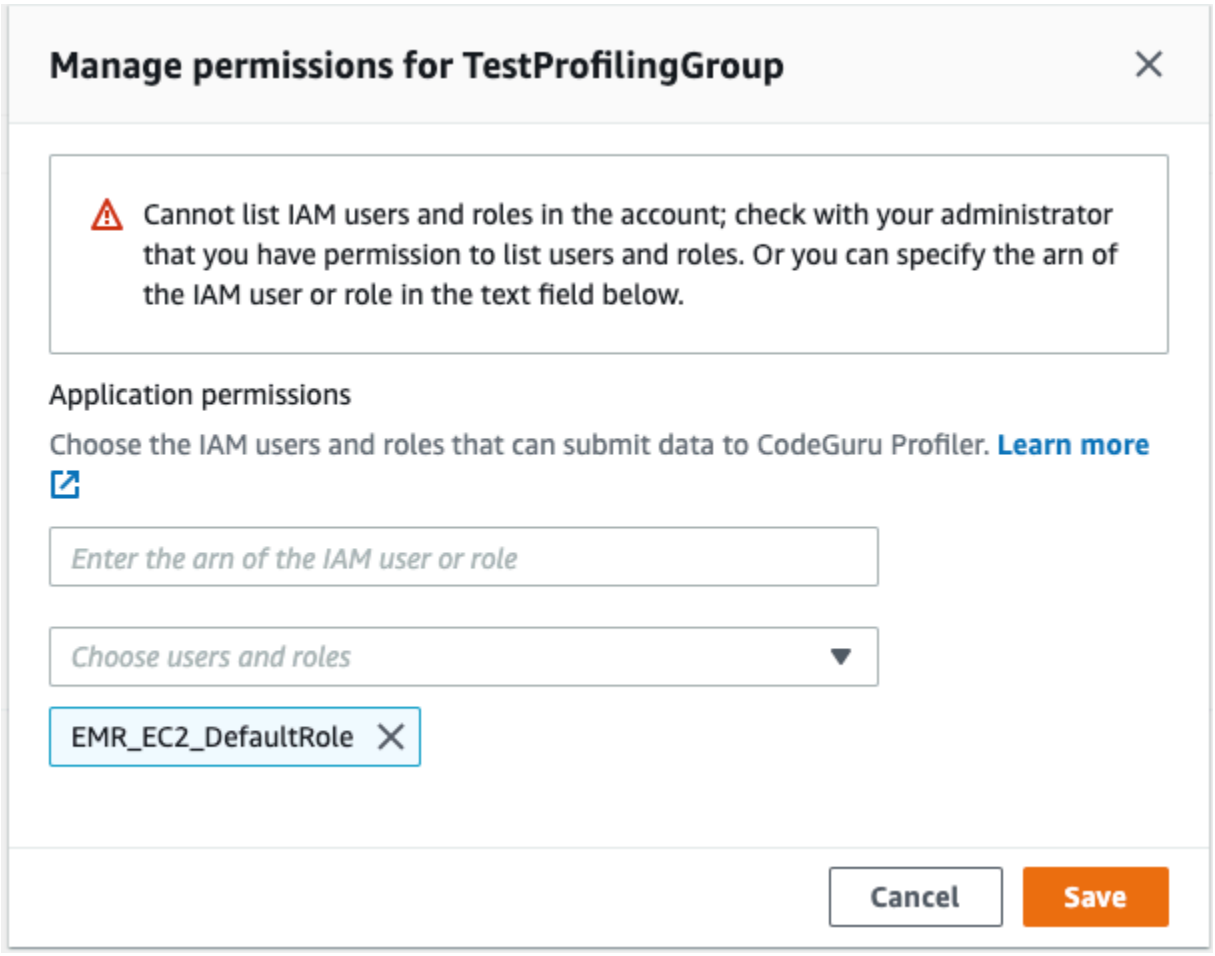
```
arn:aws:iam::aws:policy/AmazonCodeGuruProfilerAgentAccess
```

To set permissions for the new CodeGuru Profiler agent:

1. Start by choosing **Give access to users and roles**. Choose the IAM users and roles that can submit profiling data and configure the agent.
2. If your applications run on AWS Lambda, choose the role that your AWS Lambda function uses.
3. After you grant permissions for a user or role, you don't need to attach IAM policies for agent permissions.



Use `IAM:ListUsers` and `IAM:ListRoles` permissions to see your users and roles. Otherwise, you can add a user or Amazon Resource Name (ARN) role. You'll see the following message.



Alternatively, you can add a policy like the following to the role that your application uses. For more information about roles, see Modifying a role.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codeguru-profiler:ConfigureAgent",
                "codeguru-profiler:PostAgentProfile"
            ],
            "Resource": "arn:aws:codeguru-
profiler:<region>:<accountID>:profilingGroup/<profilingGroupName>"
        }
    ]
}
```

If your application is running in a Region that CodeGuru Profiler doesn't support and if you have the appropriate permissions, you can submit profiling data to one of the supported Regions. For more information about using CodeGuru Profiler in a Region it doesn't support, see Working with unsupported Regions.

Step 4: Start CodeGuru Profiler in your application

Run your application with the profiling agent

Run your application with the CodeGuru Profiler profiling agent. You can use CodeGuru Profiler on your Python or Java virtual machine (JVM)-based applications.

For your JVM-based application, you can either start the agent as a JVM agent, or start it manually with a code change in your application. To start profiling your application, see Integrating with JVM.

To start profiling your Python application, see Integrating with Python.

Enable data collection for the heap summary visualization

The CodeGuru Profiler heap summary shows your application's heap usage over time. This feature is available for JVM applications. For more information on the heap summary, see Understanding the heap summary.

Heap summary collection requires Java agent version 1.2.4 or greater. Opt in to heap summary data collection by completing the onboarding method used to enable the agent. The following are the onboarding methods from which you can choose.

Command line (-javaagent)

Add `heapSummaryEnabled:true`. The following example shows how to enable heap summary collection.

```
-javaagent:/path/to/codeguru-profiler-java-agent-
standalone-1.2.4.jar="profilingGroupName:myProfilingGroup,heapSummaryEnabled:true"
```

Update your code

Set `.withHeapSummary` to `true`. The following is an example of what your code might look like.

```
class MyClass {
```

```
    public static void main(String[] args) {
        Profiler.builder()
            .profilingGroupName("MyProfilingGroup")
            .withHeapSummary(true) // optional - to start without heap profiling set to
false or remove line
            .build()
            .start();
        ...
    }
```

Environment variables

Set `AWS_CODEGURU_PROFILER_HEAP_SUMMARY_ENABLED` to `true`.

Getting started with CodeGuru Profiler

The tutorials in this section are provided to help you start using Amazon CodeGuru Profiler. They show you how to set up your first profiling group and understand the console.

After you set up CodeGuru Profiler, you can use the demo application to learn about profiling groups. For more information about using the demo application, see the Amazon CodeGuru Profiler Demo Application. It will help to familiarize you with profiling groups and the ways you can visualize application data in CodeGuru Profiler. It's also a good place to start if you're new to profiling.

Topics

- Python sample application
- Java sample application

Python sample application

In this tutorial, you'll walk through the complete set up necessary to run Amazon CodeGuru Profiler within a sample application. You'll then be able to view the profiling group's resulting runtime data.

To view the sample application, navigate to the CodeGuru Profiler Python demo application. CodeGuru Profiler runs inside the sample application and collects and reports profiling data about the application, which can be viewed from the AWS console.

This tutorial's sample application does some basic image processing, with some CPU-heavy operations alongside some IO-heavy operations. It uses an Amazon Simple Storage Service bucket for cloud storage of images and an Amazon Simple Queue Service queue to order the names of images to be processed. It consists chiefly of two components which run in parallel, the task publisher and the image processor:

`TaskPublisher` checks the S3 bucket for available images, and submits the name of some of these images to the SQS queue.

`ImageProcessor` polls SQS for names of images to process. Processing an image involves downloading it from S3, applying some image transforms (e.g. converting to monochrome), and uploading the result back to S3.

Prerequisites

For this tutorial, you will need:

- The latest version of the AWS Command Line Interface. For more information, see Installing, updating, and uninstalling the AWS Command Line Interface.
- Python 3.6 or later.
- The sample application cloned to your desired directory.

Step 1: Create a profiling group

Set up the required components by running the following AWS commands in your terminal.

1. Configure the AWS CLI.

   When prompted, specify the AWS access key and AWS secret access key of the IAM user that you will use with CodeGuru Profiler.

   When prompted for the default Region name, specify the Region where you will create the pipeline, such as `us-west-2`.

   When prompted for the default output format, specify the .json.

   ```
   aws configure
   ```

2. Create a profiling group in CodeGuru Profiler, named `PythonDemoApplication`.

   ```
   aws codeguruprofiler create-profiling-group --profiling-group-name
    PythonDemoApplication
   ```

3. Create an Amazon SQS queue.

   ```
   aws sqs create-queue --queue-name DemoApplicationQueue
   ```

4. Create an Amazon S3 bucket. Replace the YOUR-BUCKET with your desired bucket name.

   ```
   aws s3 mb s3://python-demo-application-test-bucket-YOUR-BUCKET
   ```

5. Provision an IAM role. For information, see Creating IAM roles or use one that is associated with your AWS account.

Attach the `AmazonSQSFullAccess, AmazonS3FullAccess,` and `AmazonCodeGuruProfilerFullAccess` AWS managed policies to the IAM role.

6. Export the Amazon SQS URL, Amazon S3 bucket name, and the target Region.

   Remember to replace YOUR-AWS-REGION, YOUR-ACCOUNT-ID, and YOUR-BUCKET . YOUR-AWS-REGION should be the Region you specified in the AWS CLI configuration. YOUR-ACCOUNT-ID should be your AWS account ID. YOUR-BUCKET should be the bucket name you specified during Amazon S3 bucket creation.

   ```
   export DEMO_APP_SQS_URL=https://sqs.YOUR-AWS-REGION.amazonaws.com/YOUR-ACCOUNT-ID/
   DemoApplicationQueue
   export DEMO_APP_BUCKET_NAME=python-demo-application-test-bucket-YOUR-BUCKET
   export AWS_CODEGURU_TARGET_REGION=YOUR-AWS-REGION
   ```

Step 2: Set up the virtual environment

Create the virtual environment and install necessary resources by running the following commands in your terminal.

1. Create and activate the virtual environment.

   ```
   python3 -m venv ./venv
   source venv/bin/activate
   ```

2. Install boto3 and skimage. Installing scikit-image with Python 3.9 may cause failures. For more information, see Installing schikit-learn via pip.

   ```
   pip3 install boto3 scikit-image
   ```

3. Install the CodeGuru Profiler agent.

   ```
   pip3 install codeguru_profiler_agent
   ```

Step 3: Run the application

Run the following command in your terminal to start the application. Then, verify that the application has begun profiling.

1. Run the sample application with the CodeGuru Profiler Python Agent.

   Note that when running the demo application for the first time, it is expected to see error messages such as No messages exist in SQS queue at the moment, retry later. and Failed to list images in demo-application-test-bucket-1092734-YOUR-BUCKET under input-images/ printing to the terminal. Our demo application will handle image upload and SQS message publication after a few seconds.

   ```
   python3 -m codeguru_profiler_agent -p PythonDemoApplication
    aws_python_sample_application/main.py
   ```

2. Verify that your profiling group is running.

   Navigate to the CodeGuru Profiler console.

   Choose **Profiling groups**.

   The PythonDemoApplication group should have status "Pending". You may need to wait a few minutes for this to update. After running the demo for approximately 15 to 20 minutes, the group should have status "Profiling", and you can view runtime data.

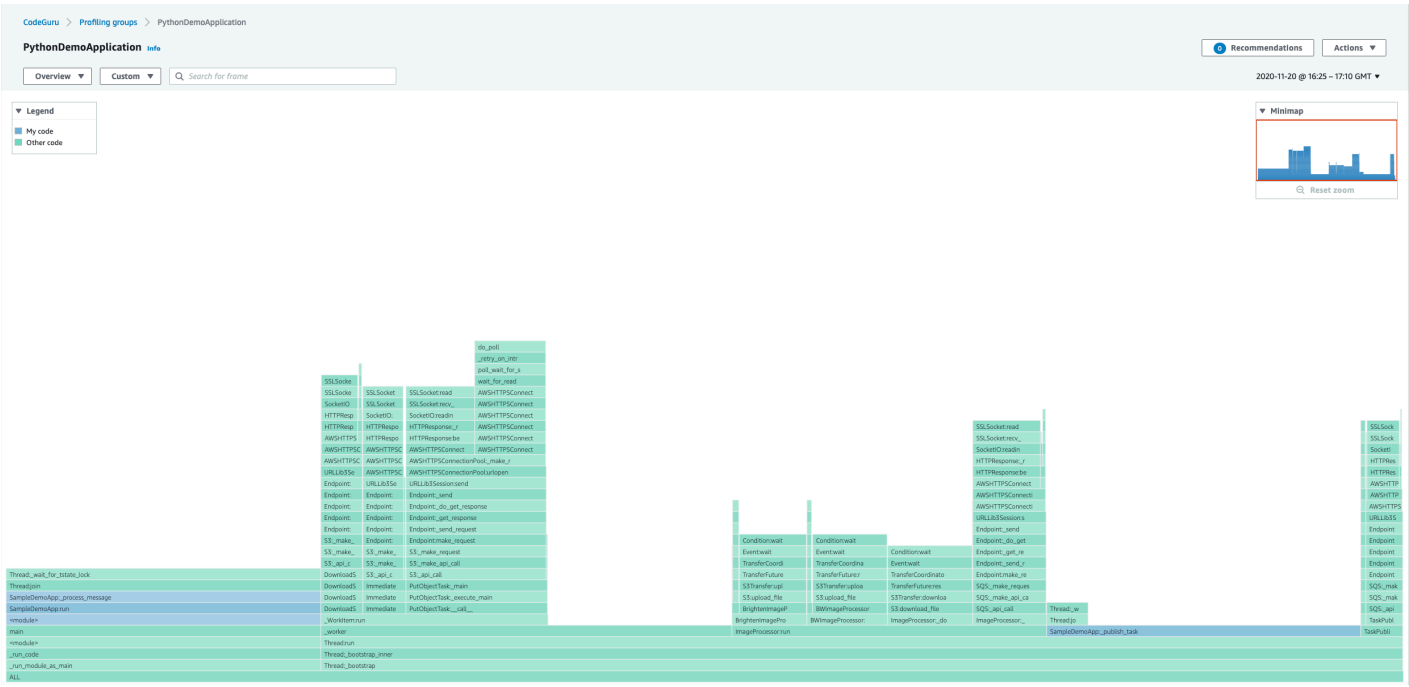
Step 4: Understanding the console

The CodeGuru Profiler console is where you can view the data that Profiler has gathered from running the application.

1. Navigate to the CodeGuru Profiler console.
2. Choose **Profiling groups**.
3. Choose the `PythonDemoApplication` group.

The following sections are displayed, along with options for other visualizations or a full recommendations report.

- **Profiling group status** displays the status of the profiling group and metrics from data collected in the past 12 hours.
- **CPU summary** displays the amount of system CPU capacity that the application consumes. You can choose **Visualize CPU** to view the flame graph.



- **Latency summary** displays the amount of time the application's threads spend in the Blocked, Waiting, and Timed Waiting thread states.
- **Heap usage** displays how much of your application's maximum heap capacity is consumed by the application.
- **Anomalies** display any deviations from trends that CodeGuru Profiler detects.
- **Recommendations** display suggestions to optimize the application.

Cleanup

Upon completion of this tutorial, clean up the resources created.

Delete the profiling groups by following the procedure in Deleting a profiling group.

Delete the Amazon S3 bucket, replacing `YOUR-BUCKET` with your specified bucket name. Note that all objects stored in the bucket will be deleted as well.

```
aws s3 rb s3://python-demo-application-test-bucket-YOUR-BUCKET --force
```

Delete the Amazon SQS queue. Note that this will delete the queue itself, not just the messages in the queue.

```
aws sqs delete-queue
```

Java sample application

This tutorial walks through the complete setup necessary to run Amazon CodeGuru Profiler within two sample applications: one that features issues to generate CodeGuru Profiler recommendations and one that does not. You can then view the resulting CodeGuru Profiler runtime data and recommendations.

To view the sample application, navigate to the CodeGuru Profiler demo application. CodeGuru Profiler runs inside the sample application and collects and reports profiling data about the application, which can be viewed from the CodeGuru Profiler console.

This tutorial's sample application does some basic image processing, with some CPU-heavy operations alongside some IO-heavy operations. It uses an Amazon Simple Storage Service bucket for cloud storage of images and an Amazon Simple Queue Service queue to order the names of images to be processed. It consists chiefly of two components which run in parallel, the task publisher and the image processor:

`TaskPublisher` checks the S3 bucket for available images, and submits the name of some of these images to the SQS queue.

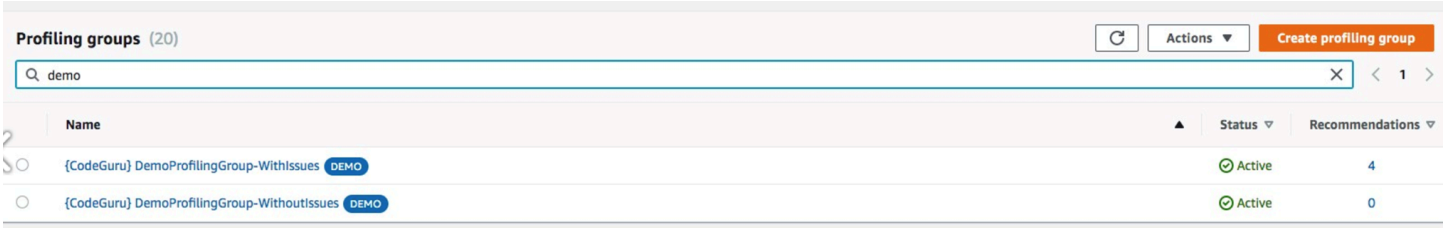
`ImageProcessor` polls SQS for names of images to process. Processing an image involves downloading it from S3, applying some image transforms (e.g. converting to monochrome), and uploading the result back to S3.

When running `DemoApplication-WithIssues`, you see messages such as `Expensive` exception and `Pointless` work being logged. This is expected, and you can see how these issues appear on the CodeGuru Profiler console.

Option 1: Quick demo

The results of this application are available to review within the CodeGuru console. There is no need to run the code if you just want to see the CodeGuru Profiler output.

1. Sign in to the AWS Management Console, and then open the CodeGuru Profiler console at https://console.aws.amazon.com/codeguru/profiler.
2. In the navigation pane on the left, choose **Profiler**, and then choose **Profiling groups**.
3. On the **Profiling groups** page, choose `{CodeGuru} DemoProfilingGroup-WithIssues` and `{CodeGuru} DemoProfilingGroup-WithoutIssues` to view runtime performance data and recommendations. To better understand the console, see step 4.



Option 2: Complete demo

Prerequisites

For this tutorial, you will need:

- The latest version of the AWS Command Line Interface. For more information, see Installing, updating, and uninstalling the AWS Command Line Interface.
- Maven to build and run the code. For information on installing Maven, see Welcome to Apache Maven.
- The sample application cloned to your desired directory.

Step 1: Create AWS resources

Set up the required components by running the following AWS commands in your terminal.

1. Configure the AWS CLI.

   When prompted, specify the AWS access key and AWS secret access key of the IAM user that you will use with CodeGuru Profiler.

   When prompted for the default Region name, specify the Region where you will create the pipeline, such as `us-west-2`.

   When prompted for the default output format, specify the .json.

   ```
   aws configure
   ```

2. Create two profiling groups in CodeGuru Profiler, named `DemoApplication-WithIssues` and `DemoApplication-WithoutIssues`.

   ```
   aws codeguruprofiler create-profiling-group --profiling-group-name DemoApplication-
   WithIssues
   ```

```
aws codeguruprofiler create-profiling-group --profiling-group-name DemoApplication-
WithoutIssues
```

3. Create an Amazon SQS queue.

```
aws sqs create-queue --queue-name DemoApplicationQueue
```

4. Create an Amazon S3 bucket. Replace the YOUR-BUCKET with your desired bucket name.

```
aws s3 mb s3://demo-application-test-bucket-YOUR-BUCKET
```

5. Provision an IAM role. For information, see Creating IAM roles or use one that is associated with your AWS account.

   Attach the `AmazonSQSFullAccess, AmazonS3FullAccess,` and `AmazonCodeGuruProfilerFullAccess` AWS managed policies to the IAM role.

Step 2: Run a sample application

Follow these steps to run one of the sample applications. You select which sample application to run in step 2. To run the other sample application, repeat the following steps, selecting the other sample application in step 2.

1. Export the Amazon SQS URL, the S3; bucket name, and the target Region.

   Remember to replace `YOUR-AWS-REGION`, `YOUR-ACCOUNT-ID`, and `YOUR-BUCKET`. `YOUR-AWS-REGION` should be the Region you specified in the AWS CLI configuration. `YOUR-ACCOUNT-ID` should be your AWS account ID. `YOUR-BUCKET` should be the bucket name you specified during Amazon S3 bucket creation.

```
export DEMO_APP_SQS_URL=https://sqs.YOUR-AWS-REGION.amazonaws.com/YOUR-ACCOUNT-ID/
DemoApplicationQueue
export DEMO_APP_BUCKET_NAME=demo-application-test-bucket-1092734-YOUR-BUCKET
export AWS_CODEGURU_TARGET_REGION=YOUR-AWS-REGION
```

2. Export the CodeGuru Profiler profiling group

   To run the sample application with issues, run the following command.

```
export AWS_CODEGURU_PROFILER_GROUP_NAME=DemoApplication-WithIssues
```

To run the sample application without issues, run the following command.

```
export AWS_CODEGURU_PROFILER_GROUP_NAME=DemoApplication-WithoutIssues
```

3. Generate the sample application .jar file.

```
1. mvn clean install
```

4. Run the demo.

```
java -javaagent:codeguru-profiler-java-agent-standalone-1.2.4.jar \
  -jar target/DemoApplication-1.0-jar-with-dependencies.jar with-issues
```

5. Verify that your profiling group is running.

   Navigate to the CodeGuru Profiler console.

   Choose **Profiling groups**.

   The `DemoApplication-WithIssues` profiling group should have the status **Pending**. You may need to wait a few minutes for this to update. After running the demo for around 15 minutes, the group should have the status **Profiling**.

   Run the demo for a few hours to generate ample data for recommendations.

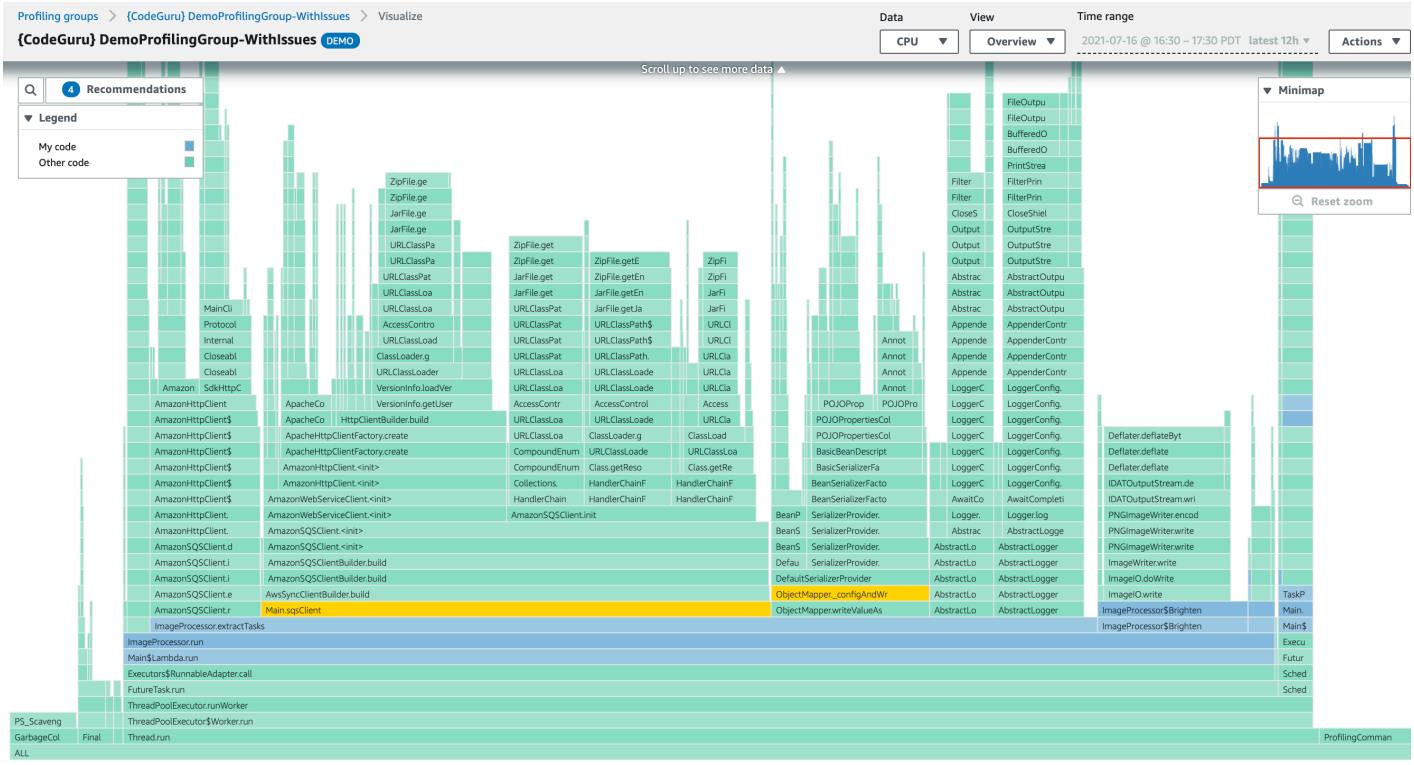
Step 3: Understanding the console

The CodeGuru Profiler console is where you can view the data that Profiler has gathered from running the application.

1. Navigate to the CodeGuru Profiler console.
2. Choose **Profiling groups**.
3. Choose the `DemoApplication-WithIssues` or the `DemoApplication-WithoutIssues` profiling group

The following sections are displayed, along with options for other visualizations or a full recommendations report.

- **Profiling group status** displays the status of the profiling group and metrics from data collected in the past 12 hours.
- **CPU summary** displays the amount of system CPU capacity that the application consumes. You can choose **Visualize CPU** to view the flame graph.



- **Latency summary** displays the amount of time the application's threads spend in the **Blocked**, **Waiting**, and **Timed Waiting** thread states.
- **Heap usage** displays how much of your application's maximum heap capacity is consumed by the application.
- **Anomalies** display any deviations from trends that CodeGuru Profiler detects.
- **Recommendations** display suggestions to optimize the application.

The `DemoApplication-WithIssues` generates recommendations, while the `DemoApplication-WithoutIssues` does not. These recommendations have to do with the purposefully inefficient lines of code that unnecessarily recreate loggers, SDK service clients, and Jackson Object Mappers. You can choose **View report** next to the **Recommendations** section for more details on these inefficiencies, including their location, estimated costs, and suggested resolution steps.

The average heap usage in the `DemoApplication-WithIssues` is significantly higher, indicating that the application requires significantly more memory.

Cleanup

Upon completion of this tutorial, clean up the resources created.

Delete the profiling groups by following the procedure in Deleting a profiling group.

Delete the Amazon S3 bucket, replacing `YOUR-BUCKET` with your specified bucket name. Note that all objects stored in the bucket will be deleted as well.

```
aws s3 rb s3://demo-application-test-bucket-1092734-YOUR-BUCKET --force
```

Delete the Amazon SQS queue. Note that this will delete the queue itself, not just the messages in the queue.

```
aws sqs delete-queue
```

Integrating with your JVM-based application

To start profiling your application, enable the CodeGuru Profiler agent to be loaded and started when your JVM-based application starts. After the agent starts, it automatically sends profiles to CodeGuru Profiler service. You'll see continuous updates and recommendations.

The following sections explain which environments and languages are supported by CodeGuru Profiler.

Topics

- Choosing the right integration option
- Profiling your applications that run on AWS Lambda
- Enabling the agent from the command line
- Enabling the agent with code

Choosing the right integration option

You can load the CodeGuru Profiler agent into your JVM-based application in two ways:

1. **Command line** – Use the `-javaagent` command line option when starting your application.
2. **Code** – Add the CodeGuru Profiler agent into your application code.

The same functionality is available in either option. Choosing the right option for your situation depends on the following:

To quickly start profiling your existing JVM-based application, the command line option might be best because it doesn't require recompiling your application.

For more control over when to start profiling, or in rare cases where you need to provide a custom authentication provider, you might want to choose the code option.

The following table helps summarize these options.

Option	Command line	Code
Profile existing application	Yes	No (requires re-compile)

Option	Command line	Code
Custom authentication provider	No	Yes
Control when profiling starts	No (profiling begins at startup)	Yes

You can always choose a different option later. All of the profiling data is stored in the CodeGuru Profiler service, and is available even when switching the CodeGuru Profiler agent.

Profiling your applications that run on AWS Lambda

To start CodeGuru Profiler in your application running on AWS Lambda, you can either update your Lambda function configuration or modify your application code. The former option is available only for Java 8 on Amazon Linux 2 and Java 11 and Java 17 (Corretto) runtimes, while the latter is available for all Java runtimes.

If you enabled profiling in the Lambda console, you don't have to complete the procedure outlined in the following sections. To learn more about enabling profiling from the Lambda console, see Set up in the Lambda console.

> **Note**
>
> You can profile your Lambda functions running in Java if they are called often enough for CodeGuru Profiler to gather enough samples. CodeGuru Profiler collects data once per second, aggregated into 5-minute sampling buckets. For Lambda functions running for fewer than 5 minutes, your application must run multiple times so CodeGuru Profiler can collect enough data. If it runs too infrequently, CodeGuru Profiler can't generate enough data to provide recommendations and flame graphs. For long-running Lambda applications, processing can take up to 15 minutes to display graphs and information. If you are running your application in shorter durations, processing takes longer to display information.

Topics

- All Java runtimes

- Easier option for Java 8 on Amazon Linux 2 and Java 11 and Java 17 (Corretto) runtimes

All Java runtimes

If you're profiling applications that run on AWS Lambda, add the following environment variables to your Lambda function.

- `AWS_CODEGURU_PROFILER_GROUP_ARN` – Identifies the profiling group ARN.
- `AWS_CODEGURU_PROFILER_ENABLED` – Enables profiling when set to `TRUE`. To disable profiling, set this variable to `FALSE`. Default value is `TRUE`.

For information about setting environment variables in the Lambda console, see Using AWS Lambda environment variables.

Add a dependency to the CodeGuru Profiler profiling agent library. You can do this manually by adding a dependency in your Maven or Gradle configuration files. For more information about adding dependencies, see Enabling the agent with code.

Make code changes to start profiling your AWS Lambda functions

If you have been using handlers provided by AWS Lambda, then you can alter your code to use handlers provided by CodeGuru to enable profiling. For information about your Lambda function's header, see AWS Lambda function handler in Java.

> **Note**
>
> No need to change your Lambda configuration! The handler function should still be `handleRequest`. This function is implemented by the CodeGuru class and calls the `requestHandler` after setting up the profiler.

If your Lambda function uses Lambda's `RequestHandler`, you can replace it with CodeGuru Profiler's `RequestHandlerWithProfiling` to enable profiling by default. `RequestHandlerWithProfiling` is a generic type that takes two parameters: the input type and the output type. Both types must be objects. When you use `RequestHandlerWithProfiling`, the Java runtime deserializes the event into an object with the input type, and serializes the output into text. Use this interface when the built-in serialization works with your input and output types. The following example provides a sample code snippet to enable profiling by default.

```
package example;

import java.util.Map;

import com.amazonaws.services.lambda.runtime.Context;

import software.amazon.codeguruprofilerjavaagent.RequestHandlerWithProfiling;

public class Handler extends RequestHandlerWithProfiling<Map<String, String>, String> {

    @Override
    public String requestHandler(Map<String, String> input, Context context) {
        // Your function code here
    }
}
```

If you are using `RequestStreamHandler`, then you can replace it with CodeGuru Profiler's `RequestStreamHandlerWithProfiling`. To use your own serialization, implement the `RequestStreamHandlerWithProfiling` interface, with which Lambda passes your handler an input stream and output stream. The handler reads bytes from the input stream, writes to the output stream, and returns void.

```
package example;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import com.amazonaws.services.lambda.runtime.Context;

import software.amazon.codeguruprofilerjavaagent.RequestStreamHandlerWithProfiling;

public class StreamHandler extends RequestStreamHandlerWithProfiling {

    @Override
    public void requestHandler(InputStream input, OutputStream output, Context context)
 throws IOException {
        // Your function code here
    }
}
```

If you don't use handlers provided by AWS Lambda, then add the following code to start profiling your AWS Lambda functions. Wrap your AWS Lambda function's logic inside a utility from the CodeGuru Profiler profiling agent.

```
package example;

import com.amazonaws.services.lambda.runtime.Context;

import software.amazon.codeguruprofilerjavaagent.LambdaProfiler;

public class MyHandler {

    // This is the handler function we use in the lambda
    public Output handleRequest(Input input, Context context) {
        return LambdaProfiler.profile(input, context, this::myHandlerFunction);
    }

    private Output myHandlerFunction(Input input, Context context) {
        // your function code here
    }
}
```

Your Lambda function runs the way it typically does, while the CodeGuru Profiler profiling agent runs in parallel. After running for 5 minutes, the agent submits your first profile. Processing can take up to 15 minutes.

Easier option for Java 8 on Amazon Linux 2 and Java 11 and Java 17 (Corretto) runtimes

You can enable CodeGuru Profiler from the AWS console by setting environment variables and updating configuration for your AWS Lambda function. This method works for Java 8 on Amazon Linux 2 and Java 11 and Java 17 (Corretto) runtimes.

If you're profiling applications that run on Lambda, set the following environment variables to your Lambda function.

- `AWS_CODEGURU_PROFILER_GROUP_NAME` – Identifies the profiling group name.
- `AWS_CODEGURU_PROFILER_TARGET_REGION` – Identifies the target region of the profiling group.

- `AWS_CODEGURU_PROFILER_HEAP_SUMMARY_ENABLED` – Optional. Set this variable to `true` to enable heap summary. The default is `false`.
- `JAVA_TOOL_OPTIONS` – Set this variable to `-javaagent:/opt/codeguru-profiler-java-agent-standalone.jar`.

For information about setting environment variables in the AWS Lambda console, see Using AWS Lambda environment variables.

Add a layer to your Lambda function using the following layer ARN. For more information on Lambda layers, see AWS Lambda Layers.

```
arn:aws:lambda:LAMBDA-FUNCTION-REGION-
CODE:157417159150:layer:AWSCodeGuruProfilerJavaAgentLayer:11
```

For example, if your Lambda function is in Region `us-east-1`, then the ARN would be the following.

```
arn:aws:lambda:us-east-1:157417159150:layer:AWSCodeGuruProfilerJavaAgentLayer:11
```

The CodeGuru Profiler heap summary is an optional feature that shows your application's heap usage over time. For more information on the heap summary, see Understanding the heap summary.

Your Lambda function runs normally with the CodeGuru Profiler agent running in parallel. The agent submits your first profile after running for a total of 5 minutes. Processing can take up to 15 minutes.

Enabling the agent from the command line

The command line option for integrating the CodeGuru Profiler agent is the easiest way to start profiling your application, because it doesn't require recompiling and redeploying your application. Add the appropriate command line options to your JVM-based runtime environment and you're ready to go.

Installation

Download the Amazon CodeGuru Profiler agent .jar file.

Save this to a location that is accessible from your JVM-based application.

Configuration

The only required configuration option to start the CodeGuru Profiler agent is the profiling group name. You can find this in the **Settings** section of your profiling group on the CodeGuru Profiler console.

You can use the credential path parameter to have the agent use credentials that are different from the default credentials. The path must point to a valid AWS credentials file. For more information about credentials, see Configuration and credential file settings.

The CodeGuru Profiler heap summary shows your application's heap usage over time. For more information on the heap summary, see Understanding the heap summary.

Opt in to heap summary data collection by adding `heapSummaryEnabled:true`. The following example shows how to enable heap summary collection.

```
-javaagent:/path/to/codeguru-profiler-java-agent-
standalone-1.2.4.jar="profilingGroupName:myProfilingGroup,heapSummaryEnabled:true"
```

You can specify these options as an environment variable or as a command line option.

Option	Environment variable	Command line option
Profiling group name (required)	`AWS_CODEGURU_PROFILER_GROUP_NAME`	`profilingGroupName`
Credential path	`AWS_CODEGURU_PROFILER_CREDENTIAL_PATH`	`credentialPath`
Region	`AWS_CODEGURU_PROFILER_TARGET_REGION`	`region`
Heap summary data collection	`AWS_CODEGURU_PROFILER_HEAP_SUMMARY_ENABLED`	`heapSummaryEnabled`

Your startup script using environment variables might look like the following.

```
#!/bin/bash
```

```
export AWS_CODEGURU_PROFILER_GROUP_NAME=MyProfilingGroup
export AWS_CODEGURU_PROFILER_TARGET_REGION=us-west-2

java -javaagent:/path/to/codeguru-profiler-java-agent-standalone-1.2.4.jar -jar
 MyApplication.jar
```

Alternatively, you can specify the configuration options by using the command line directly, as follows.

```
java -javaagent:/path/to/codeguru-profiler-java-agent-
standalone-1.2.4.jar=profilingGroupName:MyProfilingGroup,region:us-west-2 -jar
 MyApplication.jar
```

The argument string can contain multiple parameters. Separate parameters with a comma (`,`). Each parameter is a key-value pair.

> **Note**
>
> Your command must either be on one continuous line or a line-continuation option appropriate for your command shell.

Supported runtime environments

Most JVM-based application runtime environments support a mechanism to specify and customize JVM startup parameters to include the CodeGuru Profiler agent in the runtime startup. This section summarizes some of the popular runtime environments that we have verified to support this option.

All the examples assume that you have set the profiling group name with an environment variable: `export AWS_CODEGURU_PROFILER_GROUP_NAME=MyProfilingGroupName`.

Topics

- Java
- Scala
- Jython
- ColdFusion
- Geronimo

- SOLR
- Tomcat
- Glassfish
- Grails
- Jetty
- Play
- Resin
- Spring Boot
- Tanuki Wrapper
- Websphere Liberty Profile
- Spark
- Other runtime environments

Java

If you start your application using the `java` command, you can enable the CodeGuru Profiler agent in your application by adding the following `-javaagent` command line option.

```
java -javaagent:/path/to/codeguru-profiler-java-agent-standalone-1.2.4.jar -jar
 MyApplication.jar
```

Scala

If you start your application using the `scala` command, you can enable the CodeGuru Profiler agent in your application by adding the following `-J-javaagent` command line option.

```
scala -J-javaagent:/path/to/codeguru-profiler-java-agent-standalone-1.2.4.jar -jar
 MyScalaApplication.jar
```

Jython

If you start your application using the `Jython` command, you can enable the CodeGuru Profiler agent in your application by adding the following `-J-javaagent` command line option.

```
jython -J-javaagent:/path/to/codeguru-profiler-java-agent-standalone-1.2.4.jar -jar
 MyJythonApplication.jar
```

ColdFusion

Enable profiling for ColdFusion applications by adding the `-javaagent` option to the JVM parameters in the administrator console.

1. Navigate to your ColdFusion administrator console.
2. From the left menu, choose **SERVER SETTINGS**.
3. From the top bar, choose **Java and JVM**.
4. In the **JVM Arguments** field, add the following `-javaagent` argument.

   ```
   -javaagent:/path/to/codeguru-profiler-java-agent-standalone-1.2.4.jar
   ```

5. Choose **Submit changes**, then restart your ColdFusion server.

Geronimo

Add the CodeGuru Profiler agent to the Geronimo startup options by adding the `-javaagent` command line option to the `JAVA_OPTS` environment variable before starting your Geronimo instance.

```
export JAVA_OPTS="$JAVA_OPTS -javaagent:/path/to/codeguru-profiler-java-agent-
standalone-1.2.4.jar"
geronimo run
```

SOLR

Add the `-javaagent` command line option to the `SOLR_OPTS` variable in your SOLR startup configuration script, `/path/to/solr/bin/solr.in.sh`, by appending the following lines to it and adjusting them to your environment.

```
SOLR_OPTS="$SOLR_OPTS -javaagent:/path/to/codeguru-profiler-java-agent-
standalone-1.2.4.jar"
```

Tomcat

Add the `-javaagent` command line option to the `JAVA_HOME` environment variable in Tomcat's startup script, `/path/to/tomcat/bin/catalina.sh`.

```
JAVA_OPTS="$JAVA_OPTS -javaagent:/path/to/codeguru-profiler-java-agent-
standalone-1.2.4.jar"
```

Glassfish

1. Add <jvm-options>-javaagent:~/codeguru-profiler-java-agent-standalone-1.2.4.jar</jvm-options> below the java-config tag.
2. Start your domain, ./bin/asadmin start-domain domain1.

If you have JDK version 1.8 or later and are running Glassfish version 5.0 or later, you receive the following error.

```
java.lang.NoSuchMethodError:
sun.security.ssl.Handshaker.receiveChangeCipherSpec()
```

Grails

1. Add the following to /appName/build.groovy.

   ```
   tasks.withType(JavaExec) { jvmArgs "-javaagent:/path/to/codeguru-profiler-java-
   agent-standalone-1.2.4.jar" }
   ```

2. Start the **grails run-app** application.

Jetty

- Append -javaagent:~/codeguru-profiler-java-agent-standalone-1.2.4.jar to the startup script.

Play

- Append the following to your startup script, and then run the following.

  ```
  ./sbt -J-javaagent:~/codeguru-profiler-java-agent-standalone-1.2.4.jar "run 8080"
  ```

Resin

- Add the following to your configuration file.

```
<server-default><jvm-arg>-javaagent:~/codeguru-profiler-java-agent-
standalone-1.2.4.jar</jvm-arg>
```

Spring Boot

- Run the server with the `javaagent`.

```
java -javaagent:~/codeguru-profiler-java-agent-standalone-1.2.4.jar -jar
 demo-0.0.1-SNAPSHOT.jar
```

Tanuki Wrapper

- Add the following code to `wrapper.conf`.

```
<NON_DUPLICATE_NUMBER_IN_ADDITIONAL_PARAM_LIST>=-javaagent:~/codeguru-profiler-
java-agent-standalone-1.2.4.jar
```

Websphere Liberty Profile

- Append the following path to `jvm.options`.

```
-javaagent:~/codeguru-profiler-java-agent-standalone-1.2.4.jar
```

Spark

There is a Spark plugin to profile with CodeGuru Profiler. See A new Spark plugin for CPU and memory profiling.

CodeGuru Profiler supports Spark, but does not have `-javaagent` support. The agent is part of your .jar package file when you use CodeGuru Profiler in Spark. It doesn't matter if the agent is the worker or the primary as long as your code takes care of starting and stopping the agent when a job begins and ends. If a job is shorter than one minute, the agent won't report recommendations.

To provide enough samples per worker, run the agent on long-running jobs. see Enabling the agent with code

Other runtime environments

You can start any Java-based application by using the `-javaagent` command line option. If your runtime environment or hosting environment uses Java, consult your documentation to see how to customize the startup parameters for Java.

Enabling the agent with code

You can enable the Amazon CodeGuru Profiler agent in your application by adding code inside the startup routine of your application.

In addition to adding code, you also need to add a dependency to the agent library in your build steps. For this you can use a package manager such as Maven or Gradle.

Installation

To include the agent in your application, you need to tell your build system how to access the agent library. You can do this manually by adding a dependency in your Maven or Gradle configuration files.

Maven

To add a dependency to the agent, add the following sections to your pom.xml file. if you already have a `repositories` or `dependencies` element in your POM, add the individual `repositories` or `dependencies` elements inside the existing outer elements.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/
maven-4.0.0.xsd">
...
    <repositories>
        <repository>
            <id>codeguru-profiler</id>
            <name>codeguru-profiler</name>
            <url>https://d1osg35nybn3tt.cloudfront.net</url>
        </repository>
    </repositories>
```

```
    ...
    <dependencies>
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>codeguru-profiler-java-agent</artifactId>
            <version>1.2.4</version>
        </dependency>
    </dependencies>
...
</project>
```

For more information about configuring repositories in Maven, see Setting up Multiple Repositories in the Maven documentation.

Gradle

To add a dependency to the agent, add the following sections to your Gradle file. If you already have a `repositories` or `dependencies` element in your Gradle file, add the individual subelements into the existing outer elements.

```
repositories {
    maven {
        url = uri("https://d1osg35nybn3tt.cloudfront.net")
    }
}
dependencies {
    implementation("com.amazonaws:codeguru-profiler-java-agent:1.2.4")
}
```

For more information about creating a custom Gradle repository, see Declaring a custom repository by URL. For examples, see examples 18 and 19 in Supported repository transport protocols.

Configuration

You can configure the agent by using explicit API calls to the `Profiler.Builder` class. The following table shows the available options.

The profiling group name is required to start the CodeGuru Profiler agent. The CodeGuru Profiler heap summary shows your application's heap usage over time. For more information on the heap summary, see Understanding the heap summary.

Important

It is not recommended to enable heap summary data collection in your production environments, as it might increase latency in your application.

Type	API call
Profiling group name (required)	`.profilingGroupName(String)`
AWS Credentials Provider	`.awsCredentialsProvider(AwsCredentialsProvider)`
Region	`.awsRegionToReportTo(Region)`
Heap summary data collection (optional)	`.withHeapSummary(Boolean)`

The following is an example of command line API calls.

```
Profiler.builder()
    .profilingGroupName(“MyProfilingGroup”)
    .withHeapSummary(true) // optional - to start without heap profiling, set to false
 or remove line

    .build()
    .start();
```

We recommend that you configure and start the agent inside the startup or `main` function. The following example shows how to add the configuration to the `main` function.

```
import software.amazon.codeguruprofilerjavaagent.Profiler;

class MyApplication {
    public static void main(String[] args) {
        Profiler.builder()
            .profilingGroupName("MyProfilingGroup")
            .withHeapSummary(true)
            .build()
            .start();
```

```
        ...
    }
}
```

If you don't have access to a startup or `main` function, you can add a static initializer to your `main` class to configure and start the agent. This configures and starts the agent during the first time your application class is used inside the application container, as shown in the following example.

```
import software.amazon.codeguruprofilerjavaagent.Profiler;

class MyClass {

    static {
        Profiler.builder()
            .profilingGroupName("MyProfilingGroup")
            .build()
            .start();
    }
    ...
}
```

When your application is running, data is available in the CodeGuru Profiler console. To view your profiling data, choose **Profiler** in the navigation pane, choose **Profiling groups**, and then select your profiling group.

After your application has run for more than 15 minutes, data is available for you to visualize. For example, you can use an **Overview** visualization to identify code paths that are executed frequently. For more information about visualizations, see Working with visualizations.

When your application has run for an hour, the first **Recommendations** report is available. After the first report, new reports are generated hourly. For more information, see Working with anomalies and recommendation reports.

Note

If you don't want to use the default credentials to run the profiler, you can provide custom credentials by using following code. For more information about custom credentials, see Supplying and Retrieving AWS Credentials.

```
public static void main(String[] args) {
```

```
    Profiler.builder()
         .profilingGroupName("MyProfilingGroup")
         .awsCredentialsProvider(myAwsCredentialsProvider).build().start();
}
```

Supported languages

The following topics provide code that you can add to your application to enable the Amazon CodeGuru Profiler agent.

- Java
- Scala
- Kotlin
- Groovy
- Jython
- JRuby
- Clojure

Java

You can add support for the CodeGuru Profiler agent into your Java application by adding the following lines into your startup or `main` function.

```
import software.amazon.codeguruprofilerjavaagent.Profiler;

class MyClass {
    public static void main(String[] args) {
        Profiler.builder()
            .profilingGroupName("MyProfilingGroup")
            .build()
            .start();
        ...
    }
}
```

You need to add a dependency to the agent .jar file.

Scala

You can add support for the CodeGuru Profiler agent into your Scala application by adding the following lines into your startup or `main` function.

```
import software.amazon.codeguruprofilerjavaagent.Profiler

object MyObject {
    def main(args: Array[String]) = {
        Profiler.builder()
            .profilingGroupName("MyProfilingGroup")
            .build()
            .start()
        ...
    }
}
```

you need to add a dependency to the agent .jar file.

Kotlin

You can add support for the CodeGuru Profiler agent into your Kotlin application by adding the following lines into your startup or `main` function.

```
import software.amazon.codeguruprofilerjavaagent.Profiler

fun main() {
    Profiler.builder()
        .profilingGroupName("MyProfilingGroup")
        .build()
        .start()
    ...
}
```

You need to add a dependency to the agent .jar file.

Groovy

You can add support for the CodeGuru Profiler agent into your Groovy application by adding the following lines into your startup or `main` function.

```
import software.amazon.codeguruprofilerjavaagent.Profiler

Profiler.builder()
    .profilingGroupName("MyProfilingGroup")
    .build()
    .start()

...
```

You need to add a dependency to the agent .jar file.

Jython

You can add support for the CodeGuru Profiler agent into your Jython application by adding the following lines into your startup or `main` function.

```
import sys
sys.path.append("/path/to/codeguru-profiler-java-agent-1.2.4.jar")
from software.amazon.codeguruprofilerjavaagent import Profiler

Profiler.builder()
    .profilingGroupName("MyProfilingGroup")
    .build()
    .start()
...
```

You need to add a dependency to the agent .jar file.

JRuby

You can add support for the CodeGuru Profiler agent into your JRuby application by adding the following lines into your startup or `main` function.

```
Java::SoftwareAmazonCodeguruprofilerjavaagent::Profiler
    .builder
    .profiling_group_name("MyProfilingGroup")
    .aws_credentials_provider(myAwsCredentialsProvider) # optional
    .build
    .start
...
```

You need to add a dependency to the agent .jar file.

Clojure

You can add support for the CodeGuru Profiler agent into your Clojure application by adding the following lines into your startup or `main` function.

```
(-> (software.amazon.codeguruprofilerjavaagent.Profiler/builder)
    (.profilingGroupName "MyProfilingGroup")
    (.awsCredentialsProvider myAwsCredentialsProvider) ; optional
    (.build)
    (.start))
...
```

You need to add a dependency to the agent .jar file.

Integrating with Python

You can use Amazon CodeGuru Profiler to profile your Python application. Before you begin profiling your Python application, make sure your application is running on Python 3.6 or later.

Topics

- Profiling your applications that run on AWS Lambda
- Enabling the agent with code
- Enabling the agent from the command line
- Profiling Distributed systems
- Enabling logs

Profiling your applications that run on AWS Lambda

CodeGuru Profiler integration for AWS Lambda is currently available for applications that run on Python 3.7 up to Python 3.9. To start CodeGuru Profiler in your application running on Lambda, you can either apply the CodeGuru Profiler function decorator to your handler function, update your Lambda function configuration by adding layers, or enable profiling in the Lambda console.

If you enabled profiling in the Lambda console, you don't have to complete the procedure outlined in the following sections. To learn more about enabling profiling from the Lambda console, see Set up in the Lambda console.

> **Note**
>
> You can profile your Lambda functions running in Python if they are called often enough for CodeGuru Profiler to gather enough samples. CodeGuru Profiler collects data once per second, aggregated into 5-minute sampling buckets. For Lambda functions running for fewer than 5 minutes, your application must run multiple times so CodeGuru Profiler can collect enough data. If it runs too infrequently, CodeGuru Profiler can't generate enough data to provide recommendations and flame graphs. For long-running Lambda applications, processing can take up to 15 minutes to display graphs and information. If you are running your application in shorter durations, processing takes longer to display information.

Topics

- Apply the CodeGuru Profiler function decorator to your handler function
- Use AWS Lambda layers

Apply the CodeGuru Profiler function decorator to your handler function

Pull your `codeguru_profiler_agent` dependency to your local environment through `pip` and include it in the .zip file for Lambda.

The only required configuration option to start the agent is the profiling group name. You can find this in the **Settings** section of your profiling group on the CodeGuru Profiler console. You can also provide the Region if you want to use a profiling group that was created in a different region than the one where Lambda is running. You can also provide the profiling group ARN directly, which contains both the name and Region. Either the profiling group name or ARN must be provided.

Option	Environment variable key	Environment variable value	Decorator argument example
Profiling group name	`AWS_CODEGURU_PROFILER_GROUP_NAME`	`MyGroupName`	`@with_lambda_profiler(profiling_group_name="MyGroupName")`
Profiling group ARN	`AWS_CODEGURU_PROFILER_GROUP_ARN`	`arn:aws:codeguru-profiler:us-east-1:123456789123:profilingGroup/MyGroupName`	An ARN cannot be passed as a decorator argument

Option	Environment variable key	Environment variable value	Decorator argument example
Region	`AWS_CODEGURU_PROFILER_TARGET_REGION`	`us-east-1`	`@with_lambda_profiler(region_name="us-east-1")`

Decorate your handler function with `@with_lambda_profiler()`. The following example shows what your handler function code looks like with CodeGuru Profiler turned on.

```
from codeguru_profiler_agent import with_lambda_profiler

@with_lambda_profiler(profiling_group_name="MyGroupName")
def handler_name(event, context):
    return "Hello World"
```

Only decorate your handler function. You do not have to decorate other internal functions. You can pass the profiling group name directly in the decorator, or with environment variables.

Use AWS Lambda layers

There are two ways you can use layers to enable CodeGuru in Lambda functions using Python. The preferred method uses a wrapper script and only works for applications that run on Python 3.8 or 3.9. The second method works for Python 3.7, and can also be used for Python 3.8 and 3.9 if you already have a lambda wrapper or if the preferred method otherwise does not work.

For applications that run on Python 3.8 or 3.9 (preferred)

1. Add the CodeGuru Profiler layer to Lambda. Choose **Specify an ARN** and add `arn:aws:lambda:`*`region`*`:157417159150:layer:AWSCodeGuruProfilerPythonAgentLambda`
   For more information on adding a Lambda layer, see AWS Lambda layers.
2. Add the following environment variable: `AWS_LAMBDA_EXEC_WRAPPER=/opt/codeguru_profiler_lambda_exec`

3. Add an environment variable with your profiling group name or ARN. For information on using your ARN, see the table listed in Apply the CodeGuru Profiler function decorator to your handler function.

> **Note**
>
> You can only have one Lambda wrapper script. If you are currently using one, try the solution for applications that run on Python 3.7, 3.8 or 3.9.

For applications that run on Python 3.7, 3.8 or 3.9

1. Add the CodeGuru Profiler layer to Lambda. Choose **Specify an ARN** and `arn:aws:lambda:region:157417159150:layer:AWSCodeGuruProfilerPythonAgentLambda` For more information on adding a Lambda layer, see AWS Lambda layers.
2. Set the environment variable, `HANDLER_ENV_NAME_FOR_CODEGURU` to your handler function.
3. Change the Lambda handler function to `codeguru_profiler_agent.aws_lambda.lambda_handler.call_handler`.
4. Add an environment variable with your profiling group name or ARN. For information on using your ARN, see the table listed in Apply the CodeGuru Profiler function decorator to your handler function.

> **Note**
>
> You can only have up to five layers for a Lambda function. If you are already using five layers, see Apply the CodeGuru Profiler function decorator to your handler function.

Enabling the agent with code

If your application runs on a platform other than Lambda, install `codeguru_profiler_agent` through `pip`.

```
pip install codeguru_profiler_agent
```

You can configure the agent by passing different parameters to the `Profiler` object.

Option	Constructor argument	Details
Profiling group name (required)	`profiling_group_name="MyProfilingGroup"`	The name of the profiling group to send the data into. The Profiling group must exist.
Region	`region_name="eu-west-2"`	Use this if your application is not running in the same region as the one where the profiling group was created.
AWS session	`aws_session=boto3.session.Session()`	The session object that should be used to target the CodeGuru Profiler backends. Use this if you want to use different credentials or region than the default one. See boto3 session for more information.

Start the agent from one place in your application. We recommend you start the agent in your startup code. Only one Profiler object can be started at the time. The following is a runtime example.

```
from codeguru_profiler_agent import Profiler
from boto3.session import Session
...
custom_session = Session(profile_name='dev', region_name='us-east-1')
Profiler(profiling_group_name='MyProfilingGroup', aws_session=custom_session).start()
start_application()
...
```

You can find the sample code for the following examples in Amazon CodeGuru Profiler Python Demo Applications.

The following is an example of a simple application that sets your profiling group name to `MyProfilingGroup`.

```
from codeguru_profiler_agent import Profiler

if __name__ == '__main__':
    Profiler(profiling_group_name='MyProfilingGroup').start()
    start_application()
```

Supported web components

The following topics provide code that you can add to your application to enable the Amazon CodeGuru Profiler agent.

- Django
- Flask
- WSGI servers

Django

Start the profiler in your settings file. This is usually the file that you're setting for `DJANGO_SETTINGS_MODULE`. Then, start your application as usual.

The following is an example that you can add to `settings.py`

```
from codeguru_profiler_agent import Profiler
Profiler(profiling_group_name='MyProfilingGroup').start()
```

Set the following in your `wsgi.py` file. This example is for a module named `mysite`. Your files may be in a different location.

```
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'mysite.settings')
```

Set the following in your `settings.py` file.

```
Profiler(profiling_group_name='MyProfilingGroup').start()
```

Flask

Start the profiler based on the configuration for your web server. For an example with gunicorn, see WSGI servers.

WSGI servers

Start the profiler based on the configuration for your web server.

uWSGI

Configure CodeGuru Profiler in your `wsgi.py` file. Then, start your application by adding the `--enable-threads` and `--lazy-apps` parameters to your uWSGI startup configuration. These are required for CodeGuru Profiler to run in your uWSGI applications.

```
uwsgi --http :8000 --chdir . --wsgi-file wsgi.py --enable-threads --lazy-apps --
workers=4
```

gunicorn

Configure CodeGuru Profiler in your `post-fork` method. Then start the application as usual.

```
def post_fork(server, worker):
    server.log.info('Starting profiler for {} in {}'.format(os.getpid(),
 threading.get_ident()))
    worker.profiler = Profiler(profiling_group_name='MyProfilingGroup')
    worker.profiler.start()
    server.log.info('Profiler started running for worker pid {}: master pid
 {}.'.format(os.getpid(), worker.ppid))
```

Apache

For Apache (httpd) with `mod_wsgi` module, use the same wsgi configuration. Make sure the `wsgi.py` file is configured to be visible in the `httpd.conf` file.

```
<Directory [to_be_replaced]>
<Files wsgi.py>
Require all granted
</Files>
</Directory>
```

Start your application with `apachectl start`.

Enabling the agent from the command line

If your application runs on a platform other than Lambda, install `codeguru_profiler_agent` through `pip`.

```
pip install codeguru_profiler_agent
```

The only required configuration option to start the CodeGuru Profiler agent is the profiling group name. You can find this in the **Settings** section of your profiling group. You can use the credential profile name parameter to have the agent use credentials that are different from the default credentials. For more information about credential profiles, see Shared credential files. You can specify these options as an environment variable or as a command line option.

Option	Environment variable	Command line option
Profiling group name (required)	`AWS_CODEGURU_PROFILER_GROUP_NAME`	`-p, --profiling-group-name`
Region	`AWS_CODEGURU_PROFILER_TARGET_REGION`	`-r, --region`
(Alternative) credential profile name	Not available	`-c, --credential-profile-name`

The following is an example that uses environment variables. In this example, `my_script.py` is your main script that you would otherwise call directly with `python my_script.py`.

```
#!/bin/bash
export AWS_CODEGURU_PROFILER_GROUP_NAME=MyProfilingGroup
export AWS_CODEGURU_PROFILER_TARGET_REGION=us-west-2

python -m codeguru_profiler_agent my_script.py
```

The following is an example that uses command line arguments to specify the configuration options.

```
python -m codeguru_profiler_agent -p MyProfilingGroup -r us-west-2 \
```

```
-c prod-credential-profile my_script.py
```

You can find more details about each command line option by running it with `-h` to display the list of available options.

Profiling Distributed systems

Amazon CodeGuru Profiler offers limited support when implemented on distributed systems like Spark on EMR or Glue jobs running across clusters. In such cases, Profiler is able to profile the application running on the manager node; however, it may not be able to profile the part of the application running on the worker nodes. Please consult with your local technical representative for further clarifications.

Enabling logs

The Amazon CodeGuru Profiler agent uses the `logging` library. It only uses logs at or below the `INFO` level. To see the logs, include `logging.basicConfig(level=logging.INFO)` or `logging.getLogger('codeguru_profiler_agent').setLevel(logging.INFO)` in your handler code.

Working with Amazon EventBridge

This section helps you get started using Amazon EventBridge with Amazon CodeGuru Profiler.

CodeGuru Profiler sends events to EventBridge. An event indicates a change in a recommendation that CodeGuru Profiler identified. CodeGuru Profiler sends a heartbeat event every 24 hours to show the continuity of the event. Events carry CodeGuru Profiler recommendation information as well as metadata for your compute resources. For information on an event lifecycle, see Amazon EventBridge Events.

Events are only sent to EventBridge for Lambda compute platforms. For compute types that are not Lambda, use the `GetRecommendations` API to see changes to recommendations that CodeGuru Profiler identifies.

> **Note**
>
> CodeGuru Profiler doesn't yet support localization. The default locale is `en_US`.

An event bus receives events from a source such as CodeGuru Profiler and routes them to rules associated with that event bus. For more information on event buses, see Event buses.

The following table explains some of the parameters you might find in the `detail` object. For more information, see Amazon EventBridge events.

Parameter	Description
`schema`	The schema version.
`expiresOn`	The ISO 8601 timestamp, after which the event expires.
`sourceUrl`	The CodeGuru Profiler console URL where you can see more details about your recommend ations.
`deduplicationId`	This is used to elimate duplicated copies of events related to the current event. In a typical

Parameter	Description
	event, you should see an ONGOING event every day, followed by a CLOSED event when event closes.
severity	The severity of the event. Only MEDIUM severity events are sent.
status	Shows the status of an event. Possible statuses are ONGOING and CLOSED.
computeInstanceArns	Information about the compute instances that are used for the profiled applications.
recommendation	Shows information such as the recommend ation name, details, and a resolution path.

The following sample event shows a CodeGuru Profiler recommendations state change. The detail-type shows a brief description of the event. The account shows the user's account number.

```
{
    "version": "0",
    "id": "315c1398-40ff-a850-213b-158f73e60175",
    "detail-type": "CodeGuru Profiler Recommendations State Change",
    "source": "aws.codeguru-profiler",
    "account": "012345678912",
    "time": "2019-02-26T19:42:21Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:codeguru-profiler:us-east-1:012345678912:profilingGroup:profiling-
group-name"
    ],
    "detail": {
        "schema": "2021-08-30",
        "expiresOn": "2019-02-27T19:42:21Z",
        "sourceUrl": "/codeguru/profiler/recommendations-report?
endTime=2019-02-27T00%3A00%3A00.000Z&profilingGroupName=profiling-group-name&region=us-
east-1&startTime=2019-02-26T00%3A00%3A00.000Z",
        "deduplicationId": "ada4cvf74uyrponkgk7tpxeo5idgqcrt",
```

```
        "title": {"value": "Issue name"},
        "eventStartTime": "2019-02-26T19:42:21Z",
        "eventEndTime": "2019-02-29T19:50:34Z", //Could be null until event closes
        "severity": "MEDIUM",
        "status": "ONGOING",
        "computeInstanceArns": ["arn:aws:lambda:us-east-1:012345678912:function:lambda-
function"],
        "recommendation" : {
            "name": {
                "value": "Fix issue"
            },
            "description": {
                "value": "Description of the issue"
            },
            "resolutionSteps": {
                "value": "Fix the issue by following step 1, step 2 and step 3"
            },
            "reason": {
                "value": "Observed the issue because it used more CPU than ideal"
            },
        }
    }
}
```

Working with unsupported AWS Regions

To use Amazon CodeGuru Profiler in an AWS Region it doesn't support, you can configure your agent to submit profiles to one of the supported Regions instead. You can specify which AWS Region to submit profiles to by adding a specified region name to your code or adding an environment variable. For example, using CodeGuru Profiler in `eu-west-1` would mean that profiled data would be stored in that Region, even while your application is running in an unsupported region.

You should create the profiling group in the target AWS Region, which might differ from the Region that the application is running in. The application role should have permissions set up to allow profiles to be submitted to the target Region.

The following code examples demonstrate how to configure your agent to get access to the CodeGuru Profiler console from `eu-west-1` when enabling the agent with code or integrating with Lambda. You can do this for any of the supported Regions, by replacing `EU_WEST_1` with the Region name you want.

To submit any profiles to the CodeGuru Profiler API, your host must have internet access.

Enabling the agent with code

The following sections show how to use Amazon CodeGuru Profiler in an unsupported region when enabling the agent with Java or Python code.

Java

If you are enabling the agent with code in Java, add the following lines to your code.

You first import the region package from the AWS SDK by adding this line to the top of your code.

```
import software.amazon.awssdk.regions.Region; // Uses AWK SDK for Java v2
```

Then add `.awsRegionToReportTo(<AWS Region>)` to the code that enables the agent.

Below is a complete example of what your code should look like if you want to configure your agent to get access to the CodeGuru Profiler console from `eu-west-1`.

```
import software.amazon.awssdk.regions.Region; // Uses AWK SDK for Java v2`
```

```
import software.amazon.codeguruprofilerjavaagent.Profiler;

Profiler.builder()
        .profilingGroupName("ExampleAppConsumingCodeGuruProfilerJavaAgent")
        .awsRegionToReportTo(Region.EU_WEST_1)
        .build()
        .start();
```

For more information on enabling the agent with Java code, see Enabling the agent with code (Java).

Python

If you are enabling the agent with code in Python, set the following parameter to the line where you start the agent.

```
region_name='<AWS Region>'
```

Below is an example of what your code should look like if you want to configure your agent to get access to the CodeGuru Profiler console from `eu-west-1`.

```
from codeguru_profiler_agent import Profiler

if __name__ == '__main__':
    Profiler(profiling_group_name='MyProfilingGroup', region_name='eu-west-1').start()
    start_application()
```

For more information on enabling the agent with Python code, see Enabling the agent with code (Python).

Profiling applications that run on AWS Lambda

The following sections show how to use Amazon CodeGuru Profiler in an unsupported region when integrating with AWS Lambda.

Java

If you're using Java 8 on Amazon Linux 2 or Java 11 (Corretto) to profile your application on AWS Lambda, you can set the environment variable `AWS_CODEGURU_PROFILER_TARGET_REGION` to your target region, see below.

```
AWS_CODEGURU_PROFILER_TARGET_REGION='eu-west-1'
```

For more information on using integrating with Java and AWS Lambda, see Profiling your Java applications that run on AWS Lambda.

Python

Using the Amazon CodeGuru Profiler function decorator

If you're applying the Amazon CodeGuru Profiler function decorator to your AWS Lambda handler function, set the following parameter in the function decorator.

```
region_name='<AWS Region>'
```

Below is an example of what your code should look like if you're applying the Amazon CodeGuru Profiler function decorator to your AWS Lambda handler function from `eu-west-1`.

```
from codeguru_profiler_agent import with_lambda_profiler

@with_lambda_profiler(profiling_group_name='MyProfilingGroup', region_name='eu-west-1')
def handler_name(event, context):
    return "Hello World"
```

For more information on the Amazon CodeGuru Profiler function decorator, see Apply the CodeGuru Profiler function decorator to your handler function.

Using AWS Lambda layers

If you're using AWS Lambda layers to profile your Python application, you can set the environment variable `AWS_CODEGURU_PROFILER_TARGET_REGION` to your target region, see below.

```
AWS_CODEGURU_PROFILER_TARGET_REGION='eu-west-1'
```

For more information on using layers, see Use AWS Lambda layers.

Working with profiling groups

A *profiling group* is a set of applications that are profiled together as a unit. Application data is sent by the Amazon CodeGuru Profiler profiling agent to a single profile group. Data from all applications in a profiling group are aggregated and analyzed together.

You manage profiling groups from the **Profiling groups** page in the CodeGuru Profiler console. The page provides a list of your profiling groups and the status. You can also create or delete a profiling group. When you select a profiling group, you can explore your profiling data by using different visualizations.

Topics

- Creating a profiling group
- Deleting a profiling group

Creating a profiling group

Follow the steps in Setting up CodeGuru Profiler to create a profiling group, set permissions, and add CodeGuru Profiler profiling agent dependencies and startup code to your application.

Deleting a profiling group

When you delete a profiling group, the profiling group and recommendations reports are deleted. Application data in the profiling group will be inaccessible.

To delete a profiling group

1. In the navigation pane on the left, choose **Profiling groups**.
2. Select the profiling group to delete.
3. Choose **Actions**, **Delete profiling group**.
4. On the confirmation page, choose **Delete** to delete the profiling group.

After you've deleted your profiling group, remove the CodeGuru Profiler profiling agent code from either the Java or Python applications in the profiling group. You can also modify the code to send data to a different profiling group.

Note

To stop being billed for CodeGuru Profiler and to avoid future charges, you must delete all profiling groups.

Working with visualizations

In Amazon CodeGuru Profiler, you can use visualizations to explore profiling data collected from applications in a profiling group. When a profiling group has enough information to display, you can view an *overview* visualization of the profiling group data.

A visualization is a collection of stack frames that were profiled in the running application. A stack frame contains the state of one method invocation. The name of the method is displayed in the visualization. You can pause over a frame to see its full name and timing details. You can also see the active CPU cost of the method as it exists in the substack of the frame. Frames with the same frame name are highlighted in the rest of the visualization. You can hide a stack frame from the visualization or inspect a specific frame. You can also zoom and search for a function.

The following topics describe how to navigate, filter, and visualize data collected from your running applications.

Topics

- Accessing visualizations
- Types of visualizations
- Exploring visualization data
- Filtering visualization data
- Selecting a custom time range
- Understanding the summary page
- Understanding the heap summary
- Comparing two time ranges

Accessing visualizations

The following instructions show you where to find the visualizations of your profiling group data in the CodeGuru Profiler console.

1. Open the CodeGuru Profiler console.
2. In the navigation pane on the left, under **Profiler**, choose **Profiling groups**.
3. Choose your profiling group to view the summary page for that group.

4. On the summary page, locate the summary panel for the data you want to visualize, and select **Visualize CPU**, **Visualize latency**, or **Visualize heap**, depending on the visualization you wish to see.

   Once you've navigated to the **Visualize** page, you can switch between **Data** and **View** using the drop down menus above the visualization panel.

For more information on visualization types, see Types of visualizations.

Types of visualizations

Amazon CodeGuru Profiler uses three types of visualizations to display profiling data collected from applications.

- An *overview* visualization provides a bottom-up view of your profiling data.
- A *hotspots* visualization provides a top-down view of your profiling data.
- An *inspect* visualization provides a focus view of a named stack frame.

Together, these visualizations can help you identify potential performance issues in your applications. All visualizations use a common set of tools to explore and filter data.

The following topics provide more information about each visualization type.

Topics

- Overview visualizations
- Hotspots visualizations
- Inspect visualizations

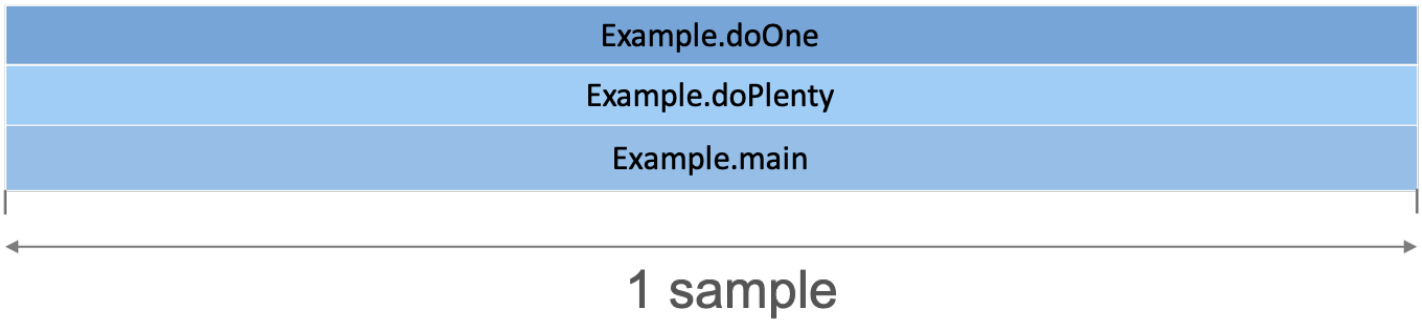
Overview visualizations

An overview visualization provides a bottom-up view of your profiling data. It's similar to reading a stack trace in many IDEs. At the bottom of the visualization are the entry point functions. As you move higher, there are functions that are called deeper in the stack trace. Functions at the top of the visualization are the ones doing basic system operations.

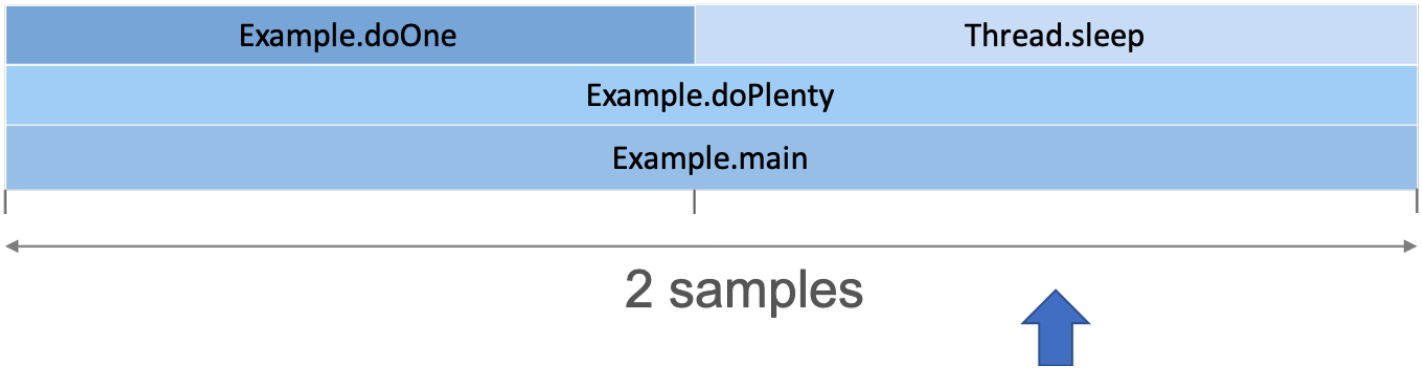
From stack traces to overview visualization

The following example shows how stack trace samples are represented in an overview visualization. Each stack trace that we sample from the profiled application is added to the visualization.

```
Thread main
    java.lang.Thread.State: RUNNABLE
      com.amazon.profiler.demo.Example.doOne()
      com.amazon.profiler.demo.Example.doPlenty()
      com.amazon.profiler.demo.Example.main(String[])
```

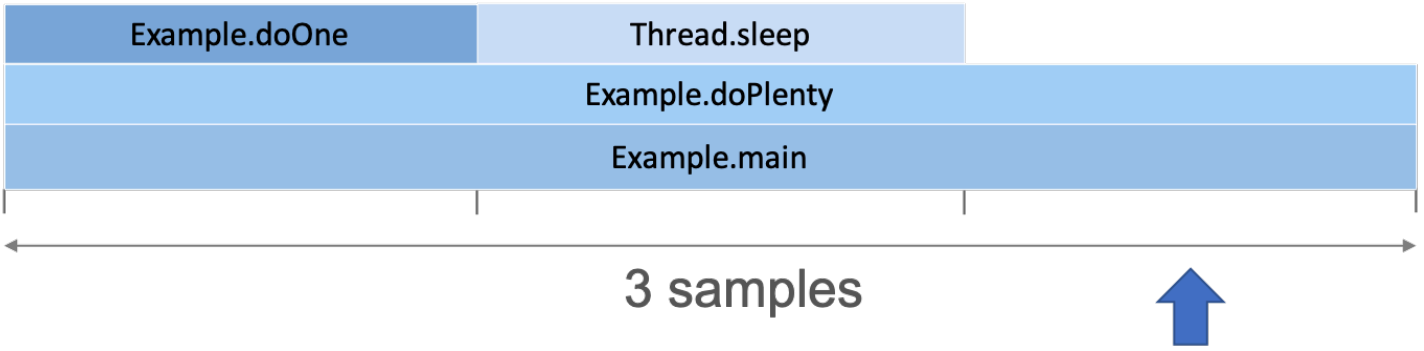



```
Thread main
   java.lang.Thread.State: TIMED_WAITING
     java.lang.Thread.sleep(long)
     com.amazon.profiler.demo.Example.doPlenty()
     com.amazon.profiler.demo.Example.main(String[])
```

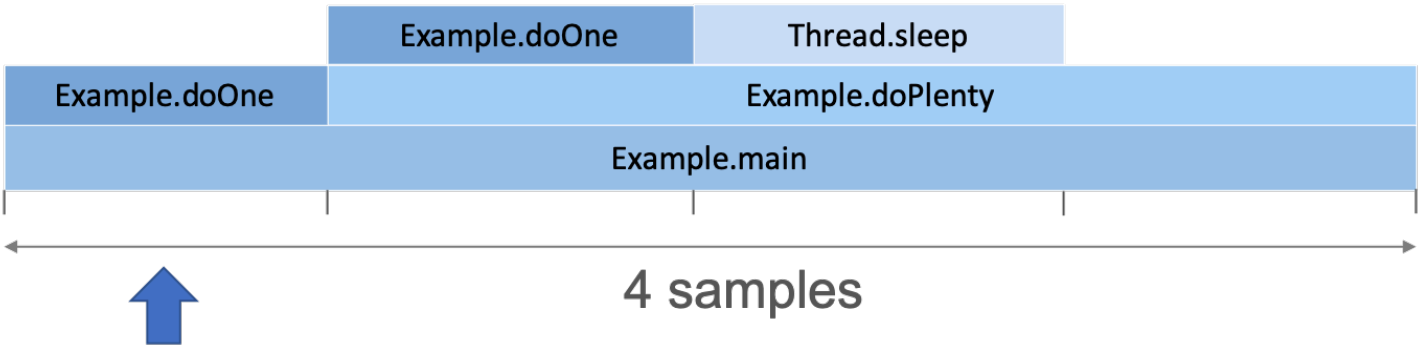



```
Thread main
    java.lang.Thread.State: RUNNABLE
```

```
      com.amazon.profiler.demo.Example.doPlenty()
      com.amazon.profiler.demo.Example.main(String[])
```



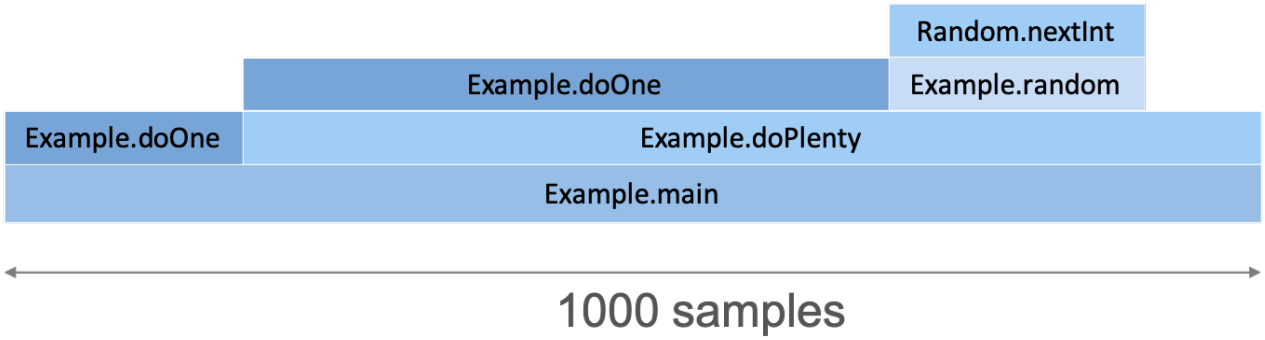

```
Thread main
    java.lang.Thread.State: RUNNABLE
      com.amazon.profiler.demo.Example.doOne()
      com.amazon.profiler.demo.Example.main(String[])
```



As we collect more samples, the functions in which threads spend a lot of time appear wider in the visualization.

What you can learn from overview visualization

An overview visualization can help you find specific call stacks that lead to inefficient code. You can find code that is running on the CPU by looking for flat tops in the visualization. The flat tops are areas where the CPU is doing work directly in that function.



Note

This example of an overview is in CPU view (see Selecting and coloring thread states).

This overview example tells the following:

- The `doOne` function is called inside both `main` function and `doPlenty` function because it appears above both frames.
- More than half of the CPU time spent in `doPlenty` is actually spent in the `doOne` function because the width of `doOne` is more than half the width of `doPlenty`.
- The `doPlenty` function is also doing some basic CPU operations because it has some self time (some width with no callee frames).

The overview example *DOES NOT* tell the following:

- Inside `main` code, the `doOne` function is called before the `doPlenty` function. Frames are ordered alphabetically, and from the visualization, we can't tell in which order the functions are called.
- The `doOne` function is called more often than the `random` function. The overview visualization only tells that more CPU time is spent in `doOne` but CodeGuru Profiler doesn't give any information about the number of times it was called. It might be that it is called less often but is more CPU heavy.
- The `doPlenty` function takes n seconds to execute. CodeGuru Profiler doesn't measure execution time; it only provides estimates of the average CPU time spent in that function over the profile's time range. It's not a duration. A CPU-heavy function that is rarely called and a cheap function that is called many times can look similar in an overview visualization.

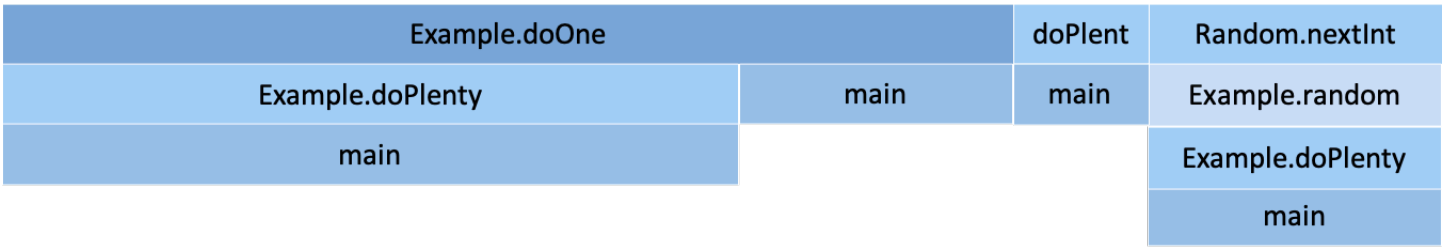
An overview visualization can make it difficult to spot problems with functions that are spread around in multiple stacks. For example, logging calls are often distributed across threads and functions. In these cases, a hotspots visualization might be a better choice.

Hotspots visualizations

A hotspots visualization shows a top-down view of your profile. The functions consuming the most application time are at the top of the visualization. The entry point functions are at the bottom of the visualization.

You can use a hotspots visualization to investigate functions that are by themselves computationally expensive.

Example



This overview example tells the following:

- The do0ne function has two different callers because there are two frames below it.
- Most of the overall CPU time is spent in the do0ne function because it is the majority of the width in the top row.

Inspect visualizations

An inspect visualization is useful to analyze a frame that appears in many places in a visualization. It groups all of the frames with the same name together in the middle of the visualization. Children (callees) are merged into the visualization above the frame. Parents (callers) are merged into the visualization below the frame.

Exploring visualization data

CodeGuru Profiler makes it easy to explore visualization data. You can pause over frames to see information about methods, zoom in on a frame to see more context, and inspect a frame to see the data in an inspect visualization.

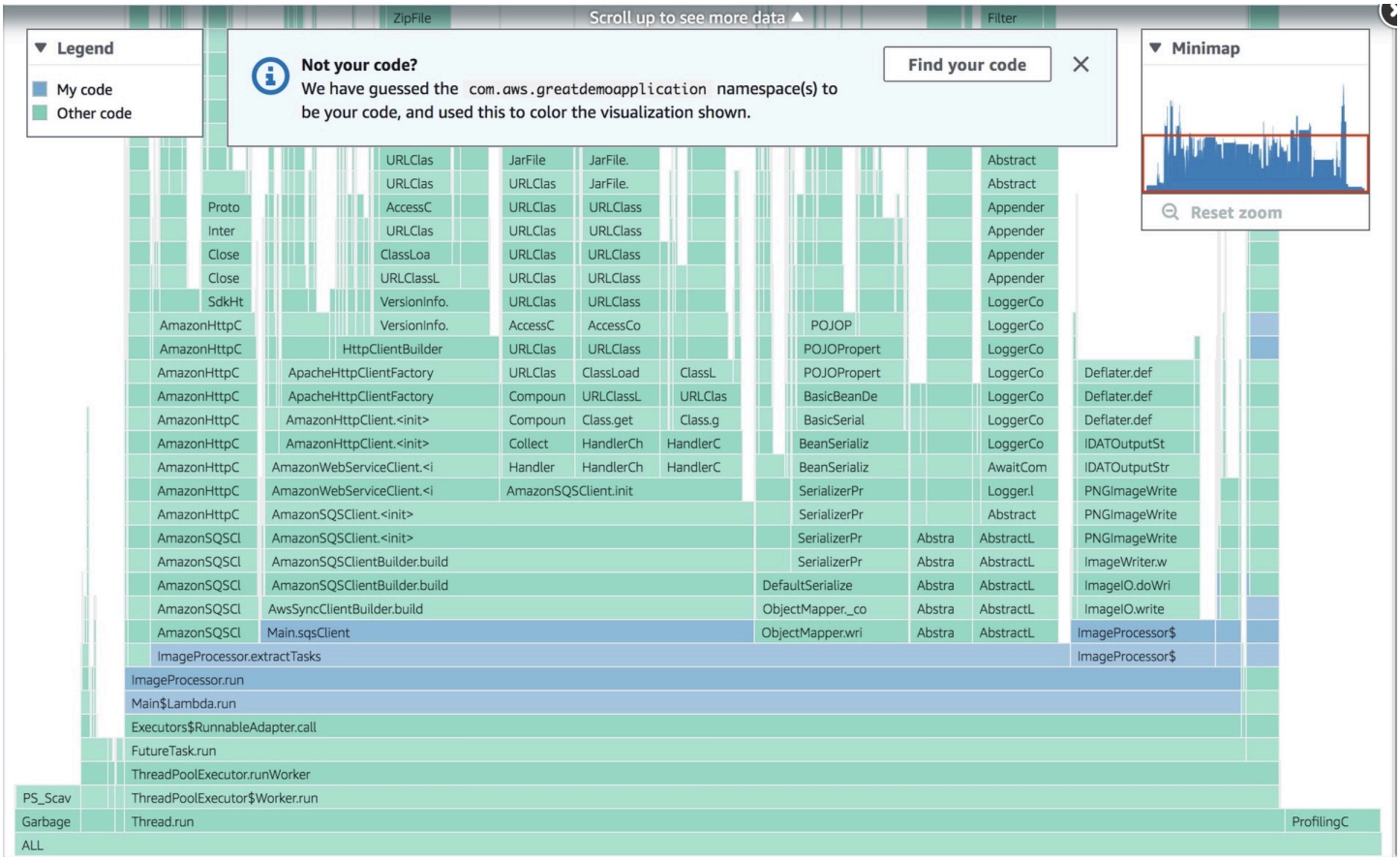
Topics

- Choosing my code in visualizations
- Pausing over a frame
- Zooming in on a frame
- Resetting zoom in a visualization
- Inspecting a frame
- Understanding the dollar estimate of the CPU cost for frames

Choosing my code in visualizations

CodeGuru Profiler differentiates your code in the overview visualization, so you can quickly identify the methods you are working on.

The blue portion of the flame graph highlights your code. The green portion of the graph highlights other code that your application uses, such as libraries and frameworks.



You can change the coloring by manually selecting which package name you want CodeGuru Profiler to identify as your code.

To view your code

1. Choose **Actions**.
2. Choose **Select my code**.
3. Search for the profile namespace that you want to view. The namespaces are sorted based on how much of the overall profiling data they represent.

Pausing over a frame

One of the easiest ways to begin exploring visualization data is by pausing over the visualization. When you pause over a frame, all frames with the same function name are highlighted. This makes it easy to see where and how often the function is called.

You can also see details about the function. CodeGuru Profiler displays the name of the function, how much time it has run on the CPU in that stack frame, and the sample time spent in the selected thread states.

Zooming in on a frame

Clicking a frame zooms in on the function. The frame becomes the new "base" of the visualization. The visualization shows callees of the selected function. Only the functions in the call chain leading to the call of the selected function are displayed.

You can zoom in on any visible frame.

To zoom back out, choose **ALL**. In a hotspots visualization, **ALL** is located at the top of the visualization. In an overview visualization, it's at the bottom.

Resetting zoom in a visualization

You can zoom in to stack frames to view details. To return to the top-most view, you can reset the zoom.

To reset the visualization

- On the **Profiling group detail** page, choose **Actions**, and then choose **Reset zoom**.

Inspecting a frame

You can inspect frames that appear in many places in a visualization. This can happen when your application code has a common set of shared functions.

For example, if you have code that compresses data, you might call it from dozens of functions. If you inspect the compress function, you can see the parent (callers) and children (callee) functions at a glance.

To inspect a frame

1. On the **Profiling group detail** page, pause over the frame you want to inspect on the visualization.
2. Open the context (right-click) menu, and then choose **Inspect frame**.

Understanding the dollar estimate of the CPU cost for frames

Amazon CodeGuru Profiler provides an estimated dollar value for the active CPU cost of a frame. The value is an estimation that can help you understand where your optimization efforts will be most valuable.

To view a frame's dollar estimate, pause over the frame. When you pause over other frames, you see a dollar value estimation that's based on that frame's portion of the total CPU time.

> **Note**
>
> This estimated value does not represent your monthly bill.

The estimated dollar value shown on the **ALL** frame represents the yearly cost of the compute fleet seen during profiling. This is based on the on-demand AWS compute pricing in the AWS Region of your profiling group.

To view information about your compute fleet, choose **Actions**, then choose **Additional profiling data information**.

Filtering visualization data

This section contains information about how to filter profiling data.

Topics

- Selecting and coloring thread states
- Hiding a frame

Selecting and coloring thread states

In a visualization view, you can filter profiling data by thread state. You can color thread states inside of stack frames to make it easy to spot how the application is behaving. You can also select which thread states are displayed.

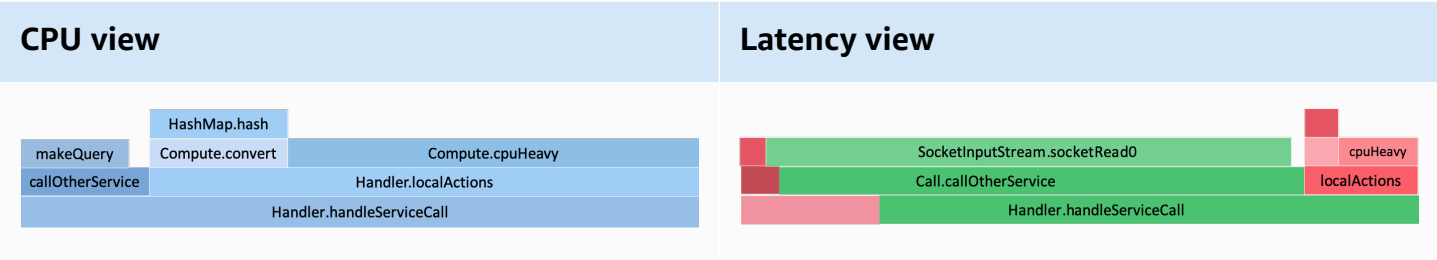
> **Note**
>
> The CPU view and latency view were not supported for first release of Python applications; if you open old Python profiles from before February 2021, the profiling data represents wall clock time percentages for each frame. This is similar to the latency view, without the different thread states and colors. On recent Python profiles the different views work normally.

CPU view – The default thread state view for visualizations, it's useful to try to reduce CPU utilization. It displays frames for thread states that correspond to CPU usage: `RUNNABLE`, `BLOCKED`, and `NATIVE`. In this view, the different shades of coloring simply help with visualization, and are based on the frame names.

Latency view – Useful to try to improve the latency of all or part of your application. When you select it, the visualization displays frames for all of the thread states except `IDLE`. All of these threads might contribute to latency. Frames in the visualization are colored based on the thread state.

Custom view – You can choose to select the thread states for frames to include in the visualization. The threads you can select are the ones found in your profile data. You can also choose whether to color the frames based on thread states.

Example of differences between CPU view and latency view



The `callOtherService` function appears smaller in the **CPU** view because it's not showing the time when the thread was in a waiting state. In the **Latency view**, we still see the part where the CPU was active (in red), but we also see when the threads were waiting (in green).

If you're trying to reduce your CPU usage, the **CPU view** shows you that `localActions` is the most CPU heavy inside `handleServiceCall`, and you might want to optimize this part.

If you're trying to improve the latency of `handleServiceCall`, the **Latency view** shows you that most of the time is spent in `callOtherService`. You can check if this is expected and try to reduce the number of calls or speed up the execution of calls (for example, caching or batching the requests, or calling a closer AWS Region).

Hiding a frame

When you hide a frame, the visualization no longer shows that frame or its callee frames. This is useful when you want to remove certain execution paths from the visualization. For example, you can exclude the `myFunction` function if it's not causing performance issues. All occurrences of that frame in the visualization will be hidden.

To hide a stack frame while pausing over it

1. On the **Profiling group detail** page, pause over the frame you want to inspect on the visualization.
2. Open the context (right-click) menu, and then choose **Hide frame**.

To search for stack frames to hide

1. On the **Profiling group detail** page, choose **Actions**, and then choose **Hide frames**.

2. In the **Hidden frames** page, specify a search string. As the string is provided, results will automatically update.
3. Select a stack frame to hide. When you're done, close the **Hidden frames** page.

To unhide stack frames

1. Choose **X Hidden frames** in the upper-left corner. It opens the **Hidden frames** menu with the list of already hidden frames. **X** is how many frames are currently hidden.
2. Choose **Show** on any of the hidden frames to stop hiding it.

Selecting a custom time range

By default, visualizations display the latest hour of data from the profiling group. You can select a different start time and end time to explore other data for other time ranges. This can be helpful to see how performance has changed over time.

To select a custom time range

1. In the **Profiling group detail** page, at the top of the visualization, select the date/time displayed. For example, **2019-12-04 @ 7:30 - 7:40 PST**.
2. In the **Select a custom time range** page, choose a **Start time**. You can optionally keep the existing start time.
3. Choose an **End time**. You can optionally keep the existing end time.
4. Choose **Confirm** to update the visualization.

If there is not enough data for the selected range, select a different time range. For the CodeGuru Profiler preview, you can reset the time range back to the default by choosing **Profiler**, **Profiling groups** in the navigation pane, and then selecting the profiling group.

Understanding the summary page

The Amazon CodeGuru Profiler summary page displays the status of your profiling group and relevant metrics gathered during profiling. The metrics shown are for any data gathered in the last 12 hours, up to the beginning of the current UTC hour.

Most information is only displayed while the profiling group is enabled. If a profiling group is disabled, or becomes inactive, then metrics and reports are not shown. You can view the following elements in the summary page.

To view the summary page, go to the CodeGuru Profiler console and choose **Profiling groups** in the navigation bar. Then select the profiling group that you want to view.

Profiling group status

This is the latest general information regarding the status of the profiling group.

CPU summary

The **CPU utilization** gives an indication of how much of the instance's CPU resources are consumed by the profiled application. For JVM applications, this gives a percentage of system CPU resources consumed by the JVM. The metric value is an average across all instances reporting data to this profiling group.

A low value (for example < 10%) indicates your application does not consume a large amount of the system CPU capacity. This means there could be an opportunity to use smaller instances or autoscaling to reduce cost, as long as there is nothing else running on the system. A high value (>90%) indicates your application is consuming a large amount of system CPU capacity. This means there is likely value in looking at your CPU profiles and recommendations for areas of optimization. The values are averages over the last full 12 hours.

The **Agent CPU usage** provides an estimate of how much of the system CPU resources are consumed by the CodeGuru Profiler agent on average across profiled instances. It's expected that this value is low (<1%); however, it can be normal for this to be higher depending on the application being profiled. If the number concerns you, please get in touch with AWS Support, or provide feedback at the bottom of the page.

The **Time spent executing code** is a measure of how frequently your application's JVM threads were in the `RUNNABLE` thread state, as a percentage of all thread states except `IDLE`.

A high percentage (>90%) indicates most of your application's time is spent executing operations on the CPU. A very low percentage (<1%) indicates that most of your application was spent in other thread states (e.g. `BLOCKED` or `WAITING`) and there may be more value in looking at the **Latency** visualization, which displays all non-`IDLE` thread states, instead of the **CPU** visualization.

Latency summary

The **Time spent blocked** is a measure of how often your threads are in the `BLOCKED` state, once we exclude all `IDLE` threads. This can happen if your application makes frequent use of synchronized blocks or monitor locks. The **Latency** visualization can help you understand what sections of code are causing threads to block.

Time spent waiting is a measure of how much time your application's thread spent in the `WAITING` and `TIMED WAITING` thread states, as a percentage of all thread states except `IDLE`. This is frequently caused by I/O operations such as network calls or disk operations. The **Latency** visualization can help you understand which sections of code are causing threads to wait.

Heap usage

The **Average heap usage** shows how much of your application's maximum heap capacity is consumed by your application on average across all profiled instances. The percentage shown is the average heap space used compared to the JVM's maximum heap capacity, with the absolute value for the average heap space used shown next to it.

A high percentage (>90%) could indicate that your application is close to running out of memory most of the time. If you wish to optimize this, then the **Heap summary** visualization shows you the object types consuming the most space on the heap. A low percentage (<10%) may indicate that your JVM is being provided much more memory than it actually requires and cost savings may be available by reducing your system memory size, although you should check the peak usage too.

The **Peak heap usage** metric shows the highest percentage of memory consumed by your application seen by the CodeGuru Profiler agent. This is based on the same dataset as seen in the **Heap summary** visualization. A high percentage (>90%) could indicate that your application has high spikes of memory usage, especially if your average heap usage is low.

Choose **Visualize heap** to see your application's heap usage over time. For information on understanding the heap summary, see Understanding the heap summary.

Anomalies

CodeGuru Profiler discovers anomalies by analyzing trends in your profiling data and detecting deviations in that data. Any anomalies found are shown here along with details of which time period is anomalous. Further details can be found in the linked report.

Recommendations

CodeGuru Profiler makes recommendations that you can use to optimize your applications. Any recommendations available are shown here along with details of the estimated impact on your overall application profile. Further details can be found on the linked report.

Understanding the heap summary

The **Heap summary** visualization shows your application's heap usage over time. You can change the time period shown using the time range selector in the top-right. For information on enabling the heap summary data collection feature, see Step 4: Start CodeGuru Profiler in your application.

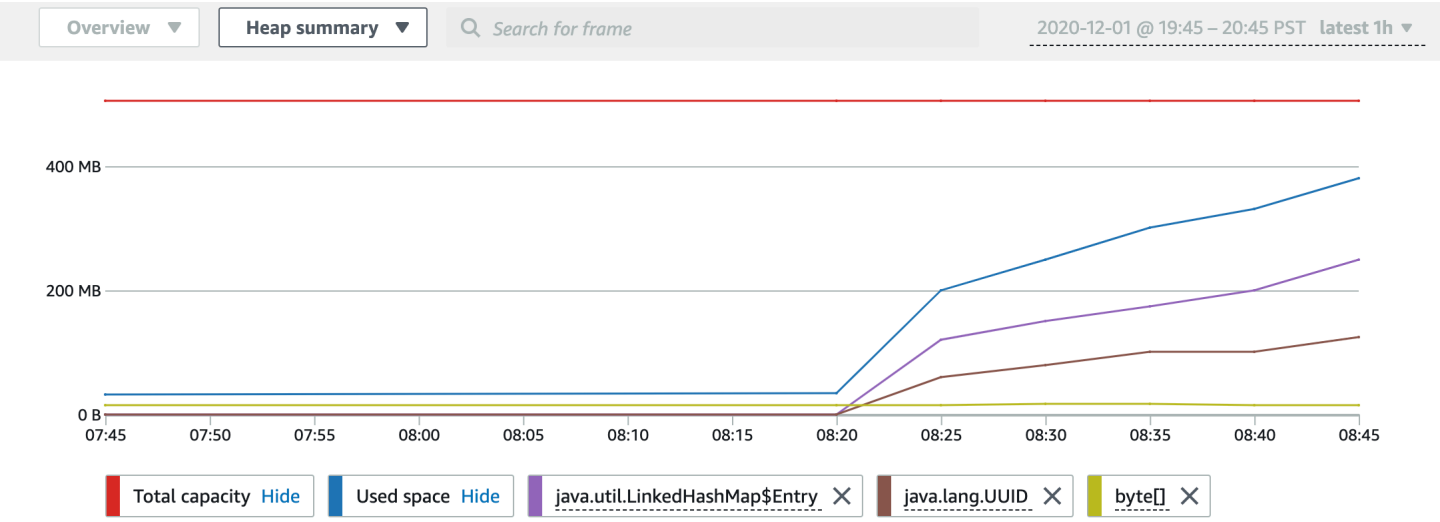
To view your application's heap summary visualization, select your profiling group in the CodeGuru Profiler console, navigate to the **Heap usage** panel, and select **Visualize heap**.

Total capacity

This shows the maximum heap size configured for the JVM. If your application's used space reaches this value then you may run out of memory. This value is equal to your JVM's Xmx value (if configured).

Used space

This shows how much heap space your application requires to store all objects required in memory after a garbage collection cycle. If this value continuously grows over time until it reaches total capacity, then that could be an indication of a memory leak. If this value is very low compared to total capacity, then you may be able to save money by reducing your system's memory.



Heap summary table

The **Heap summary** table shows information about the object types consuming the most heap space in your application. The size of each type only accounts for the shallow memory requirement of that type, and does not include the size of objects referenced by objects of that type. Values are averaged across all hosts reporting data at the same time.

Only object types that consume more than 0.5% (by default) of your heap's total capacity across all objects are detected. Object types with consumption below that threshold are counted as part of the used capacity value, and are not shown individually in the table.

Object types that are below that threshold for at least one data point are shown with an **Incomplete data** badge. The values shown are only averages of the data available, and do not include time periods during which the object type size was lower than the 0.5% capacity threshold.

Object types

- The **Average usage by type** shows how much heap space your application requires to store all objects required in memory after a garbage collection cycle. If this value continuously grows over time until it reaches total capacity, it could indicate a memory leak. If this value is very low compared to total capacity, then you may be able to save money by reducing your system's memory.
- The **Average number of objects** indicates the average number of objects of this type on the heap during the time period.
- Use the **my code** namespace (if configured) to categorize the type into one of several categories. The **my code** namespace can be configured in the **Actions** dropdown list at the top of the page. The table of object types can be filtered based on this code type.

Comparing two time ranges

The CodeGuru Profiler **Compare** option allows you to view differences between two different time ranges of the same profiling group. It can be used for overview, hotspots, and inspect visualizations. This feature is not available for the heap summary visualization.

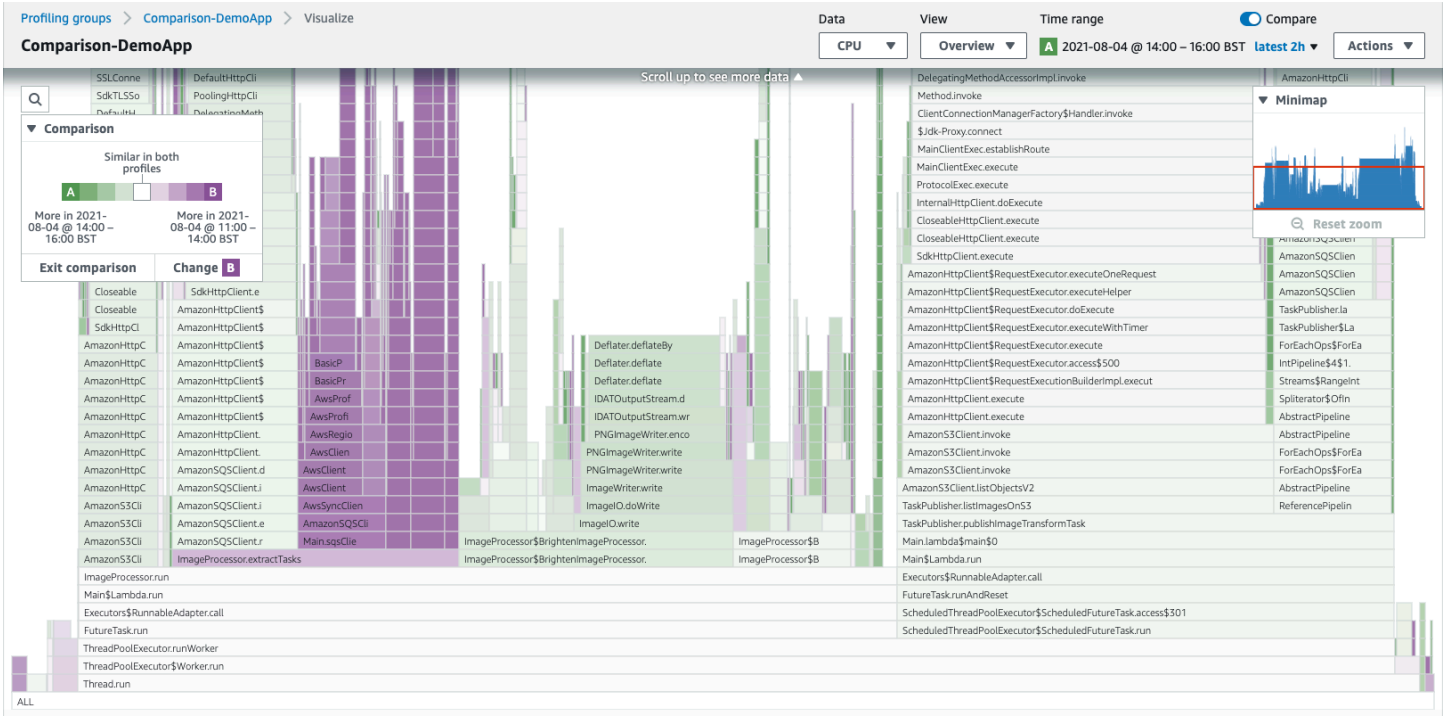
To enable the compare visualization

1. Choose **Compare**.
2. Set **B** to be the time to which you want to compare to your current visualization.

3. Choose **Apply**.

If you'd like to change the time range comparison, choose **Change B**.

These comparisons are visualized by color. Functions that take more time in one time range appear more prominently as the color of the corresponding time range. Less saturated colors indicate a smaller difference between the two time ranges. A very light or white color indicates little to no difference between the time ranges.

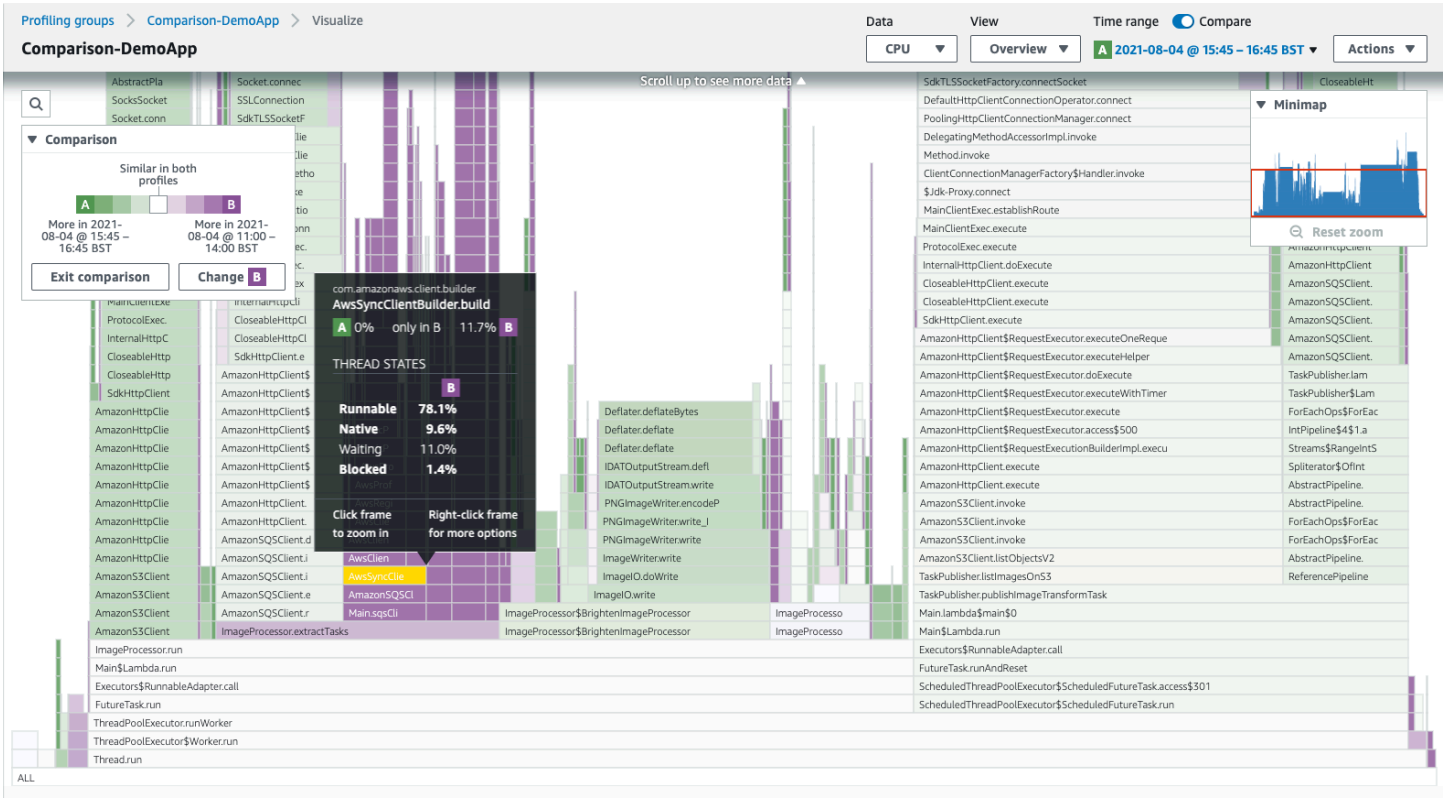


In this example, the two time ranges are A (green) and B (purple). The visualization allows you to pinpoint the areas in darker violet that indicate the function took more time in time range B. Similarly, the darker green indicates functions spending more time in time range A. The more faded sections of the visualization indicate a similar amount of time was taken by the function in both time ranges.

Understanding the comparison

You can see more information about a function if you pause over it. In the following example, you can see the CPU cost reduced betwen the first time range and the second.

> **Note**
>
> It's normal to see changes in your profiling data without any code changes. The profiling data may vary depending on when and how your application runs. Keep in mind your application characteristics when selecting a comparison time range. For example, you might find interesting profiling data if you compare data from your current time range with the data from a week earlier.



Working with anomalies and recommendation reports

Amazon CodeGuru Profiler continuously analyzes your application profiles in real-time to identify potential performance-impacting issues, and also anomalies in your normal application behavior. Each issue identified during analysis results in creating performance improvement recommendations and anomalies which are then included in the recommendation report. The recommendation report is available in the console summary page. If configured, an Amazon SNS notification will also be sent when a new Anomaly is detected.

Viewing reports

Each report contains performance improvement recommendations and anomalies that describe what anomalies were found, and suggests steps to take to resolve the issue.

To view anomaly reports and recommendations

1. Sign in to the Amazon CodeGuru console at https://console.aws.amazon.com/codeguru.
2. In the navigation pane, choose **Profiling groups**.
3. Choose the profiling group with recommendations you want to view.
4. Choose **Actions**, and then choose **View recommendations reports**.
5. In the list of latest reports, choose a report.
6. (Optional) In the report, under **Recommendations**, specify a search string to filter results.
7. (Optional) In the report, at the upper right, choose **View all reports** to open the list of all reports for the profiling group. Select a report to see its recommendations.

You can view reports generated by CodeGuru Profiler up to 30 days in the past.

To view recommendations from the flame graph window

- Choose **<number> Recommendations** next to **Actions** to open the latest report (*<number>* is the number of recommendations found for this profiling group in the latest report).

  <number> Recommendations opens the report that contains the current time range for this profiling group.

Understanding performance improvement recommendations

Each performance improvement recommendation explains why CodeGuru Profiler recommends a change in your code. CodeGuru Profiler gives you suggestions on how and where to improve your code.

CodeGuru Profiler calculates an estimated dollar value for the active CPU cost of the discovered efficiency issue. You can use this to understand where your optimization efforts will be most valuable. For more information on estimated dollar values, see Understanding the dollar estimate of the CPU cost for frames .

Understanding anomaly reports

Anomaly reports can help you to avoid outages, latency, and other performance issues by monitoring application metrics with machine learning models.

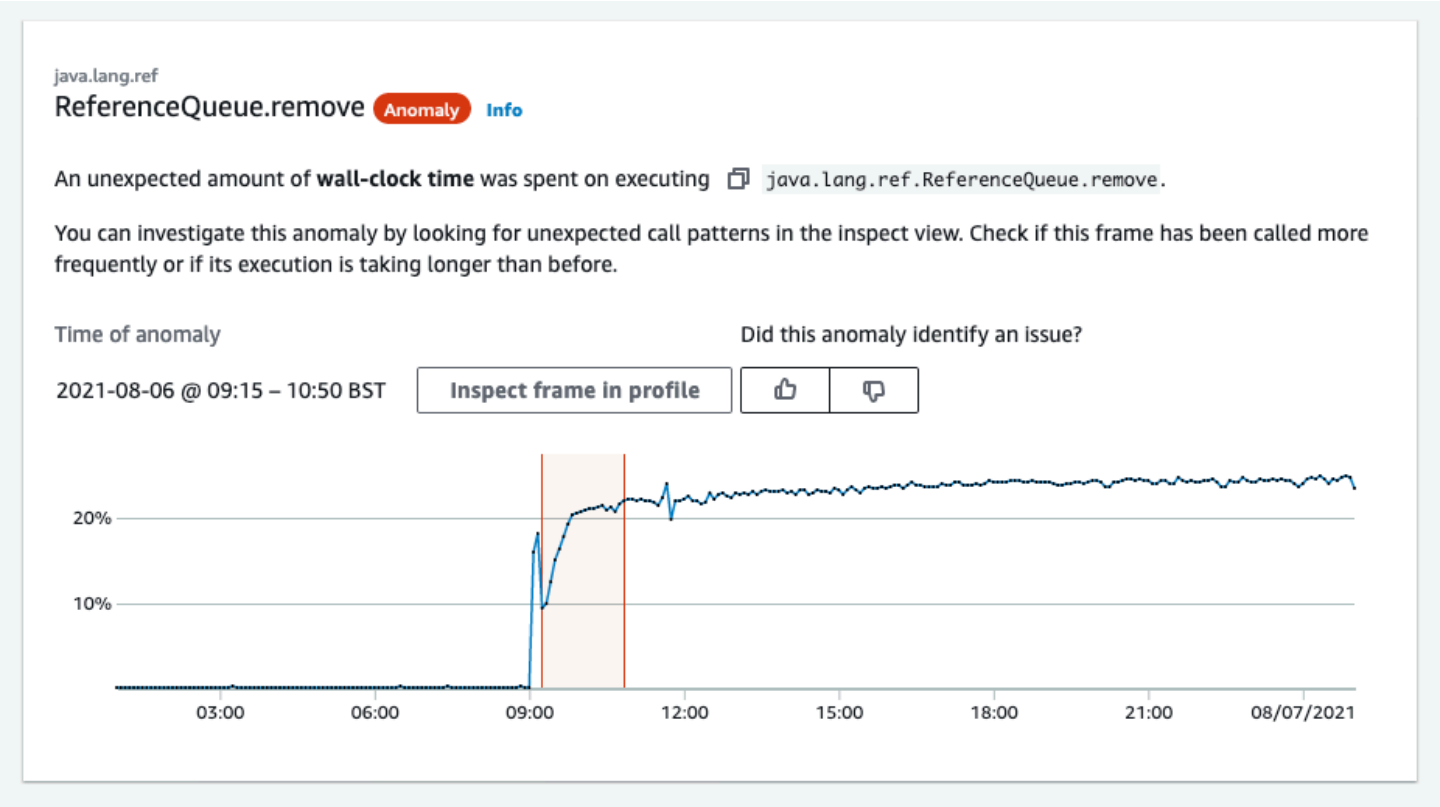
CodeGuru Profiler detects anomalies when your application performance deviates from past behavior. For example, you can have a CPU time anomaly. This is the time that was spent performing calculations and executing instructions. CPU time anomalies are commonly caused by computationally expensive sections of your code.

Another example of an anomaly is a wall clock time deviation. When making external requests or performing I/O, threads are often waiting for these operations to finish. The wall clock time is the sum of the CPU time and the time a thread was blocked from continuing its execution. You can find a list of generated anomalies in your hourly recommendations report. CodeGuru Profiler takes up to 36 hours to generate the first anomaly report.

Each anomaly includes the following information:

- **Frame name** – The frame name provides a brief description of the anomaly. Each title displays the package name, followed by the function name.
- **Why did CodeGuru Profiler trigger this anomaly?** – This section describes why the anomaly was triggered.
- **Graph** – Displays the percentage that represents how frequently this frame occurs, spanning the time the report is requested for. Anomalies are highlighted in red.
- **Show anomalies in inspect view** – Choose this link to go back to the flame graph and see an **overview** visualization for the given frame name.

- **Did this anomaly identify an issue?** – Submit feedback by choosing **thumbs up** or **thumbs down** on an anomaly report. Providing feedback improves the quality of the generated anomalies.



You can set up Amazon SNS notifications to let you know when CodeGuru Profiler generates new anomaly reports. For information about creating and subscribing to an SNS topic, see Getting started with Amazon SNS.

To set up notifications for anomaly detection

1. On the **Profiling groups** page, choose **Edit profiling group**.
2. Choose the **Notifications** tab. Choose from your account's existing SNS topics. This is where you will receive notifications.

Tagging profiling groups

A *tag* is a custom attribute label that you or AWS assigns to an AWS resource. Each AWS tag has two parts:

- A *tag key* (for example, `CostCenter`, `Environment`, `Project`, or `Secret`). Tag keys are case sensitive.
- An optional field known as a *tag value*. Omitting the tag value is the same as using an empty string. Like tag keys, tag values are case sensitive.

Together these are known as key-value pairs.

Tags help you identify and organize your AWS resources. Many AWS services support tagging, so you can assign the same tag to resources from different services to indicate that the resources are related. For example, you can assign the same tag to a profiling group that you assign to an S3 bucket. For more information about using tags, see the Tagging best practices whitepaper.

In CodeGuru Profiler, the primary resource is the profiling group. You can use the CodeGuru Profiler console, the AWS CLI, CodeGuru Profiler APIs, or AWS SDKs to add, manage, and remove tags for a profiling group. In addition to identifying, organizing, and tracking your profiling group with tags, you can use tags in IAM policies to help control who can view and interact with your profiling group.

Topics

- Add a tag to a profiling group
- View tags for a profiling group
- Edit tags for a profiling group
- Remove a tag from a profiling group

Add a tag to a profiling group

Adding tags to a profiling group can help you identify and organize your profiling groups and manage access to them. First, you add one or more tags (key-value pairs) to a profiling group. Keep in mind that there are limits on the number of tags you can have on a profiling group. After you have tags, you can create IAM policies to manage access to the profiling group based on these tags. You can use the CodeGuru Profiler console to add tags to a profiling group.

You can add tags when you create your profiling group on the **Create profiling group** page. You can also add tags to your existing profiling group.

Add a tag to a profiling group

You can use the CodeGuru Profiler console to add one or more tags to an existing profiling group.

1. In **Profiling groups**, choose the name of the profiling group where you want to add tags.
2. Choose **Actions**. Choose **Edit profiling group**.
3. Choose the **Tags** tab.
4. Choose **Add new tag**.
5. In **Key**, enter a name for the tag. You can add an optional value for the tag in **Value**.
6. (Optional) To add another tag, choose **Add new tag** again.
7. When you have finished adding tags, choose **Save**.

View tags for a profiling group

Tags can help you identify and organize your AWS resources and manage access to them. For more information about using tags, see the Tagging best practices whitepaper.

View tags for a profiling group

You can use the CodeGuru Profiler console to view the tags associated with a CodeGuru Profiler profiling group.

1. In **Profiling groups**, choose the name of the profiling group where you want to view tags.
2. Choose **Actions**. Choose **Edit profiling group**.
3. Choose the **Tags** tab.

Edit tags for a profiling group

You can change the value for a tag associated with a profiling group. While you can't change the key, you can deleting a tag and creating a new one with the updated key. Keep in mind that there are limits on the characters you can use in the key and value fields.

> **Important**
>
> Editing tags for a profiling group can impact access to that profiling group. Before you edit the name (key) or value of a tag for a profiling group, make sure to review any IAM policies that might use the key or value for a tag to control access to resources such as profiling groups.

Edit a tag for a profiling group

You can use the CodeGuru Profiler console to edit the tags associated with a profiling group.

1. In **Profiling groups**, choose the name of the profiling group where you want to edit tags.
2. Choose **Actions**. Choose **Edit profiling group**.
3. Choose the **Tags** tab. To change the tag, enter a new name in the **Value**. You cannot change the key of a tag.
4. Do one of the following:
   - To change the value of a tag, enter a new value. If you want to change the value to nothing, delete the current value and leave the field blank.
   - If you want to change the key of a tag, you can remove a tag and add a new one with the updated key name. Find the tag you want to remove, then choose **Remove**. Choose **Add a new tag**. In **Key**, enter a name for the tag. You can add an optional value for the tag in **Value**.
5. When you have finished editing tags, choose **Save**.

Remove a tag from a profiling group

You can remove one or more tags associated with a profiling group. You can also change the name of the key, which is equivalent to removing the current tag and adding a different one with the new name and the same value as the other key. Removing a tag does not delete the tag from other AWS resources that are associated with that tag.

Important

Removing tags for a profiling group can impact access to that profiling group. Before you remove a tag from a profiling group, make sure to review any IAM policies that might use the key or value for a tag to control access to resources such as profiling groups.

Remove a tag from a profiling group

You can use the CodeGuru Profiler console to remove the association between a tag and a profiling group.

1. In **Profiling groups**, choose the name of the profiling group where you want to remove tags.
2. Choose **Actions**. Choose **Edit profiling group**.
3. Choose the **Tags** tab.
4. Find the tag you want to remove, and then choose **Remove**.
5. When you have finished removing tags, choose **Save**.

Security in CodeGuru Profiler

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 Amazon CodeGuru Profiler, 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 CodeGuru Profiler. The following topics show you how to configure CodeGuru Profiler to meet your security and compliance objectives.

Topics

- Data protection for CodeGuru Profiler
- Identity and access management in CodeGuru Profiler
- Compliance validation for Amazon CodeGuru Profiler
- Using CodeGuru Profiler with VPC Endpoints
- Infrastructure security in Amazon CodeGuru Profiler

Data protection for CodeGuru Profiler

The AWS shared responsibility model applies to data protection in Amazon CodeGuru Profiler (CodeGuru Profiler). 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 that you protect AWS account credentials and set up individual user accounts 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 that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources.
- Set up API and user activity logging with AWS CloudTrail.
- 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 personal data that is stored in Amazon S3.

We strongly recommend that 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 CodeGuru Profiler or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into CodeGuru Profiler 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.

Topics

- Captured data in CodeGuru Profiler
- Data encryption in CodeGuru Profiler
- Data retention in CodeGuru Profiler
- Internetwork Traffic Privacy

Captured data in CodeGuru Profiler

The CodeGuru Profiler agent collects stack traces at regular intervals using either the Java virtual machine or Python interfaces. The data is submitted in batches to CodeGuru Profiler.

A stack trace is a sequence of names of functions or methods in execution, followed by the names of functions or methods that called them successively, continuing to the root of the service process.

The CodeGuru Profiler profiling agent doesn't have access to the names or values of function parameters. It also doesn't have access to the values of variables or application data.

Data encryption in CodeGuru Profiler

Encryption is an important part of CodeGuru Profiler security. Data in transit and at rest are provided by default and don't require you to do anything.

- **Encryption of data at rest** - Data collected by CodeGuru Profiler is stored using Amazon S3, Amazon Kinesis, and Amazon DynamoDB and their data-at-rest encryption capabilities.
- **Encryption of data in transit** - All communication between customers and CodeGuru Profiler and between CodeGuru Profiler and its downstream dependencies is protected using TLS connections that are signed using the Signature Version 4 signing process. All CodeGuru Profiler endpoints use SHA-256 certificates that are managed by AWS Private Certificate Authority. For more information, see Signature Version 4 Signing Process and What is ACM PCA.

Data retention in CodeGuru Profiler

Data received from an agent is aggregated into profiles representing five-minute periods. These are then aggregated into hourly and daily profiles. CodeGuru Profiler currently retains five-minute, hourly, and daily profiles for 15 days, 60 days, and three years, respectively.

Internetwork Traffic Privacy

An Amazon Virtual Private Cloud (Amazon VPC) endpoint for CodeGuru Profiler is a logical entity within a VPC that allows connectivity only to CodeGuru Profiler. Amazon VPC routes requests to CodeGuru Profiler and routes responses back to the VPC. For more information, see VPC Endpoints in the *Amazon VPC User Guide*. For information about using Amazon VPC endpoints with CodeGuru Profiler see Using CodeGuru Profiler with VPC Endpoints.

Identity and access management in CodeGuru Profiler

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 CodeGuru Profiler resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Audience in CodeGuru Profiler
- Authenticating with identities in CodeGuru Profiler
- Managing access using policies
- Overview of managing access permissions to your CodeGuru Profiler resources
- Using identity-based policies for CodeGuru Profiler
- Resource-based policies in CodeGuru Profiler
- Amazon CodeGuru Profiler permissions reference
- AWS managed policies for CodeGuru Profiler
- Troubleshooting CodeGuru Profiler identity and access
- Using service-linked roles for CodeGuru Profiler
- Using tags to control access to Amazon CodeGuru Profiler resources

Audience in CodeGuru Profiler

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

- **Service user** - request permissions from your administrator if you cannot access features (see Troubleshooting CodeGuru Profiler identity and access)
- **Service administrator** - determine user access and submit permission requests (see Overview of managing access permissions to your CodeGuru Profiler resources)
- **IAM administrator** - write policies to manage access (see Customer managed policy examples)

Authenticating with identities in CodeGuru Profiler

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

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

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

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you don't use the root user for everyday tasks. For tasks that require root user credentials, see Tasks that require root user credentials in the *IAM User Guide*.

IAM users and groups

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

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

IAM roles

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

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

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For more information about JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

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

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

Identity-based policies

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

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

Resource-based policies

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

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

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 S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

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

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

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

Multiple policy types

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

Overview of managing access permissions to your CodeGuru Profiler resources

Every AWS resource is owned by an AWS account, and permissions to create or access a resource are governed by IAM permissions policies. An account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles).

> **Note**
>
> An account administrator (or administrator user) is a user with administrator privileges. For more information, see IAM best practices in the *IAM User Guide*.

When you grant permissions, you decide who is getting the permissions, the resources they can access, and the actions that can be performed on those resources.

Topics

- CodeGuru Profiler resources and operations
- Understanding resource ownership
- Managing access to resources
- Specifying policy elements: actions, effects, and principals

CodeGuru Profiler resources and operations

In CodeGuru Profiler, the primary resource is a profiling group. In a policy, you use an Amazon Resource Name (ARN) to identify the resource the policy applies to. For more information, see Amazon Resource Names (ARNs) in the *Amazon Web Services General Reference*.

Resource type	ARN format
profiling group	`arn:aws:codeguru-profiler:` *region-ID* `:` *account-ID* `:profilingGroup/` *profiling-group-name*

For example, you can indicate a specific profiling group (*my-profiling-group*) in your statement using its ARN, as follows.

```
"Resource": "arn:aws:codeguru-profiler:us-east-2:123456789012:profilingGroup/my-profiling-group"
```

To specify all resources, or if an API action does not support ARNs, use the wildcard character (*) in the `Resource` element, as follows.

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

To specify multiple resources in a single statement, separate their ARNs with commas, as follows.

```
"Resource": [
     "arn:aws:codeguru-profiler:us-east-2:123456789012:profilingGroup/my-profiling-group",
     "arn:aws:codeguru-profiler:us-east-2:123456789012:profilingGroup/my-other-profiling-group"
   ]
```

CodeGuru Profiler provides a set of operations to work with the CodeGuru Profiler resources. For a list, see the Amazon CodeGuru Profiler permissions reference.

Understanding resource ownership

The AWS account owns the resources that are created in it, regardless of who created the resources. Specifically, the resource owner is the AWS account of the principal entity (that is, the root account,

an IAM user, or an IAM role) that authenticates the resource creation request. The following examples illustrate how this works:

- If you use the root account credentials of your AWS account to create a profiling group, your AWS account is the owner of the CodeGuru Profiler resource.
- If you grant permissions to create CodeGuru Profiler resources to a user, the user can create CodeGuru Profiler resources. However, your AWS account, to which the user belongs, owns the CodeGuru Profiler resources.
- If you create an IAM role in your AWS account with permissions to create CodeGuru Profiler resources, anyone who can assume the role can create CodeGuru Profiler resources. Your AWS account, to which the role belongs, owns the CodeGuru Profiler resources.

Managing access to resources

A permissions policy describes who has access to which resources.

> **Note**
>
> This section discusses the use of IAM in Amazon CodeGuru Profiler. It doesn't provide detailed information about the IAM service. For complete IAM documentation, see What is IAM? in the *IAM User Guide*. For information about IAM policy syntax and descriptions, see the IAM JSON Policy Reference in the *IAM User Guide*.

Policies attached to an IAM identity are referred to as *identity-based policies* (IAM policies). Policies attached to a resource are referred to as *resource-based policies*. CodeGuru Profiler supports identity-based (IAM policies) only.

Identity-based policies

You can attach policies to IAM identities.

- **Attach a permissions policy to a user or a group in your account** – To grant a user permissions to view profiling groups in the CodeGuru Profiler console, you can attach a permissions policy to a user or group that the user belongs to.
- **Attach a permissions policy to a role (grant cross-account permissions)** – You can attach an identity-based permissions policy to an IAM role to grant cross-account permissions. For

example, the administrator in Account A can create a role to grant cross-account permissions to another AWS account (for example, Account B) or an AWS service, as follows:

1. Account A administrator creates an IAM role and attaches a permissions policy to the role that grants permissions on resources in Account A.
2. Account A administrator attaches a trust policy to the role identifying Account B as the principal who can assume the role.
3. Account B administrator can then delegate permissions to assume the role to any users in Account B. Doing this allows users in Account B to create or access resources in Account A. The principal in the trust policy must also be an AWS service principal if you want to grant an AWS service permissions to assume the role.

For more information about using IAM to delegate permissions, see Access management in the *IAM User Guide*.

In CodeGuru Profiler, identity-based policies are used to manage permissions to the resources related to artifact management. For example, you can control access to a profiling group.

You can create IAM policies to restrict the calls and resources that users in your account have access to, and then attach those policies to IAM users. For more information about how to create IAM roles and to explore example IAM policy statements for CodeGuru Profiler, see Identity-based policies.

Specifying policy elements: actions, effects, and principals

For each CodeGuru Profiler resource, the service defines a set of API operations. To grant permissions for these API operations, CodeGuru Profiler defines a set of actions that you can specify in a policy. Some API operations can require permissions for more than one action to perform the API operation. For more information, see CodeGuru Profiler resources and operations and the Amazon CodeGuru Profiler permissions reference.

The following are the basic policy elements:

- **Resource** – You use an Amazon Resource Name (ARN) to identify the resource that the policy applies to.
- **Action** – You use action keywords to identify resource operations to allow or deny. For example, the `codeguru-profiler:DeleteProfilingGroup;` permission gives the user permission to perform the `DeleteProfilingGroup` operation.

- **Effect** – You specify the effect, either allow or deny, when the user requests the action. If you don't explicitly grant access to (allow) a resource, access is implicitly denied. You can also explicitly deny access to a resource. You might do this to make sure that a user cannot access a resource, even if a different policy grants access.
- **Principal** – In identity-based policies (IAM policies), the user the policy is attached to is the implicit principal. For resource-based policies, you specify the user, account, service, or other entity that you want to receive permissions.

To learn more about IAM policy syntax and descriptions, see AWS IAM Policy Reference in the *IAM User Guide*.

For a table showing all of the CodeGuru Profiler API actions and the resources they apply to, see the Amazon CodeGuru Profiler permissions reference.

Using identity-based policies for CodeGuru Profiler

By default, IAM users and roles don't have permission to create or modify CodeGuru Profiler 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 attach policies to an IAM user or group, see Adding and removing IAM identity permissions in the *IAM User Guide*.

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

Topics

- Policy best practices
- Permissions required to use the CodeGuru Profiler console
- Permissions required by the CodeGuru Profiler profiling agent
- Permissions required to access CodeGuru Profiler data
- AWS managed (predefined) policies for CodeGuru Profiler
- Customer managed policy examples

Policy best practices

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

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

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

Permissions required to use the CodeGuru Profiler console

A user who uses the CodeGuru Profiler console must have a minimum set of permissions that allows them to describe other AWS resources for the AWS account. You must have permissions from the following services:

- CodeGuru Profiler
- `ListUsers` and `ListRoles` from the IAM service

If you create an IAM policy that is more restrictive than the minimum required permissions, the console won't function as intended.

Permissions required by the CodeGuru Profiler profiling agent

The CodeGuru Profiler profiling agent is imported into your profiled application. When your application runs, the agent starts in a different thread to profile your application. The following permissions are required to submit data to CodeGuru Profiler:

- `codeguru-profiler:ConfigureAgent`
- `codeguru-profiler:PostAgentProfile`

For more information, see Enabling the agent with code.

The following example policy grants the current AWS user permission to write to a single profiling group in the current AWS Region.

```
{
   "Statement": [{
      "Effect": "Allow",
      "Action": [
         "codeguru-profiler:ConfigureAgent",
         "codeguru-profiler:PostAgentProfile"
      ],
      "Resource": "arn:aws:codeguru-profiler:region-id:aws-account-
id:profilingGroup/profilingGroupName"
   }]
}
```

Permissions required to access CodeGuru Profiler data

Data collected and submitted to CodeGuru Profiler by an agent is used to create application profiles for visualizations:

- `codeguru-profiler:GetProfile`
- `codeguru-profiler:DescribeProfilingGroup`

For more information, see Working with visualizations.

The following is an example.

```
{
   "Statement": [{
      "Effect": "Allow",
      "Action": [
         "codeguru-profiler:GetProfile",
         "codeguru-profiler:DescribeProfilingGroup"
      ],
      "Resource": "arn:aws:codeguru-profiler:region-id:aws-account-id:profilingGroup/
profilingGroupName"
   }]
}
```

For the CodeGuru Profiler console, it can be useful to have an additional policy statement providing `ListProfilingGroups` permissions to allow users to see the list of `ProfilingGroups`. For example, the following allows users to see a list of all profiling groups in their AWS account and Region.

```
{
   "Statement": [{
      "Effect": "Allow",
      "Action": [
         "codeguru-profiler:ListProfilingGroups"
      ],
      "Resource": "*"
   }]
}
```

For more information, see `ConfigureAgent`, `PostAgentProfile`, `GetProfile`, `DescribeProfilingGroup`, and `ListProfilingGroups` in the *Amazon CodeGuru Profiler API Reference*.

AWS managed (predefined) policies for CodeGuru Profiler

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 you can avoid having to investigate what permissions are needed. For more information, see AWS Managed Policies in the *IAM User Guide*.

To create and manage CodeGuru Profiler service roles, you must also attach the AWS managed policy named `IAMFullAccess`.

You can also create your own custom IAM policies to allow permissions for CodeGuru Profiler actions and resources. You can attach these custom policies to the IAM users or groups that require those permissions.

The following AWS managed policies, which you can attach to users in your account, are specific to CodeGuru Profiler.

Topics

- AmazonCodeGuruProfilerFullAccess
- AmazonCodeGuruProfilerReadOnlyAccess
- AmazonCodeGuruProfilerAgentAccess

AmazonCodeGuruProfilerFullAccess

`AmazonCodeGuruProfilerFullAccess` – Provides full access to CodeGuru Profiler, including permissions to create, update, and delete profiling groups. Apply this only to administrative-level users who you want to grant full control over CodeGuru Profiler profiling groups and related resources in your AWS account, including the ability to delete profiling groups.

The `AmazonCodeGuruProfilerFullAccess` policy contains the following statement.

JSON

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

```
    "Statement": [
        {
            "Action": [
                "codeguru-profiler:*",
                "iam:ListRoles",
                "iam:ListUsers",
                "codeguru:*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

AmazonCodeGuruProfilerReadOnlyAccess

`AmazonCodeGuruProfilerReadOnlyAccess` – Grants read-only access to CodeGuru Profiler and related resources in other AWS services. Apply this policy to users who you want to grant the ability to view profiling group visualizations, but not make any changes to them.

The `AmazonCodeGuruProfilerReadOnlyAccess` policy contains the following statement.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "codeguru-profiler:Get*",
                "codeguru-profiler:Describe*",
                "codeguru-profiler:List*",
                "iam:ListRoles",
                "iam:ListUsers",
                "codeguru:Get*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

AmazonCodeGuruProfilerAgentAccess

`AmazonCodeGuruProfilerAgentAccess` – Provides access to CodeGuru Profiler agent to create a Profiling Group, refresh its configuration and submit profiles to the CodeGuru Profiler service.

The `AmazonCodeGuruProfilerAgentAccess` policy contains the following statement.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "codeguru-profiler:ConfigureAgent",
                "codeguru-profiler:CreateProfilingGroup",
                "codeguru-profiler:PostAgentProfile"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

Customer managed policy examples

You can create your own custom IAM policies to allow permissions for CodeGuru Profiler actions and resources. You can attach these custom policies to the IAM users, roles, or groups that require those permissions. You can also create your own custom IAM policies for integration between CodeGuru Profiler and other AWS services.

The following example IAM policies grant permissions for various CodeGuru Profiler actions. Use them to limit CodeGuru Profiler access for your IAM users and roles. These policies control the ability to perform actions by using the CodeGuru Profiler console, API, AWS SDKs, or the AWS CLI.

> **Note**
>
> All examples use the US East (Ohio) Region (us-east-2) and contain fictitious account IDs.

Examples

- Example 1: Allow a user to see all profiling groups and the visualizations of only one profiling group
- Example 2: Allow a user to perform CodeGuru Profiler operations in a single Region
- Example 3: Allow a user connecting from a specified IP address range access to a profiling group

Example 1: Allow a user to see all profiling groups and the visualizations of only one profiling group

The following example policy grants permissions for the AWS user with account ID `123456789012` to see a list of all profiling groups in their AWS account and Region. That user can see visualizations for only one profiling group named `my-profiling-group`.

```
{
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "codeguru-profiler:GetProfile",
            "codeguru-profiler:DescribeProfilingGroup"
         ],
         "Resource": "arn:aws:codeguru-profiler:us-
east-2:123456789012:profilingGroup/my-profiling-group"
      },
      {
         "Effect": "Allow",
         "Action": [
            "codeguru-profiler:ListProfilingGroups"
         ],
         "Resource": "*"
      }
   ]
}
```

Example 2: Allow a user to perform CodeGuru Profiler operations in a single Region

The following permissions policy uses a wildcard character (`"codeguru-profiler:*"`) to allow users to perform all CodeGuru Profiler actions in the us-east-2 Region and not from other AWS Regions.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codeguru-profiler:*",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": "us-east-1"
        }
      }
    }
  ]
}
```

Example 3: Allow a user connecting from a specified IP address range access to a profiling group

You can create a policy that allows users to view a CodeGuru Profiler profiling group only if their IP address is within a certain IP address range. Because the `GetFindingsReportAccountSummary` and `ListProfilingGroups` actions don't support resource-level permissions, their resource is specified as wildcard character (*) in a separate statement. For more information, see the Amazon CodeGuru Profiler permissions reference.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codeguru-profiler:*",
        "iam:ListRoles",
        "iam:ListUsers",
        "codeguru:*"
      ],
```

```
      "Resource": "arn:aws:codeguru-profiler:us-
east-1:123456789012:profilingGroup/DemoProfilingGroup",
      "Condition": {
        "IpAddress": {
          "aws:SourceIp": "203.0.113.0/24"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "codeguru-profiler:GetFindingsReportAccountSummary",
        "codeguru-profiler:ListProfilingGroups"
      ],
      "Resource": "*",
      "Condition": {
        "IpAddress": {
          "aws:SourceIp": "203.0.113.0/24"
        }
      }
    }
  ]
}
```

Resource-based policies in CodeGuru Profiler

You control access to profiling groups in Amazon CodeGuru Profiler using profiling group resource-based policies.

AWS defines a *profiling group* as a *resource* in CodeGuru Profiler. You, as the account administrator, control access to a resource in an AWS service. For profiling groups, resource-based policies support the agent-related actions `ConfigureAgent` and `PostAgentProfile`.

In CodeGuru Profiler, permissions policies are *resource-based policies* that are attached directly to profiling groups. You can use resource-based policies to manage the IAM roles or users that have permission to submit profiling data and configure the agent. You can also grant access with identity-based policies. For an example, see Permissions required by the CodeGuru Profiler profiling agent. For more information about IAM policies, see Identity-based policies and resource-based policies in the *AWS Identity and Access Management User Guide*.

You can use the console, the SDK, or the AWS CLI to specify resource-based permissions on a profiling group

Topics

- Add a resource-based policy to a profiling group (console)
- Add a resource-based policy to a profiling group (AWS CLI)
- Add a resource-based policy to a profiling group (AWS SDKs)

Add a resource-based policy to a profiling group (console)

1. Open the Amazon CodeGuru Profiler console at https://console.aws.amazon.com/codeguru/profiler.
2. In the navigation pane, choose **Profiling groups**.
3. Choose the profiling group to add a resource-based policy to.
4. Choose **Actions**, and then choose **Manage permissions**.
5. From **Application permissions**, select the users and roles you want to grant access to the profiling group.
6. Choose **Save**.

For more information, see Set permissions.

Add a resource-based policy to a profiling group (AWS CLI)

Run the following AWS CLI command to add a resource-based policy to a profiling group. Use your profiling group name and the Amazon Resource Names (ARNs) of the roles and users you want to grant access to the profiling group.

The only valid value for the `action-group` argument is the `agentPermissions` action group. `agentPermissions` grants the `ConfigureAgent` and `PostAgentProfile` permissions on a profiling group to the roles and users listed in the `principals` argument.

```
aws codeguruprofiler put-permission --action-group agentPermissions \
            --profiling-group-name "my-profiling-group-name" \
            --principals "arn:aws:iam::123456789012:user/my-user-name"
```

The following is an example output that grants access to a profiling group named `my-profiling-group` to an AWS user specified using its ARN, `arn:aws:iam::123456789012:user/my-user-name`.

```
{
    "policy": "{\n  \"Version\" : \"2012-10-17\",       \n  \"Statement\" : [ {\n
    \"Sid\" : \"agentPermissions-statement\",\n    \"Effect\" : \"Allow\",\n
  \"Principal\" : {\n      \"AWS\" : \"arn:aws:iam::123456789012:user/my-user-
name\"\n    },\n    \"Action\" : [ \"codeguru-profiler:ConfigureAgent\", \"codeguru-
profiler:PostAgentProfile\" ],\n    \"Resource\" : \"arn:aws:codeguru-profiler:us-
west-2:123456789012:profilingGroup/my-profiling-group-name\"\n  } ]\n}",
    "revisionId": "125820ee-98c7-4df9-8739-442ffad7b3a0"
}
```

Add a resource-based policy to a profiling group (AWS SDKs)

To add a resource-based policy using an AWS SDK, use the `PutPermission` method. For more information, see PutPermission in the *Amazon CodeGuru Profiler API Reference*.

Amazon CodeGuru Profiler permissions reference

You can use AWS-wide condition keys in your CodeGuru Profiler policies to express conditions. For a list, see the IAM JSON Policy Elements Reference in the *IAM User Guide*.

You specify the actions in the policy's `Action` field. To specify an action, use the `codeguru-profiler:` prefix followed by the API operation name (for example, `codeguru-profiler:CreateProfilingGroup` and `codeguru-profiler:GetFindingsReport`). To specify multiple actions in a single statement, separate them with commas (for example, `"Action": [ "codeguru-profiler:CreateProfilingGroup", "codeguru-profiler:GetFindingsReport" ]`).

Using wildcard characters

You specify an ARN, with or without a wildcard character (*), as the resource value in the policy's `Resource` field. You can use a wildcard to specify multiple actions or resources. For example, `codeguru-profiler:*` specifies all CodeGuru Profiler actions and `codeguru-profiler:Get*` specifies all CodeGuru Profiler actions that begin with the word `Get`. The following example refers to all profiling groups with names that begin with `my`.

```
arn:aws:codeguru-profiler:us-east-2:123456789012:profilingGroup/my*
```

You can use the following table as a reference when you are setting up authenticating with identities in CodeGuru Profiler and writing permissions policies that you can attach to an IAM identity (identity-based policies).

CodeGuru Profiler API operations and required permissions for actions

`ConfigureAgent`

Action: `codeguru-profiler:ConfigureAgent`

Required for an agent to register with an orchestration service and retrieve profiling configuration information.

Resource: `arn:aws:codeguru-profiler:`*`region-ID`*`:`*`account-ID`*`:profilingGroup/`*`profiling-group-name`*

`CreateProfilingGroup`

Action: `codeguru-profiler:CreateProfilingGroup`

Required to create a profiling group.

Resource: `arn:aws:codeguru-profiler:`*`region-ID`*`:`*`account-ID`*`:profilingGroup/`*`profiling-group-name`*

`DeleteProfilingGroup`

Action: `codeguru-profiler:DeleteProfilingGroup`

Required to delete a profiling group.

Resource: `arn:aws:codeguru-profiler:`*`region-ID`*`:`*`account-ID`*`:profilingGroup/`*`profiling-group-name`*

`DescribeProfilingGroup`

Action: `codeguru-profiler:DescribeProfilingGroup` Required to get information about a profiling group.

Resource: `arn:aws:codeguru-profiler:`*`region-ID`*`:`*`account-ID`*`:profilingGroup/`*`profiling-group-name`*

`GetFindingsReport`

Action: `codeguru-profiler:GetFindingsReport` Required to get a recommendations report.

Resource: `arn:aws:codeguru-profiler:`*`region-ID`*`:`*`account-ID`*`:profilingGroup/`*`profiling-group-name`*

`GetFindingsReportAccountSummary`

Action: `codeguru-profiler:GetFindingsReportAccountSummary`

Required to get a summary of recent recommendations for each profiling group in an AWS account.

Resource: *

`GetPolicy`

Action: `codeguru-profiler:GetPolicy`

Required to get the resource policy that is associated with a profiling group.

Resource: `arn:aws:codeguru-profiler:`*`region-ID`*`:`*`account-ID`*`:profilingGroup/`*`profiling-group-name`*

`GetProfile`

Action: `codeguru-profiler:GetProfile`

Required to get aggregated profiles for one profiling group.

Resource: `arn:aws:codeguru-profiler:`*`region-ID`*`:`*`account-ID`*`:profilingGroup/`*`profiling-group-name`*

`GetRecommendations`

Action: `codeguru-profiler:GetRecommendations`

Required to get recommendations.

Resource: `arn:aws:codeguru-profiler:`*`region-ID`*`:`*`account-ID`*`:profilingGroup/`*`profiling-group-name`*

`ListFindingsReports`

Action: `codeguru-profiler:ListFindingsReports`

Required to list recommendations reports for one profiling group.

Resource: `arn:aws:codeguru-profiler:`*`region-ID`*`:`*`account-ID`*`:profilingGroup/`*`profiling-group-name`*

`ListProfileTimes`

Action: `codeguru-profiler:ListProfileTimes`

Required to list the start times of profiles for one profiling group.

Resource: `arn:aws:codeguru-profiler:`*`region-ID`*`:`*`account-ID`*`:profilingGroup/`*`profiling-group-name`*

`ListProfilingGroups`

Action: `codeguru-profiler:ListProfilingGroups`

Required to list the profiling groups in one AWS account.

Resource: *

`PostAgentProfile`

Action: `codeguru-profiler:PostAgentProfile`

Required to submit a profile for aggregation.

Resource: `arn:aws:codeguru-profiler:`*`region-ID`*`:`*`account-ID`*`:profilingGroup/`*`profiling-group-name`*

`PutPermission`

Action: `codeguru-profiler:PutPermission`

Required to update the list of principals for an action group in the resource policy of a profiling group.

Resource: `arn:aws:codeguru-profiler:`*`region-ID`*`:`*`account-ID`*`:profilingGroup/`*`profiling-group-name`*

`RemovePermission`

Action: `codeguru-profiler:RemovePermission`

Required to remove the permission of an action group from the resource policy of a profiling group.

Resource: `arn:aws:codeguru-profiler:`*`region-ID`*`:`*`account-ID`*`:profilingGroup/`*`profiling-group-name`*

`UpdateProfilingGroup`

Action: `codeguru-profiler:UpdateProfilingGroup`

Required to update a profiling group.

Resource: `arn:aws:codeguru-profiler:`*`region-ID`*`:`*`account-ID`*`:profilingGroup/`*`profiling-group-name`*

AWS managed policies for CodeGuru Profiler

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

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

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

AWS managed policy: AmazonCodeGuruProfilerFullAccess

You can attach the `AmazonCodeGuruProfilerFullAccess` policy to your IAM identities.

Provides full access to CodeGuru Profiler, including permissions to create, update, and delete profiling groups. Apply this only to administrative-level users who you want to grant full control over CodeGuru Profiler profiling groups and related resources in your AWS account, including the ability to delete profiling groups.

Permissions details

This policy includes the following permissions.

- `codeguru-profiler` – Allows principals full access to all CodeGuru Profiler actions.
- `codeguru` – Allows principals full access to all CodeGuru actions.
- `iam` – Allows principals to list roles and users from IAM

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "codeguru-profiler:*",
          "iam:ListRoles",
          "iam:ListUsers",
          "codeguru:*"
          ],
        "Resource": "*"
      }
    ]
   }
```

AWS managed policy: AmazonCodeGuruProfilerReadOnlyAccess

You can attach the `AmazonCodeGuruProfilerReadOnlyAccess` policy to your IAM identities.

Grants read-only access to CodeGuru Profiler and related resources in other AWS services. Apply this policy to principals who you want to grant the ability to view profiling group visualizations, but not make any changes to them.

Permissions details

This policy includes the following permissions.

- `codeguru-profiler` – Allows principals access to CodeGuru Profiler Describe, Get, List actions.

- `codeguru` – Allows principals access to CodeGuru Get actions.
- `iam` – Allows principals to list roles and users from IAM

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "codeguru-profiler:Get*",
          "codeguru-profiler:Describe*",
          "codeguru-profiler:List*",
          "iam:ListRoles",
          "iam:ListUsers",
          "codeguru:*"
        ],
        "Resource": "*"
      }
    ]
   }
```

AWS managed policy: AmazonCodeGuruProfilerAgentAccess

You can attach the `AmazonCodeGuruProfilerAgentAccess` policy to your IAM identities.

This policy is added to the execution role of AWS Lambda functions onboarded to CodeGuru Profiler via Lambda console's monitoring page. It allows the Profiler agent to create a Profiling Group, refresh its configuration and submit agent profiles to CodeGuru Profiler service.

Permissions details

This policy includes the following permissions.

- `codeguru-profiler` – Allows principals access to CodeGuru Profiler ConfigureAgent, CreateProfilingGroup and PostAgentProfile actions.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "codeguru-profiler:ConfigureAgent",
          "codeguru-profiler:CreateProfilingGroup",
          "codeguru-profiler:PostAgentProfile"
        ],
        "Resource": "arn:aws:codeguru-profiler:*:*:profilingGroup/*"
      }
     ]
    }
```

CodeGuru Profiler updates to AWS managed policies

View details about updates to AWS managed policies for CodeGuru Profiler since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the CodeGuru Profiler Document history page.

Change	Description	Date
AmazonCodeGuruProfilerAgentAccess – Updates to policy	CodeGuru Profiler reduced resource scope in order to improve application security.	July 12, 2021
AmazonCodeGuruProfilerAgentAccess – Updates to policy	CodeGuru Profiler added permissions needed for CodeGuru Profiler agent to Create a Profiling Group.	April 1, 2021

Change	Description	Date
CodeGuru Profiler started tracking changes	CodeGuru Profiler started tracking changes for its AWS managed policies.	March 25, 2021

Troubleshooting CodeGuru Profiler identity and access

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

Topics

- I am not authorized to perform an action in CodeGuru Profiler
- I am not authorized to perform iam:PassRole
- I want to view my access keys
- I'm an administrator and want to allow others to access CodeGuru Profiler
- I want to allow people outside of my AWS account to access my CodeGuru Profiler resources

I am not authorized to perform an action in CodeGuru Profiler

If the AWS Management Console tells you that you're not authorized to perform an action, 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 a profiling group, but does not have `codeguru-profiler:`*`DescribeProfilingGroup`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
            codeguru-profiler:DescribeProfilingGroup on resource: my-example-profiling-
group
```

In this case, Mateo asks his administrator to update his policies to allow him to access the *`my-example-profiling-group`* resource using the `codeguru-profiler:`*`DescribeProfilingGroup`* action.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to CodeGuru Profiler.

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

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

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

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

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

I want to view my access keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

> **Important**
>
> Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your AWS account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing access keys in the *IAM User Guide*.

I'm an administrator and want to allow others to access CodeGuru Profiler

To allow others to access CodeGuru Profiler, you must grant permission to the people or applications that need access. If you are using AWS IAM Identity Center to manage people and applications, you assign permission sets to users or groups to define their level of access. Permission sets automatically create and assign IAM policies to IAM roles that are associated with the person or application. For more information, see Permission sets in the *AWS IAM Identity Center User Guide*.

If you are not using IAM Identity Center, you must create IAM entities (users or roles) for the people or applications that need access. You must then attach a policy to the entity that grants them the correct permissions in CodeGuru Profiler. After the permissions are granted, provide the credentials to the user or application developer. They will use those credentials to access AWS. To learn more about creating IAM users, groups, policies, and permissions, see IAM Identities and Policies and permissions in IAM in the *IAM User Guide*.

I want to allow people outside of my AWS account to access my CodeGuru Profiler 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 CodeGuru Profiler supports these features, see Overview of managing access permissions to your CodeGuru Profiler resources.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.

- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Using service-linked roles for CodeGuru Profiler

Amazon CodeGuru Profiler uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to CodeGuru Profiler. Service-linked roles are predefined by CodeGuru Profiler and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up CodeGuru Profiler easier because you don't have to manually add the necessary permissions. CodeGuru Profiler defines the permissions of its service-linked roles, and unless defined otherwise, only CodeGuru Profiler can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your CodeGuru Profiler resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for CodeGuru Profiler

CodeGuru Profiler uses the service-linked role named **AWSServiceRoleForCodeGuruProfiler**. This is a managed IAM policy with scoped permissions that CodeGuru Profiler needs to run in your account.

The AWSServiceRoleForCodeGuruProfiler service-linked role trusts the following services to assume the role:

- `codeguru-profiler.amazonaws.com`

The role permissions policy allows CodeGuru Profiler to complete the following actions on the specified resources.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowSNSPublishToSendNotifications",
      "Effect": "Allow",
      "Action": [ "sns:Publish" ],
      "Resource": "*"
    }
  ]
}
```

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-Linked Role Permissions in the *IAM User Guide*.

Creating a service-linked role for CodeGuru Profiler

You don't need to manually create a service-linked role. When you configure notifications on any profiling group for the first time, you configure an Amazon SNS topic for forwarding notifications from CodeGuru Profiler to the subscribers of the Amazon SNS topic. When you create the first notification configuration, CodeGuru Profiler automatically creates the IAM service-linked role, which you can see in the IAM console. You don't need to manually create or configure this role.

> **Important**
>
> This service-linked role can appear in your account if you completed an action in another service that uses the features supported by this role. Also, if you were using the CodeGuru Profiler service before January 1, 2017, when it began supporting service-linked roles, then CodeGuru Profiler created the AWSServiceRoleForCodeGuruProfiler role in your account. To learn more, see A New Role Appeared in My IAM Account.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create the first new notification configuration in CodeGuru Profiler, it creates the service-linked role for you again.

In the AWS CLI or the AWS API, create a service-linked role with the CodeGuru Profiler service name. For more information, see Creating a Service-Linked Role in the *IAM User Guide*. If you delete this service-linked role, you can use this same process to create the role again.

Editing a service-linked role for CodeGuru Profiler

CodeGuru Profiler does not allow you to edit the AWSServiceRoleForCodeGuruProfiler service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a Service-Linked Role in the *IAM User Guide*.

Deleting a service-linked role for CodeGuru Profiler

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

> **Note**
>
> If the CodeGuru Profiler service is using the role when you try to delete the resources, the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To delete CodeGuru Profiler resources used by the AWSServiceRoleForCodeGuruProfiler service-linked role

1. Sign in to the AWS Management Console, and then open the CodeGuru Profiler console at https://console.aws.amazon.com/codeguru/profiler.
2. Choose the profiling group that has notification configuration set up.
3. Choose **Actions**, and then choose **Edit profiling group**.
4. Delete all the selected Amazon SNS topics.
5. Repeat steps 2–4 until notification configuration is removed from all the profiling groups.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForCodeGuruProfiler service-linked role. For more information, see Deleting a Service-Linked Role in the *IAM User Guide*.

Supported Regions for CodeGuru Profiler service-linked roles

CodeGuru Profiler supports using service-linked roles in all of the AWS Regions where the service is available. For more information, see AWS Regions and Endpoints.

CodeGuru Profiler does not support using service-linked roles in every Region where the service is available. You can use the AWSServiceRoleForCodeGuruProfiler role in the following Regions.

Region name	Region identity	Support in CodeGuru Profiler
US East (N. Virginia)	us-east-1	Yes
US East (Ohio)	us-east-2	Yes
US West (N. California)	us-west-1	Yes
US West (Oregon)	us-west-2	Yes
Asia Pacific (Mumbai)	ap-south-1	Yes
Asia Pacific (Osaka)	ap-northeast-3	Yes
Asia Pacific (Seoul)	ap-northeast-2	Yes
Asia Pacific (Singapore)	ap-southeast-1	Yes
Asia Pacific (Sydney)	ap-southeast-2	Yes
Asia Pacific (Tokyo)	ap-northeast-1	Yes
Canada (Central)	ca-central-1	Yes
Europe (Frankfurt)	eu-central-1	Yes
Europe (Ireland)	eu-west-1	Yes

Region name	Region identity	Support in CodeGuru Profiler
Europe (London)	eu-west-2	Yes
Europe (Paris)	eu-west-3	Yes
South America (São Paulo)	sa-east-1	Yes
AWS GovCloud (US)	us-gov-west-1	No

Using tags to control access to Amazon CodeGuru Profiler resources

Conditions in IAM policy statements are part of the syntax that you can use to specify permissions for CodeGuru Profiler profiling group-based actions. You can create a policy that allows or denies actions for profiling groups based on the tags associated with those profiling groups, and then apply those policies to the IAM groups you configure for managing IAM users. For information about applying tags to a profiling group, see Tagging profiling groups.

Example 1: Give all CodeGuru Profiler permissions to the role.

The first statement gives all CodeGuru Profiler permissions to all groups with the role. The second statement provides deny permissions to delete any profiling group with tag `{stage: prod}` from the role.

JSON

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

```
                "codeguru-profiler:DeleteProfilingGroup"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/stage": "prod"
                }
            },
            "Resource": "*"
        }
    ]
}
```

Example 2: Deny tagging and untagging a resource.

The following policy prevents a role from tagging or untagging a resource if the resource is marked with the tag `{stage: prod}`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "codeguru-profiler:*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "codeguru-profiler:TagResource",
                "codeguru-profiler:UntagResource"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/stage": "prod"
                }
            },
            "Resource": "*"
        }
```

```
    ]
}
```

Compliance validation for Amazon CodeGuru Profiler

Third-party auditors assess the security and compliance of Amazon CodeGuru Profiler as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. 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 CodeGuru Profiler is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. If your use of CodeGuru Profiler is subject to compliance with standards such as HIPAA, PCI, or FedRAMP, AWS provides resources to help:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- Architecting for HIPAA Security and Compliance Whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Config – This AWS service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub CSPM – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Using CodeGuru Profiler with VPC Endpoints

If you use Amazon Virtual Private Cloud (Amazon VPC) to host your AWS resources, you can establish a private connection between your VPC and Amazon CodeGuru Profiler. You can use

this connection to access and submit profiles to your profiling group without crossing the public internet.

Amazon VPC is an AWS service that you can use to launch AWS resources in a virtual network that you define. With a VPC, you have control over your network settings, such the IP address range, subnets, route tables, and network gateways. With VPC endpoints, the AWS network handles the routing between the VPC and AWS services.

To connect your VPC to CodeGuru Profiler, you define an interface VPC endpoint for CodeGuru Profiler. An interface endpoint is an elastic network interface with a private IP address that serves as an entry point for traffic destined to a supported AWS service. The endpoint provides reliable, scalable connectivity to CodeGuru Profiler —and it doesn't require an internet gateway, a network address translation (NAT) instance, or a VPN connection. For more information, see What Is Amazon VPC in the *Amazon VPC User Guide*.

Interface VPC endpoints are enabled by AWS PrivateLink. This AWS technology enables private communication between AWS services by using an elastic network interface with private IP addresses.

Creating Amazon VPC Endpoints for CodeGuru Profiler

You can create a VPC endpoint to use with CodeGuru Profiler to submit profiles to your profiling group.

To start using CodeGuru Profiler with your VPC, use the Amazon VPC console to create an interface VPC endpoint for CodeGuru Profiler. For instructions, see the procedure "To create an interface endpoint to an AWS service using the console" in Creating an Interface Endpoint. Note the following procedure steps:

- Step 3 –For **Service category**, choose *AWS services*.
- Step 4 – For **Service Name**, choose *com.amazonaws.region.codeguru-profiler* – Creates a VPC endpoint for CodeGuru Profiler operations.

For more information, see Getting Started in the *Amazon VPC User Guide*.

> **Note**
>
> CodeGuru Profiler does not support Amazon VPC endpoint policies at this time.

Infrastructure security in Amazon CodeGuru Profiler

> **Note**
>
> CodeGuru Profiler uses edge optimized endpoints that route traffic through Amazon CloudFront POP (Point of Presence) infrastructure to mitigate DDOS attacks and reduce latency in unsupported AWS Regions. The endpoints communicate with CloudFront over HTTPS/TLS and meet AWS's Security Standards and CloudFront's compliance assurance programs.

As a managed service, Amazon CodeGuru Profiler is protected by 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 CodeGuru Profiler 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.

Logging and monitoring in Amazon CodeGuru Profiler

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon CodeGuru Profiler and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure, if one occurs. AWS provides the following tools for monitoring your CodeGuru Profiler resources and builds and for responding to potential incidents.

Topics

- Logging Amazon CodeGuru Profiler API calls with AWS CloudTrail
- Monitoring Amazon CodeGuru Profiler with Amazon CloudWatch

Logging Amazon CodeGuru Profiler API calls with AWS CloudTrail

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

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

Amazon CodeGuru Profiler information in CloudTrail

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

For an ongoing record of events in your AWS account, including events for CodeGuru Profiler, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default,

when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the S3 bucket that you specify. You can configure other AWS services to further analyze and act on the event data collected in CloudTrail logs. For more information, see the following:

- Overview for Creating a Trail
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple Regions and Receiving CloudTrail log files from multiple accounts

All CodeGuru Profiler actions are logged by CloudTrail. CodeGuru Profiler actions are documented in the Amazon CodeGuru Profiler API Reference. For example, calls to the `CreateProfilingGroup` (in the AWS CLI, `create-profiling-group`) and `UpdateProfilingGroup` (in the AWS CLI, `update-profiling-group`) actions generate entries in the CloudTrail log files. CodeGuru Profiler agent APIs (`ConfigureAgent` and `PostAgentProfile`) are data plane operations and are not logged by default. To log data events, see Logging data events in the *AWS CloudTrail User Guide*.

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

- Whether the request was made with root or IAM user credentials
- Whether the request was made with temporary security credentials for a role or federated user
- Whether the request was made by another AWS service

For more information, see the CloudTrail userIdentity element in the *AWS CloudTrail User Guide*.

Understanding Amazon CodeGuru Profiler log file entries

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

Example: A log entry for calling the DescribeProfilingGroup API

A log entry created by DescribeProfilingGroup includes the name of the profile group in the `requestParameters` field.


```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE:i-1234567890abcdef0",
    "arn": "arn:aws:sts::123456789012:assumed-role/user-name",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/user-name",
        "accountId": "123456789012",
        "userName": "user-name"
      },
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2020-05-06T16:56:59Z"
      },
      "ec2RoleDelivery": "1.0"
    }
  },
  "eventTime": "2020-05-06T18:51:49Z",
  "eventSource": "codeguru-profiler.amazonaws.com",
  "eventName": "DescribeProfilingGroup",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "203.0.113.12",
  "userAgent": "aws-sdk-java/2.9.17 Linux/4.14.154-128.181.amzn2.x86_64 OpenJDK_64-
 Bit_Server_VM/25.252-b09 Java/1.8.0_252 vendor/Amazon.com_Inc. io/sync http/Apache",
  "requestParameters": {
    "profilingGroupName": "ExampleProfilingGroup"
  },
  "responseElements": null,
  "requestID": "cb8c167e-EXAMPLE",
  "eventID": "e3c6f4ce-EXAMPLE",
  "readOnly": false,
```

```
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

Example: A log entry for getting a profile

A log entry created by `GetProfile` includes the name of the profile group and the period in the `requestParameters` field.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE:i-1234567890abcdef0",
    "arn": "arn:aws:sts::123456789012:assumed-role/user-name",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/user-name",
        "accountId": "123456789012",
        "userName": "user-name"
      },
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2020-05-06T16:56:59Z"
      },
      "ec2RoleDelivery": "1.0"
    }
  },
  "eventTime": "2020-05-06T18:51:49Z",
  "eventSource": "codeguru-profiler.amazonaws.com",
  "eventName": "GetProfile",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "203.0.113.12",
  "userAgent": "aws-sdk-java/2.9.17 Linux/4.14.154-128.181.amzn2.x86_64 OpenJDK_64-
 Bit_Server_VM/25.252-b09 Java/1.8.0_252 vendor/Amazon.com_Inc. io/sync http/Apache",
  "requestParameters": {
    "period": "PT5M",
    "profilingGroupName": "ExampleProfilingGroup"
```

```
  },
  "responseElements": null,
  "requestID": "cb8c167e-EXAMPLE",
  "eventID": "e3c6f4ce-EXAMPLE",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

Monitoring Amazon CodeGuru Profiler with Amazon CloudWatch

You can use Amazon CloudWatch to monitor the number of recommendations created on your profiling groups over time. You can set a CloudWatch alarm that notifies you when the number of recommendations on a profiling group exceeds a threshold you set.

For more information about creating and using CloudWatch alarms and metrics, see Using Amazon CloudWatch metrics.

You can track the following metric per profiling group.

Metric	Description
`Recommendations`	The number of recommendations for a profiling group. Units: Count Valid CloudWatch statistic: Maximum Valid CloudWatch period: Hourly

Topics

- Monitoring profiling groups with CloudWatch metrics
- Monitoring profiling groups with CloudWatch alarms

Monitoring profiling groups with CloudWatch metrics

You can view Amazon CodeGuru Profiler metrics in the Amazon CloudWatch console.

To access profiling group metrics

1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**.
3. On the **All metrics** tab, choose **AWS/CodeGuruProfiler**.
4. Choose **ProfilingGroupName**. Metrics for recommendations for all selected profiling groups are displayed in the graph on the page.

Monitoring profiling groups with CloudWatch alarms

You can create an Amazon CloudWatch alarm for your profiling groups to monitor their recommendations.

An alarm watches the number of recommendations for a profiling group over a period of time that you specify. You set one or more actions that happen when the number of recommendations for a profiling group exeeds a count over a number of time periods you choose. For example, you can specify that an Amazon SNS notification is sent when more than five recommendations are generated for a profiling group within an hour.

A user or role must have CloudWatch `PutMetricAlarm` permissions to create an alarm. For more information, see Using identity-based policies for CodeGuru Profiler and Amazon CloudWatch permissions reference in the *Amazon CloudWatch User Guide*.

To create a CloudWatch alarm for CodeGuru Profiler recommendations

1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**.
3. Choose **Create alarm**.
4. Choose **Select metric**.
5. Choose **AWS/CodeGuruProfiler**.
6. Choose **ProfilingGroupName**. Then choose a metric to create an alarm for.

7. Continue through the process to create your alarm.

   For more information about setting up CloudWatch alarms in the CloudWatch console, see Using Amazon CloudWatch alarms in the *Amazon CloudWatch User Guide*.

Troubleshooting

This section helps you troubleshoot common problems you might encounter when working with Amazon CodeGuru Profiler.

Topics

- Why are certain methods missing from my profile?
- I don't see any messages from CodeGuru Profiler in my application logs.
- I get a ResourceNotFoundException in the application logs. The profiling group doesn't exist.
- I received a 403 Forbidden error in the agent. The agent doesn't have permission to submit data.
- I don't see any data in the console.
- There isn't enough data. The profile only has a few frames.
- I don’t see any profiling data for my AWS Lambda function.
- I have errors in my Lambda function log.
- I received a ValidationException error in the agent.
- Why don't I see heap summary data?

Why are certain methods missing from my profile?

The CodeGuru Profiler tool can miss methods that the Java virtual machine (JVM) has chosen to inline for performance reasons. This inlining biases the CodeGuru Profiler data.

Additionally, because CodeGuru Profiler does statistical sampling, methods that are rarely called or are executed quickly might not be sampled in any given time range.

I don't see any messages from CodeGuru Profiler in my application logs.

Make sure CodeGuru Profiler is running. If you are integrating with CodeGuru Profiler by making a code change, make sure you call `.start()` on your `Profiler` object at the beginning of your program. If you are integrating with CodeGuru Profiler without making a code change and are instead using either the JVM command line or Lambda configuration, then make sure all configuration and environment variables are properly set.

Check the logs to make sure the agent is running. When CodeGuru Profiler starts and is configured correctly, one of the following log statements appear.

- Java agent: `Starting the profiler`
- Python agent: `Starting profiler`

After you deploy the CodeGuru Profiler agent to your application, wait 15 minutes for data to arrive.

The following example shows a successful submission of CodeGuru Profiler data when integrating with your JVM-based application.

```
INFO: Attempting to report profile data: ....

INFO: Successfully reported profile
```

The following example shows a successful submission of CodeGuru Profiler data when integrating with your Python-based application.

```
INFO: Attempting to report profile data: ....

INFO: Reported profile successfully
```

I get a ResourceNotFoundException in the application logs. The profiling group doesn't exist.

Make sure you've created a profiling group with the same name that is used in the error through the console or API. Also, be sure you're using the correct AWS Region for the profiler. Do this by running your application in the same Region where you created the profiling group, or by manually configuring the agent to target a given Region.

For more information, see Step 3: Set permissions for CodeGuru Profiler.

I received a 403 Forbidden error in the agent. The agent doesn't have permission to submit data.

Make sure you've given full CodeGuru Profiler permissions to the role with which the agent is running. Make sure the agent is using the right credentials, either through the default credential provider or by explicitly providing the credentials in the builder.

For more information, see Setting up CodeGuru Profiler.

I don't see any data in the console.

Make sure the agent is configured and deployed successfully so that it reports profiles. By default, the CodeGuru Profiler profiling agent profiles for 5 minutes before submitting its first profile. Wait 10–15 minutes after the first profile submission, and check the logs to make sure the agent is running.

There isn't enough data. The profile only has a few frames.

For CodeGuru Profiler to provide statistically valid information, it needs your application to be running under load. We recommend running your application for at least an hour with at least 30% CPU utilization.

I don’t see any profiling data for my AWS Lambda function.

You can profile your Lambda functions running in Java or Python if they are called often enough for CodeGuru Profiler to gather enough data. Invoke your Lambda function several times over a 5 minute period.

Your Lambda function runs the way it typically does, while the CodeGuru Profiler agent runs in parallel. After running for 5 minutes, the agent submits your first profile. Processing can take up to 15 minutes.

I have errors in my Lambda function log.

If you enabled profiling from the Lambda console and continuously see `CreateProfilingGroup` error messages or `ResourceNotFoundException` in your logs, your Lambda function did not run long enough for the `CreateProfilingGroup` API call to succeed. You can manually create a

profiling group with Lambda as your compute platform from the CodeGuru Profiler console. Set the name to `aws-lambda-`*`lambda-function-name`*.

I received a `ValidationException` error in the agent.

If you see the following error in your logs, it means you configured your profiling group for AWS Lambda, but not your agent. For more information on how to configure the agent with Lambda, see Profiling your applications that run on AWS Lambda.

```
software.amazon.awssdk.services.codeguruprofiler.model.ValidationException: Profiling
 group is configured for AWS Lambda compute platform, whereas CodeGuru Profiler agent
  is not running on AWS Lambda. Please refer to CodeGuru Profiler's documentation to
 learn more.
```

Why don't I see heap summary data?

The heap summary feature is supported in **OpenJDK8u262b01+** and all versions of **OpenJDK11**. If your JDK version is not supported, you will see a log message from the CodeGuru Profiler agent such as `Cannot use memory profiling features because JDK Flight Recorder is not available on this JVM.` In this case, heap summary data will not be collected.

If your JDK version is supported, make sure heap summary data collection is enabled.

To enable heap summary data collection by code, add `.withHeapSummary(true)` to the CodeGuru Profiler builder.

To enable heap summary data collection from the command line, refer to the following example.

```
-javaagent:/path/to/codeguru-profiler-java-agent-
standalone-1.2.4.jar="profilingGroupName:myProfilingGroup,heapSummaryEnabled:true"
```

To enable heap summary data collection using environment variables, set `AWS_CODEGURU_PROFILER_HEAP_SUMMARY_ENABLED` to `true`.

Garbage collection

ACPRP uses data collected during garbage collection cycles to provide the heap summary feature. The data gathered depends on your application's garbage collection configuration and activity.

CodeGuru Profiler supports garbage collectors such as Serial, Parallel, CMS, and G1. CodeGuru Profiler currently does not support Shenandoah and ZGC.

Heap summaries are collected during garbage collection cycles, depending on your garbage collection algorithm. This typically corresponds to a 'full' or 'old generation' collection, which is normally the rarest cycle. Depending on your application's characteristics (such as low-traffic environments or applications that don't have long-lived objects) you may never see a full collection. In this case, you do not receive heap summary data.

For applications with low garbage collection activity, you can try selecting a wider time range on the CPU view. Then, check if the heap summary option is available.

You can try to force manual garbage collection in your application periodically by using `System.gc()`. You can also enable garbage collection logging to monitor which collections are happening.

Advanced troubleshooting

Enable the CodeGuru Profiler debugging log. The CodeGuru Profiler agent uses `java.util.logging` (JUL).

The following is an example `log4j` configuration.

```
<Logger name="javaClass" level="TRACE"/>
<Logger name="software.amazon.codeguruprofilerjavaagent" level="TRACE"/>
```

The following is an example `logging.properties` configuration.

```
javaClass.level=ALL
software.amazon.codeguruprofilerjavaagent.level=ALL
```

> **Note**
>
> Including the `javaClass` namespace is important for some messages due to the way some Kotlin classes are compiled.

Enable Java Flight Recorder debug logging for JDK11+. Add `'-Xlog:jfr*=trace'` to your JVM arguments to see more logging information from the Java Flight Recorder subsystem. There are some issues with JDK8 logging, which make it difficult or impossible to enable these debug logs.

Quotas for CodeGuru Profiler

The following tables list the current quotas in Amazon CodeGuru Profiler. These quotas are for each supported AWS Region for each AWS account, unless otherwise specified.

Profiling groups

Resource	Default
Maximum number of profiling groups	500

CodeGuru Profiler user guide document history

The following table describes the major updates and new features for the *Amazon CodeGuru Profiler User Guide*. We also update the documentation frequently to address the feedback that you send us. For notification about updates to this documentation, you can subscribe to an RSS feed.

Latest documentation update: April 12, 2022

Change	Description	Date
New topic	CodeGuru Profiler now integrates with Amazon DevOps Guru by sending events to Amazon EventBridge. For more information, see Working with Amazon EventBridge.	February 16, 2022
New topic	CodeGuru Profiler now allows you to view differences between two different time ranges of the same profiling group. For more information, see Comparing two time ranges.	August 17, 2021
Updated topics	CodeGuru Profiler now supports Python recommendations. For more information, see What languages are supported by CodeGuru Profiler.	August 12, 2021
New tutorials	New CodeGuru Profiler tutorials for creating profiling groups for sample applications. CodeGuru Profiler has	August 9, 2021

	sample applications written in Java and Python that are available to use. For more information, see Getting started.	
Managed Policy updates	CodeGuru Profiler has a managed policy `AmazonCodeGuruProfilerAgentAccess` , which now enables the agent to Create a Profiling Group. For more information, see AWS Managed policies for CodeGuru Profiler.	April 1, 2021
New topics	CodeGuru Profiler now supports a summary analysis of your profiling group. For more information, see Understanding the summary page. CodeGuru Profiler also includes visualizations that help you understand your application's heap utilization over time. For more information, see Understanding the heap summary.	December 8, 2020
New topics	CodeGuru Profiler now supports profiling your Python applications. For more information, see What is CodeGuru Profiler.	December 3, 2020

General availability release	CodeGuru Profiler is now available for general use. You can now profile your applications that run on AWS Lambda. CodeGuru Profiler also generates reports that contain efficiency issues in your code with recommend ations on how to fix them as well as anomaly reports.	June 29, 2020
New topic	CodeGuru Profiler now supports monitoring your profiling groups using Amazon CloudWatch metrics and alarms. For more information, see Monitorin g CodeGuru Profiler with CloudWatch.	June 17, 2020
New topics	This user guide now includes a security section. Learn about data retention, IAM policies, monitoring your profiling groups with AWS CloudTrail, and more. For more information, see Security in Amazon CodeGuru Profiler.	June 7, 2020
Preview release	This is the preview release of the *Amazon CodeGuru Profiler User Guide*.	December 3, 2019

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.